

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

1.0 INTRODUCTION

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.

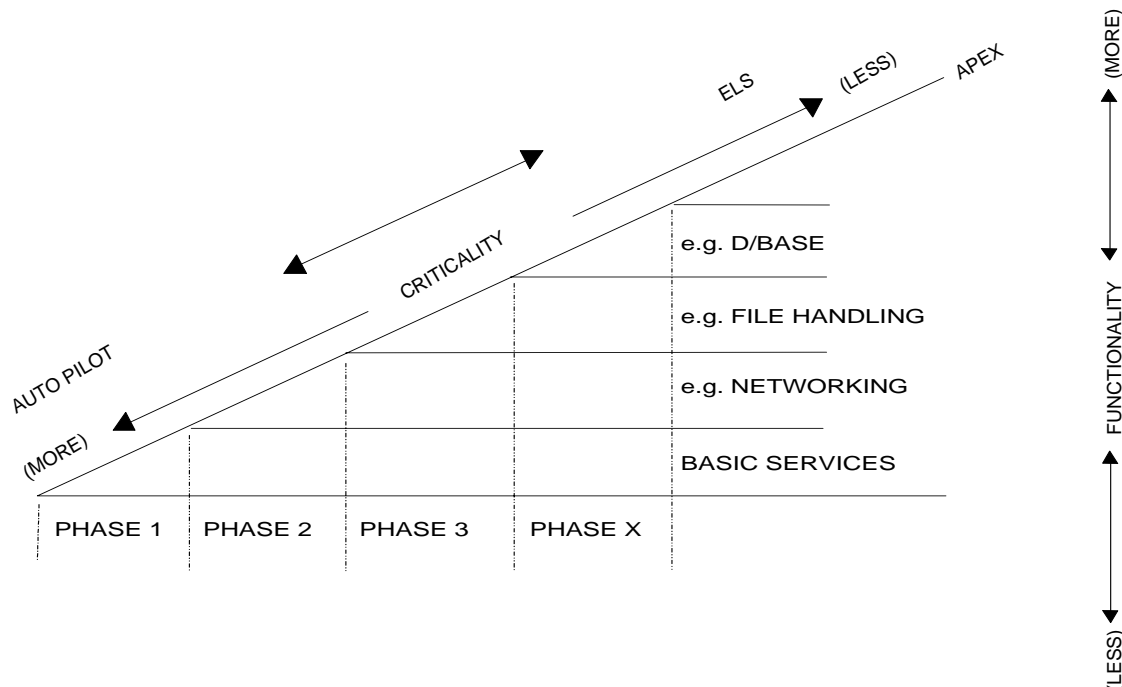


Figure 1.1 - Example of APEX Interface Phased Approach

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.

1.0 INTRODUCTION

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.

1.0 INTRODUCTION

Figure 1.2 shows the relationship between the core module components.

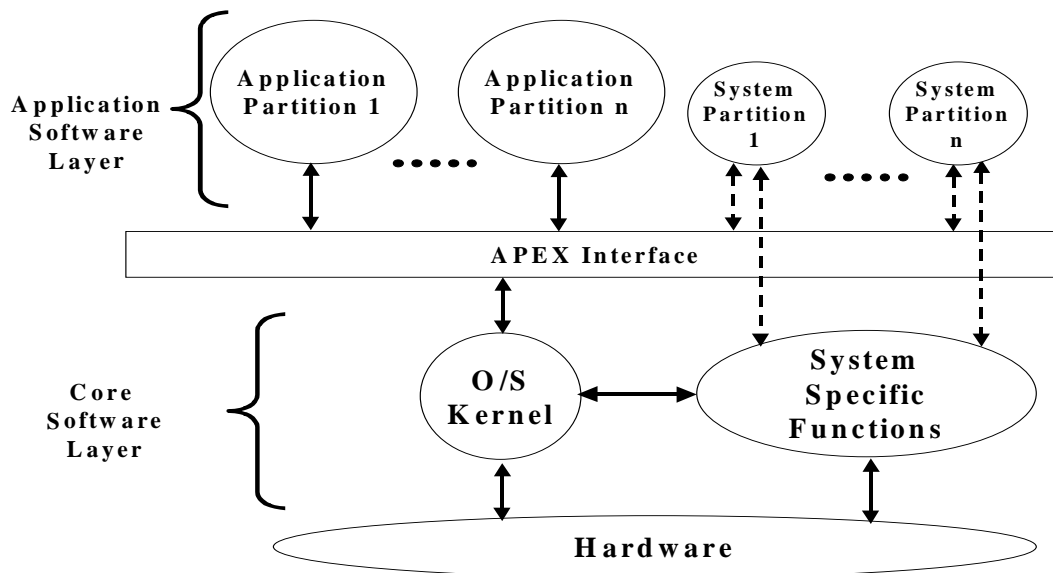


Figure 1.2 - Core Module Component Relationships

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

1.0 INTRODUCTION

development and ownership. To ensure that the APEX interface brings maximum benefit to the industry, the following goals are established:

1. 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.
2. 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.
3. 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.
4. 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.
5. 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:

1. 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.
2. 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.
3. 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.
4. 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.

1.0 INTRODUCTION

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

1.0 INTRODUCTION

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 processor module may not necessarily use the same specification (i.e., in the same language).

1.6 Document Overview

ARINC Specification 653 is 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 - APEX Services Specification Grammar**

1.0 INTRODUCTION

- **Appendix D - The Ada83 and Ada95 language Interface Specification**
- **Appendix E - The C language Interface Specification**
- **Appendix F - (deleted)**
- **Appendix G - Contains a graphical view of the XML configuration schema.**
- **Appendix H - Contains the actual XML configuration schema.**

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 defines the structure of the data needed to specify any ARINC 653 configuration. The XML-Schema is extensible; therefore, the ARINC 653 O/S implementers can extend the schema for a particular implementation.

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.

1.0 INTRODUCTION

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 Avionic Documents

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.

1.0 INTRODUCTION

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

(This section was deleted by Supplement 1, renumbered in Supplement 2).

1.8.5 ARINC Specification 659

(This section was deleted by Supplement 1, renumbered in Supplement 2).

1.8.6 RTCA DO-178/EUROCAE ED-12

RTCA Document DO-178, "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.

1.8.8 RTCA SC-200 / EUROCAE WG-60

RTCA Special Committee SC-200 and EUROCAE WG 60 are chartered with developing guidance to support certification for modular avionics systems. This would include guidance for robust partitioning integrity and possibly core software and API guidance.

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 single function LRU (federated). The software architecture supports partitioning in accordance with the IMA philosophy.

Figure 2.1 illustrates the System Architecture.

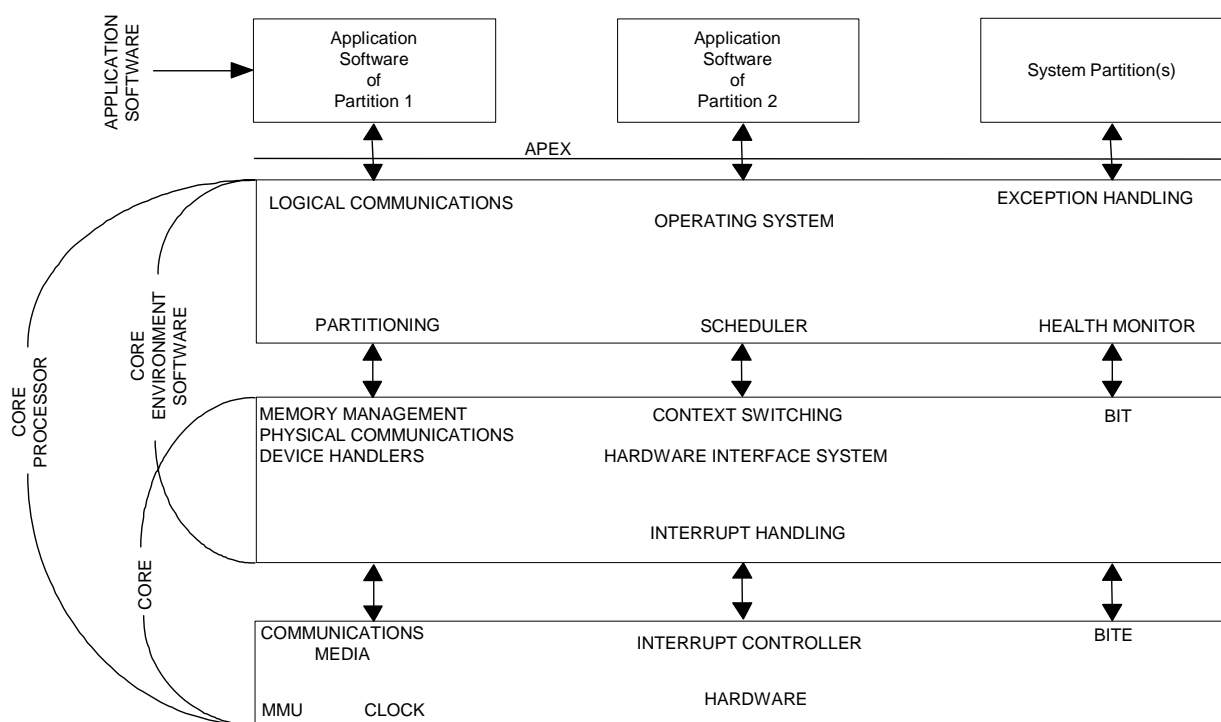


Figure 2.1 - System Architecture

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. The processes of a partition (and therefore the partition itself) cannot be distributed over multiple processors, either in the same core module or in different core modules. Application software should be portable between core modules and between individual processors of a core module without modifying its interface with the O/S.

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,

2.0 SYSTEM OVERVIEW

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 on a core module and between core 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.

2.0 SYSTEM OVERVIEW

2.3.1 Partition Management

Central to the ARINC 653 philosophy is the concept of partitioning, whereby the functions resident on a core 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 core module, without affecting one another spatially or temporally.

The core module O/S is responsible for enforcing partitioning and managing the individual partitions within that core module.

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 being defined by its offset from the start of the major time frame and expected duration. The order of partition activation is defined at configuration time using configuration tables. This provides a deterministic scheduling methodology whereby the partitions are furnished with a predetermined amount of time to access processor resources. Temporal partitioning therefore ensures each partition uninterrupted access to common resources during their assigned time periods.

A core module may contain several partitions running with different periods. The major time frame is defined as a multiple of the least common multiple of all partition periods in the module. Each major time frame contains identical partition scheduling windows. The periodic requirement of each partition on the module must be satisfied by the appropriate size and frequency of partition windows within the major time frame.

The release point for a partition is the start of the partitions' period.

For periodic processes, the first release point for the process is relative to the start of its partition's first window in the next period of the partition. For aperiodic processes, the release point for the process is relative to the current time.

COMMENTARY

Temporal partitioning is influenced by the core module level O/S overhead. Backplane bus acknowledgements and time-outs may interrupt one partition even though the events relate to a different partition. As a result, this may impact on the determinism associated with the application duration.

Each partition has predetermined areas of memory allocated to it. These unique memory spaces are identified based upon the requirements of the individual partitions, and vary in size and access rights. At most, one partition only has write access to any particular area of memory. Memory partitioning is ensured by prohibiting memory accesses (at a minimum, write access) outside of a partition's defined memory areas.

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.

2.0 SYSTEM OVERVIEW

There are limited partition management services available to the application software.

2.3.1.1 Partition Attribute Definition

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

Partition attributes are as follows:

FIXED ATTRIBUTES

1. Identifier - uniquely defined on a system-wide basis, and used to facilitate partition activation and message routing.
2. Memory Requirements - defines memory bounds of the partition, with appropriate code/data segregation.
3. Period - defines the activation period of the partition, and is used to determine the partition's runtime placement within the core module's overall time frame.
4. Duration - the amount of processor time given to the partition every period of the partition.
5. Criticality Level - denotes criticality level of partition.
6. Communication Requirements - denotes those partitions and/or devices with which the partition communicates.
7. Partition Health Monitor Table (health monitor reconfigurations) - denotes instructions to the HM on the actions required.
8. Entry Point (i.e., partition initialization) - denotes partition elaboration restart address.
9. System Partition – denotes the partition is a system partition.

VARIABLE ATTRIBUTES

1. Lock level – denotes the current lock level of the partition.
2. Operating Mode – denotes the partition mode.
3. Start condition – denotes the reason the partition is started.

2.3.1.2 Partition Control

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

2.3.1.3 Partition Scheduling

Scheduling of partitions is strictly deterministic over time. Based upon the configuration of partitions within the core 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 window.

2.0 SYSTEM OVERVIEW

The schedule is fixed for the particular configuration of partitions on the processor. The main characteristics of the partition scheduling model are:

1. The scheduling unit is a partition.
2. Partitions have no priority.
3. The scheduling algorithm is predetermined, repetitive with a fixed periodicity, and is configurable by the system integrator only. At least one partition window is allocated to each partition during each cycle.

COMMENTARY

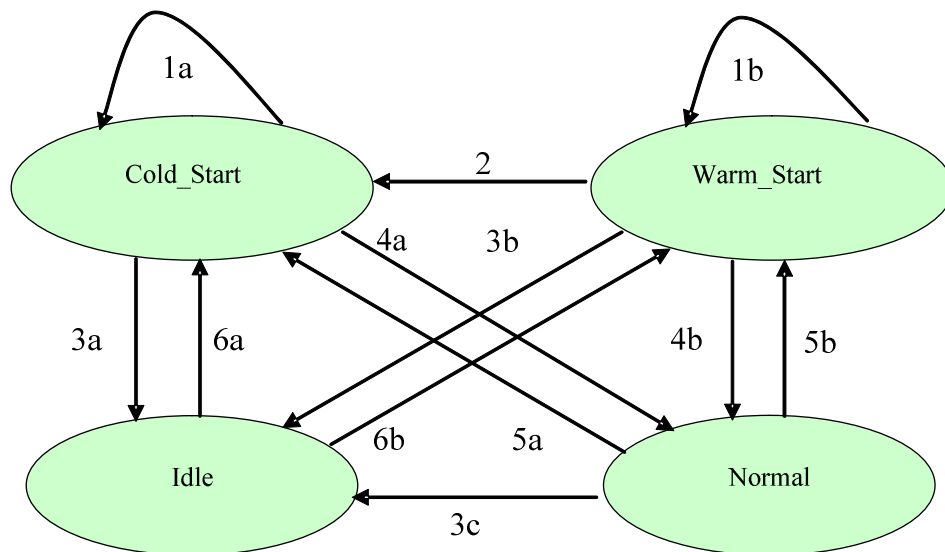
The usage of the ARINC 659 backplane data bus requires the system integrator to schedule the message exchanges amongst all of the partitions in each core module sharing the bus. Synchronizing partition message generation/processing within the partition's activation period implies partition scheduling driven by the backplane bus. Altering an established bus schedule could potentially affect partition scheduling on all of the core modules sharing the bus.

4. The core module level O/S exclusively controls the allocation of the resources to the partition.

2.3.1.4 Partition Modes

The SET_PARTITION_MODE service allows the partition to request a change to its operating mode. The Health Monitor, through its health monitoring configuration data, can also instigate mode changes. The current mode of the partition is available with the GET_PARTITION_STATUS service.

Partition modes and their state transitions are shown in the following diagram:



2.0 SYSTEM OVERVIEW

2.3.1.4.1 Partition Modes Description

IDLE:	In this mode, the partition is not executing any processes within its 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 time windows allocated to the partition are unchanged.
NORMAL:	In this mode, the process scheduler is active. All processes have been created and those that are in the ready state are able to run. The system is in an operational mode.
COLD_START:	In this mode, the initialization phase is in progress, preemption is disabled with LOCK_LEVEL=0 (process scheduling is inhibited) and the partition is executing its respective initialization code.
WARM_START:	In this mode, the initialization phase is in progress, preemption is disabled with LOCK_LEVEL=0 (process scheduling is inhibited) and the partition 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 program 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 program 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 in the main program (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 the 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.**

2.0 SYSTEM OVERVIEW

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 its unique Entry Point (the same Entry Point for COLD_START and WARM_START).

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 gives the partition the cause of entering the COLD_START or WARM_START mode. This parameter allows the partition to manage its own processing environment on initialization error depending on its own start context (e.g., cannot create a 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 recovery action the HM has to make is WARM_START then a COLD_START is still 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.

2.0 SYSTEM OVERVIEW

(5a) NORMAL -> COLD_START

(5b) NORMAL -> WARM_START

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

(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, hardware module reset, or application reset, if an external means exist.

2.3.2 Process Management

Within the IMA 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.

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.

Although enforcing partitioning between constituent processes of a partition is not an IMA requirement, the APEX interface does not preclude designs that operate in this capacity.

The partition level O/S is responsible for managing the individual processes within that partition.

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 internal processes. The processes are not visible outside of the partition.

2.3.2.1 Process Attribute Definition

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

2.0 SYSTEM OVERVIEW

A tabular description of the process attributes is given below. Fixed attributes are statically defined and cannot be changed once the partition has been loaded. Variable attributes are defined with initial values, but can be changed through the use of service requests once the system has been started.

FIXED ATTRIBUTES

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 runtime stack of the process. Depending on the selected architecture, multiple stack requirements may be required for each process.
4. Base Priority - Denotes the capability of the process to manipulate other processes.
5. Period - Identifies the period of activation for a periodic process. A distinct and unique value should be specified to designate the 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”. For a “hard” deadline, failure of the process to meet that deadline within the specified time period will cause the O/S to take some remedial action. Failure of a process to meet a “soft” deadline is less severe and would typically result in the failure being recorded and processing being allowed to continue.

VARIABLE ATTRIBUTES

1. Current Priority - Defines the priority with which the process may access and receive resources. It is set to base priority at initialization time and is dynamic at runtime.
2. Deadline Time - The deadline time is periodically evaluated by the operating system to determine whether the process is satisfactorily completing its processing within the allotted time.
3. Process State - Identifies the current scheduling state of the process. The state of the process could be either 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., starting address, 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.

COMMENTARY

Since the information is unique to each partition, and should therefore be provided by the partition designers, it seems appropriate to collect it from a data structure (“Process List”) that is loaded as part of the partition. One concern is that this method is not portable from the partition/process standpoint.

If all the information is provided by the user directly via a “hard-coded” module, it appears that the information specific to a particular load may restrict the usage of that load to one system.

However, if this module is separately loadable from the actual code, such that a change to the attribute requirements of a process requires no modifications (or re-certification/testing) to the actual code, the portability argument loses its effectiveness. Similarly, the linker should provide certain attribute information for the initial process of the partition anyway, so this partition-supplied

2.0 SYSTEM OVERVIEW

data table should be viewed as nothing more than an extension of this function, with no detrimental effects to portability issues.

2.3.2.2 Process Control

The system resources required 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 at 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 the life of the partition.

A partition should be able to 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. Both types of processes can co-exist in the same partition. Any process could be preempted at any time by a process with a higher current priority. The process in the ready state with the highest (most positive) current priority is always executing while the partition is active.

Preemptibility control allows sections of code to be guaranteed to execute without preemption by other processes within the partition. If a process within a section of code 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 the preferred development language 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 is not 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 have to be 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 parameterless 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.

2.0 SYSTEM OVERVIEW

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.

COMMENTARY

The advantage of creating the processes and mechanisms at system initialization is that the system has a much higher degree of determinism. With this method, it can be established that the processes required for a partition will always have the necessary resources, stack space, etc. Also, the memory management function of the O/S is much more straightforward. By defining the memory management requirements of a partition and its processes at initialization, the system can provide process level protection of resources. This level of protection could not easily be provided if processes were created during runtime, since it would require dynamic interaction with MMU hardware and its data structures when a process is created. This interaction would complicate the certification effort of the O/S, while increasing the duration for that effort.

Another considerable advantage to creating processes only during system initialization concerns memory fragmentation. Since the primary advantage to destroying and creating processes and other mechanisms during runtime would be to utilize a limited stack and/or data area, extensive fragmentation of this space would be likely due to the unique requirements of each process or data space. This would levy additional requirements on the O/S, increasing its size and complexity, thereby potentially increasing certification effort. By allocating these spaces at initialization, the considerable task of dealing with memory fragmentation is not required.

A controlled and deterministic method for the runtime system to create and initialize processes according to their defined attributes needs to be specified, including any restrictions on either the O/S or the processes themselves.

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 deallocated).

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 2.2 shows the possible process state transitions and 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 - process ineligible to receive resources. A process is in the dormant state before it is started and after it is terminated (or stopped).

2.0 SYSTEM OVERVIEW

2. Ready - process eligible for scheduling. A process is in the ready state if it is able to be executed.
3. Running - process 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 at any time.
4. Waiting - process not allowed to receive resources until a particular event 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)

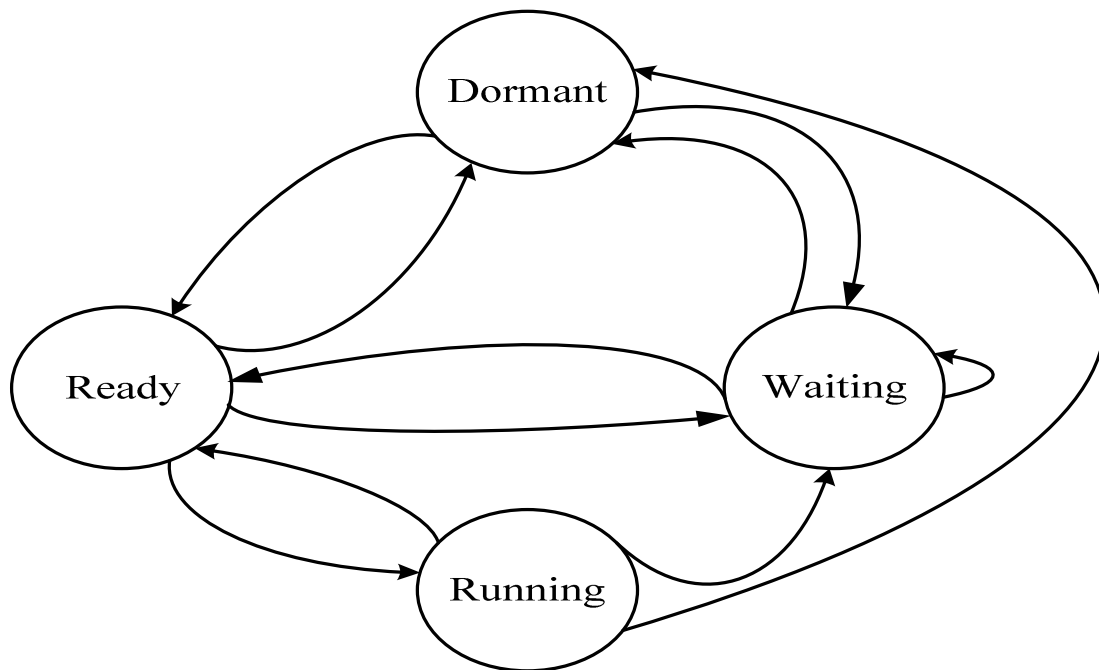


Figure 2.2 – Process State Diagram

2.0 SYSTEM OVERVIEW

2.3.2.2.1.2 Process State Transitions Versus Partition Mode Transitions

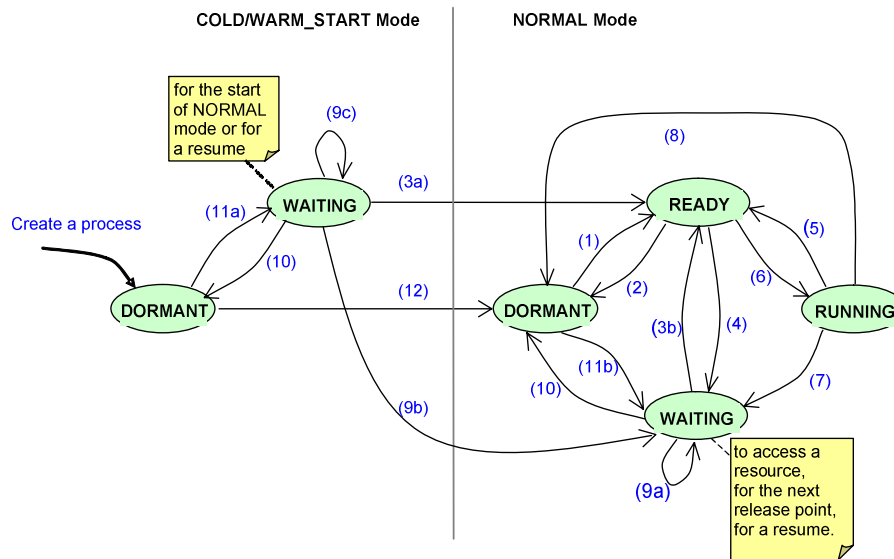


Figure 2.3 - Process States and State Transitions in Accordance with the Modes of the Partition

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 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 in the ready state when the awaited release point is reached.

2.0 SYSTEM OVERVIEW

(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 or 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.

2.0 SYSTEM OVERVIEW

(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.

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 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 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. 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 oldest one. That 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 or its implementation must distinguish between two types of processes. It merely states that the APEX interface should provide the application software with a means to request scheduling of processes on a periodic or event-driven basis.

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

2.3.3 Time Management

Time management is an important characteristic of the O/S used in real-time systems. Time is unique and independent of partition execution within a core 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.

2.0 SYSTEM OVERVIEW

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.

Time replenishment determines the next deadline value as represented in the figure below:

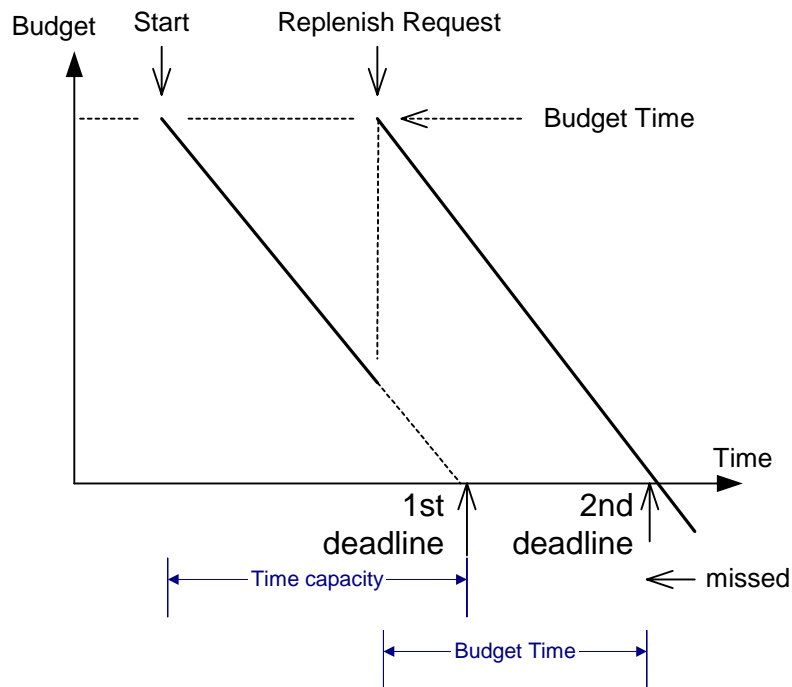


Figure 2.3.3-1 – Time Replenishment

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.

A time-out (or 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 one 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 Allocation

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

2.0 SYSTEM OVERVIEW

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. Its primary definition is communication between two or more partitions executing either on the same core module or on different core modules. It may also mean communication between APEX partitions on a core module and non-ARINC 653 equipment that is external to the 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 continuous block of data of finite length. A message is sent from a single source to one or more destinations. The destination of a message on a core module is a partition, and not a process within a partition.

COMMENTARY

The expression “continuous 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.

The data contents of a message are transparent to the message passing system.

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 configuration, 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 through intermediate 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 partition developer) configures the channel connections within a core module and the channel connections between a core module and component external to the core module.

2.0 SYSTEM OVERVIEW**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 core module, or on different core modules.

The core software 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 software, required to transport the data from source to destination, is invisible to the application(s). The core software, along with the underlying hardware, 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, which are synonymous to external communication services, however formatting and validation, are entirely the responsibility of the application.

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.

2.0 SYSTEM OVERVIEW

4. The core software should ensure that the messages provided by the application are transmitted to the transport mechanism in the same order. The core software should ensure that the messages received from the transport mechanism are delivered to the application in the same order.
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.

2.0 SYSTEM OVERVIEW**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.)

2.0 SYSTEM OVERVIEW

2.3.5.5 Channels and Ports

Channels and ports are expected to be entirely defined at system configuration time. The definition of channels is not confined to the core modules O/S, but is rather distributed on the constituent 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 channel may be configured to operate in a specific mode. Two modes of transfer may be used, sampling mode and queuing mode.

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 partition mode, i.e., NORMAL, COLD_START and WARM_START. It is assumed that the underlying system supports the behavior defined herein.

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 therefore 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

2.0 SYSTEM OVERVIEW

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 is not 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.

Messages are allowed to be segmented/reassembled in the queuing mode. If the transmission of variable length messages is not directly supported, the source port must decompose the message into a series of fixed length segments (except the last one) and the destination port must reassemble those segments. The length of the message segments is lower or equal to the maximum unsegmented length acceptable by the underlying data communication system. The segmentation and reassembling of messages are transparent to application software.

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 assumed to be provided at core module configuration time. 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 (which configuration is a system integrator duty). 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").

2.0 SYSTEM OVERVIEW**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 and transfer the messages in the channel connected to that port.

2.3.5.7.4 Transfer Direction

The transfer 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 Message Segment Length

The message segment length attribute defines the length of data blocks which are to be transferred in the channel. Sampling mode defines the maximum size of the unsegmented message. Queuing mode defines the maximum size of a segment. The size is dependent on the media utilized (i.e., ARINC 429, ARINC 659).

COMMENTARY

Although the maximum segment size is dependent on the architecture, the O/S may segment a message into smaller pieces than can be accommodated by the bus.

There are two distinct types of message decomposition. One is logical and is associated with the definition of that port. The other is physical and is associated with the communication media. The logical type is the responsibility of the application, and the physical type is the responsibility of the system software. Note that there may be several different decompositions according to the various types of communication media relating to that channel.

2.3.5.7.6 Message Storage Requirements

The message storage requirement attribute defines 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.

2.3.5.7.7 Required Refresh Rate

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

2.0 SYSTEM OVERVIEW

2.3.5.7.8 Mapping Requirements

The mapping requirements attribute defines the connection between that port and the physical communication media (i.e., memory, backplane bus, etc.)

COMMENTARY

Messages received by a port may originate either from another port of the core module or the backplane bus. Messages sent by a port are routed to one or more other ports of the core module and/or the backplane bus in the sampling mode, whereas they are routed to either one other port or the backplane bus in the queuing mode.

Ports can be mapped to external physical addresses or to procedures (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 implementor 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 core software 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 should 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. The port creation does not actually create any resources since this has already been done at configuration time.

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 acknowledgment) in the queuing mode.

Either fixed or variable length messages are allowed to be communicated via a port. A length indication must be provided by the application software in addition to the message when sending a message or 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.

2.0 SYSTEM OVERVIEW

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 a 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 may enter the waiting state or the send request may be cancelled. The queued messages are segmented according to the message segment length defined for the port before they are transmitted in FIFO order. If the previous message segment has been correctly transmitted and there are still data to be sent, a new segment is removed from the message queue and transmitted. The channel protocol may allow failed messages (or segments) to be resent`.
4. Queuing mode, receive direction: **Each correct (internally consistent) new received message segment is moved to the receive message queue where the successive segments are reassembled to form a message. 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 at the 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. These undesired effects could constitute a partitioning violation. For example, if a process is held waiting for input from one source, it will be unable to perform alternative processing using different data or the computation of other unrelated processing for which it is responsible. If the waiting upon one source results in the suspension or delay of other processing which should continue in the presence of the failed input, then partitioning could be violated.

2.0 SYSTEM OVERVIEW**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 are provided for 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).

2.3.6.1 Buffers and Blackboards

Inter-process communication of messages is conducted via buffers and blackboards which can support the communication of a single message type 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.

As with process creation, the amount of memory space required to manage buffers and blackboards and to store messages is allocated from the partition's memory which is defined at system build time. Processes can create as many buffers and blackboards as the pre-allocated memory space will support.

Rescheduling of processes will occur when a process attempts to wait. 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. No message should be lost in queuing mode. The number of messages that can be stored in a buffer is determined by the size of the buffer, and is specified at creation time.

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.

2.0 SYSTEM OVERVIEW

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.

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 awaiting 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 resource, and then signals the semaphore when it is finished. A semaphore's value indicates the number of currently available resources.

As with process and message buffer creation, the amount of memory space required to create semaphores is allocated from the partition's memory which is defined at system build time.

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. 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.

2.0 SYSTEM OVERVIEW

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

The Health Monitor (HM) is the function of the O/S responsible for monitoring and reporting hardware, application and O/S software faults and failures. The HM helps to isolate faults and to prevent failures from propagating. Components of the HM are contained within the following software elements:

2.0 SYSTEM OVERVIEW

Core O/S – All HM implementations will use the ARINC 653 O/S Kernel. The O/S uses a HM configuration table 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 as error handlers. Fault messages may be passed by the O/S to a system partition via the HM Callback defined in HM tables (module HM table and partition HM tables). Application partitions may report errors to the system partition via ports. The advantage is that responses do not have to be based purely on a look up table and the implementation is outside of the O/S. However, this functionality could be embedded within the O/S, at the discretion of the system implementer.

2.4.1 Error Levels

Errors may occur at the system, module, partition or process level. These error levels are described in the following sections. The level of an error is defined by the system integrator in the system HM table in accordance with the detected error and the state of the system. For example, if a checksum error is detected for one partition during module initialization, the system integrator will decide if it is a module level error or a partition level error. An error raised at one level may, due to its nature and scope, propagate to a higher level where it is processed.

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 the 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 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.

2.0 SYSTEM OVERVIEW**2.4.1.3 Module Level Errors**

A module level error impacts all the partitions within the module. Examples of module level errors are:

- Module configuration error during module initialization.
- Other errors during module initialization.
- 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
- Core Software - 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 (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.

HM Callback can be defined at the module or partition level. The HM Callback is a procedure that is called by the O/S Kernel whenever a partition or module level fault is reported. The procedure invoked depends on whether the error is a partition or module level error. Also, the HM Callback may be unique for each partition. The parameters passed to the HM Callback are the O/S detected error identification and the system state that caused the partition or module HM table to be invoked. The HM Callback is independent of the O/S HM implementation and the entry point is configured in the module and partition HM tables. Any Partition HM Callback should be written such that it does not violate partitioning.

A mechanism must exist for the HM Callback to pass messages to a partition (typically a system partition) on the same module. This allows the system integrator to manage faults from a partition in order to perform analysis or send fault data to an Onboard Maintenance System.

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) process of the partition, the error handler process. The error handler process is active in NORMAL mode only. The programmer can identify the error and the faulty process via a HM 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: IDLE, COLD_START, WARM_START). Errors, which may occur during the error handler process, are considered to be partition level errors. 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

2.0 SYSTEM OVERVIEW

error code to the faulty application. To maintain application portability, error codes must be implementation independent. These 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 a process level error handler does not exist for the partition then the partition level error response mechanism is invoked by the O/S Kernel.

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 Kernel looks up the reference to the HM Callback in the HM Configuration tables and calls the procedure if one has been identified for the partition.
- On completion of the HM Callback the O/S Kernel looks up the error code response action in the HM configuration tables.
- The O/S Kernel 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 Kernel looks up the reference to the HM Callback procedure in the HM Configuration tables and calls the procedure if one has been identified for the module.
- On completion of the HM Callback the O/S Kernel looks up the error code response action in the HM configuration tables.
- The O/S Kernel performs the response identified in the configuration table.

2.4.3 Recovery Actions

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

2.0 SYSTEM OVERVIEW**2.4.3.1 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.
- Confirm 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.

2.4.3.2 Partition Level Error Recovery Actions

The recovery action is specified for partition errors in the HM configuration tables. The system integrator defines the partition HM table for each partition. For each partition, the fault response for the error type and system state takes into account the partition behavior capability (re-settable or not, degraded mode, etc). Additional recovery actions may be performed in the HM Callback. Example recovery actions are listed below:

- Do nothing, the error should be treated via the HM Callback.
- Stop the partition.
- Stop and restart the partition.

2.4.3.3 Module Level Error Recovery Actions

The recovery action is specified for a module error in the HM configuration tables. The system integrator defines the module HM table for each module. Additional recovery actions may be performed in the HM Callback. Example recovery actions are listed below:

- Do nothing, the error should be treated via the HM Callback.
- Stop the module.
- Stop and restart the module.

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 cabinet 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

2.0 SYSTEM OVERVIEW

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 integrator to ensure that message formats are compatible for 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
- Identity of messages to be sent by the partition
- Identity of messages to be received by the partition

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 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 (Version 1.0) is used to describe the configuration data. XML-Schema (Version 1.0) is used to define the format of the XML data and is defined in this document.

Section 5 of this document discusses the use of XML for defining configuration data. Appendices G, H and I contain the XML-Schema and an example XML instance for the current version of the ARINC 653.

The system integrator is responsible for building the XML instance file for module configuration.

2.5.2.1 Configuration Tables for System Initialization

During core module initialization each partition within the core module has to be brought into a state where it is ready to execute. Checks must be performed to ensure that all partitions are present and all the channels of communications between partitions exist and are operable.

The operating system ensures that the partition configuration is correct by reference to a configuration table. This configuration table will contain partition identifiers which are resident on the module, the memory requirements of each partition and the number and size of ports required by the partition. The operating system checks that all partitions are installed on the module, that each partition has been allocated sufficient memory and that sufficient port space has been allocated to

2.0 SYSTEM OVERVIEW

each partition. The health monitor will be notified of any discrepancies (e.g., absent partitions, memory clashes) and an appropriate recovery action initiated. During initialization the operating system will also carry out port creation. Port creation creates a link between a port name and the physical port area. Configuration tables will be used to generate the link between the port name and the physical area of memory assigned to that port. The configuration table will contain a list of port names, the start address of each port, its corresponding size and in the case of queuing ports the size of the queue. When a port create takes place, the operating system refers to the configuration table to ensure that the port is valid for the partition (**i.e., compatible with the configuration**) and returns an identifier corresponding to the port location.

2.5.2.2 Configuration Tables for Inter-Partition Communication

Configuration tables are also required in inter-partition communication. These tables must be built by the system integrator, they contain information that can only be known when the configuration of partitions on modules is determined (e.g., whether the sending and receiving partitions are located on the same module, different module within the same cabinet or different cabinet). When the operating system detects that a message has been written into a port then it refers to a configuration table which contains the mapping from the identified port to the next port in the channel, this port may be a port in another partition on the module, or an address in the inter-module memory. The precise contents of message configuration table will depend on the actual media employed in external message passing.

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 System HM table, the Module HM table and the Partition HM table.

The System HM table defines the level of an error (Module, Partition, Process) according to the detected error and the state of the system. For process level errors only, the error codes are indicated in this table. These error codes are implementation independent.

The Module HM table defines the recovery action (e.g., shut down the module, reset the module) associated with the detected error for each module level error.

The Partition HM table defines the recovery action (e.g., stop the partition, restart the partition in warm or cold mode) associated with the detected error for each partition level error. Each partition has a separate table.

Both the Partition HM table and the Module HM table have optional HM Callback entries for system integrator supplied HM functionality (e.g., Reporting faults to a higher level HM application or central maintenance system).

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.

3.0 SERVICE REQUIREMENTS

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 provided in Appendix C of this document. 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 of the procedure body is not. 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.

Partition level objects 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 items. For a dynamic system, the object attributes need to be checked for reasonableness (e.g., range checking) or 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 a basic multitasking capability. 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.

3.0 SERVICE REQUIREMENTS

3.1.1 Return Code Data Type

This section contains the return code data type declaration. This is common to all APEX services. They are listed only once 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.

3.0 SERVICE REQUIREMENTS

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	
<u>Return Code Value</u>	<u>Commentary</u>
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 expected to be used for setting the partition back to idle (partition shutdown), and to cold start or warm start (partition restart), when a serious fault is detected and processed.

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

3.0 SERVICE REQUIREMENTS

```

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 first release points of all previously started periodic processes
    to their next partition period;
    set first release points of all previously delay started periodic processes
    to their next partition period, including their delay times;
    calculate the DEADLINE_TIME of all non-dormant processes in the partition;

    activate the process scheduling;
  end if;
  RETURN_CODE := NO_ERROR;

end SET_PARTITION_MODE;

```

The return codes for this service are explained below.

<u>SET_PARTITION_MODE</u>	
<u>Return Code Value</u>	<u>Commentary</u>
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 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. These specifications define and manage processes. The scope of a process management service is restricted to its 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 implementation dependent;

```


3.0 SERVICE REQUIREMENTS

```
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 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;
```

3.0 SERVICE REQUIREMENTS

```

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.

<u>Return Code Value</u>	<u>Commentary</u>
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 operating 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.

<u>Return Code Value</u>	<u>Commentary</u>
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. Partitions can create as many processes as the pre-allocated memory space will support. Consistency among process parameters and partition parameters are checked.

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 without DEADLINE.

```

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) =>

```

3.0 SERVICE REQUIREMENTS

```

    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_CONFIG;
when (ATTRIBUTES.TIME_CAPACITY out of range) =>
    RETURN_CODE := INVALID_PARAM;
when (operating mode is NORMAL) =>
    RETURN_CODE := INVALID_MODE;

```

normal

```

    set the process attributes to ATTRIBUTES;
    set the process state to DORMANT;
    reset context and stack;
    PROCESS_ID := identifier of the created process;
    RETURN_CODE := NO_ERROR;

```

end CREATE_PROCESS;

The return codes for this service are explained below.

<u>CREATE_PROCESS</u>	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_CONFIG	Implementation limits exceeded
NO_ACTION	The process named ATTRIBUTES.NAME is already created
INVALID_PARAM	The ATTRIBUTES.STACK_SIZE out of range
INVALID_PARAM	The ATTRIBUTES.BASE_PRIORITY out of range
INVALID_CONFIG	The ATTRIBUTES.PERIOD is not consistent with the other parameters
INVALID_PARAM	The ATTRIBUTES.TIME_CAPACITY out of range
INVALID_MODE	The operating mode is NORMAL

3.3.2.4 SET_PRIORITY

The SET_PRIORITY service request changes a process's current priority. The process is placed as the newest process with that priority in the ready state. 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 priority queue for that resource may or may not be reordered. Therefore, from the viewpoint of portability, applications should not rely on the reordering of the queue.

Two types of priority are represented in the system, current and base. Current priority is the priority used by the O/S for dispatching and resource allocation.

```

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

```

3.0 SERVICE REQUIREMENTS

```

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 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
  end if;
  RETURN_CODE := NO_ERROR;

end SET_PRIORITY;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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, until the RESUME service request is issued or the specified time-out value expires.

COMMENTARY

If a process suspends 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) =>
    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 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 time-out) then
      RETURN_CODE := TIMED_OUT;
    end if;
  end if;
end SUSPEND_SELF;

```

3.0 SERVICE REQUIREMENTS

```

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.

<u>SUSPEND_SELF</u>	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_MODE	Preemption is disabled or process is error handler process
INVALID_MODE	Process is periodic
INVALID_PARAM	TIME_OUT is out of range
TIMED_OUT	TIME_OUT has elapsed

3.3.2.6 SUSPEND

The SUSPEND service request allows the current process to suspend the execution of any aperiodic process except itself, until the suspended process is 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 (either by occurrence of a condition or expiration of a time-out or a reset of the queue) before the end of its suspension.

A process may suspend any other process asynchronously. Though this practice is not recommended, there may be partitions that require it.

COMMENTARY

If a process suspends an already suspended process, it has no effect.

```

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_preemption was set.
  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 itself) =>
    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;

```

3.0 SERVICE REQUIREMENTS

```

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 itself
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	The specified process is periodic

3.3.2.7 RESUME

The RESUME service request allows the current process to resume another previously suspended process. The resumed process will become ready if it is not waiting on a resource (delay, semaphore, period, event, message). A periodic process cannot be suspended, so it can not 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 itself) =>
      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 is suspended with a time-out) then
      stop the affected time counter;
    end if;
    if (the specified process is not waiting on a resource) then
      set the specified process state to READY;
      if (preemption is enabled) then
        ask for process scheduling;
        -- The current process may be preempted
      end if;
    end if;
    RETURN_CODE := NO_ERROR;

end RESUME;

```

3.0 SERVICE REQUIREMENTS

The return codes for this service are explained below.

RESUME	
<u>Return Code Value</u>	<u>Commentary</u>
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 itself
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 is placed in the unlocked condition. 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.

No return code is returned to the requesting process procedure.

procedure STOP_SELF is

```

normal
    release all the resources used by the current process;
    if (not error handler process) then
        reset the LOCK_LEVEL counter;
    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 procedure allows the current process to abort the execution of any process except itself. When a process aborts another process that is currently pending in a queue, the aborted process is removed from the 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 itself) =>
        RETURN_CODE := INVALID_PARAM;
    when (the state of the specified process is DORMANT) =>
        RETURN_CODE := NO_ACTION;
```

3.0 SERVICE REQUIREMENTS

```

normal
  set the specified process state to DORMANT;
  release all the resources used by the specified process;
  if (current process is error handler and process id is process which the error
  handler preempted) then
    reset the LOCK_LEVEL counter;
  end if;
  if (specified process is pending in a queue) then
    remove the process from the pending queue;
  end if;
  RETURN_CODE := NO_ERROR;

end STOP;

```

The return codes for this service are explained below.

STOP	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process or identifies itself
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, 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 procedure 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;

  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

```


3.0 SERVICE REQUIREMENTS

```

-- (the mode of the partition is COLD_START or WARM_START)
set the specified process state to WAITING;
-- Wait for NORMAL mode
end if;
RETURN_CODE := NO_ERROR;
-- At the end of the initialization phase (when the partition mode
-- becomes NORMAL) then
-- set the state of the specified process to READY;
-- DEADLINE_TIME value is calculated;.
-- Because the partition enters NORMAL mode, the process scheduling is
-- called.
-- 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;
    -- Wait for partition switches to 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
endif;

end START;
```

The return codes for this service are explained below.

START

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process
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' deadline expiration time is also calculated.

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

Note: DELAYED_START with a delay of zero is equivalent to START.

3.0 SERVICE REQUIREMENTS

```

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 (the process is periodic and DELAY_TIME is greater or equal to the period
        of the specified process) =>
    RETURN_CODE := INVALID_PARAM;

normal
  if (the 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 the 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
      end if;
    else
      -- (the mode of the partition is COLD_START or WARM_START)
      set the specified process state to WAITING;
      -- Wait for NORMAL mode
    end if;
    RETURN_CODE := NO_ERROR;
    -- At the end of the initialization phase (when the partition mode
    -- becomes 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
    --   The process is already in the WAITING state;
    --   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;
    -- Because the partition enters NORMAL mode, the process scheduling is
    -- called.
    -- End of start_aperiodic_process with delay_time
  else

```

3.0 SERVICE REQUIREMENTS

```

-- 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 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;
    -- Wait for partition switches to 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 with delay_time
endif;

End DELAYED_START

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process
INVALID_PARAM	DELAY_TIME is greater or equal to the period of the specified process
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 the partition and disables process rescheduling for a partition.

A partition should be able to prevent the normal process rescheduling operations of the operating system while its processes are accessing critical sections or resources shared by multiple processes of the same partition. These critical sections may be specific areas of memory, certain physical devices, or simply the normal calculations and activity of a particular process.

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 operating system software. Since this service is provided by the operating system and all resultant actions and effects are known beforehand, the integrity of the operating system remains intact and unaffected by the request. It should also be noted that this service has no impact on the scheduling of other partitions, it only affects rescheduling within the individual partition.

3.0 SERVICE REQUIREMENTS

Preemptibility control allows critical sections of code to be guaranteed to execute without preemption by other processes within the partition. If a process within a critical section is interrupted by the end of a partition window, it is guaranteed to be the first to execute when the partition is resumed.

```

procedure LOCK_PREEMPTION
  (LOCK_LEVEL   : out LOCK_LEVEL_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 (LOCK_LEVEL >= MAX_LOCK_LEVEL) =>
      RETURN_CODE := INVALID_CONFIG;

  normal
    LOCK_LEVEL := LOCK_LEVEL + 1;
    RETURN_CODE := NO_ERROR;

end LOCK_PREEMPTION;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_CONFIG	The LOCK_LEVEL value is higher or equal to MAX_LOCK_LEVEL
NO_ACTION	When the calling 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 the partition. The process rescheduling function is performed only when the lock level becomes zero.

```

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

  error
    when (process is error handler process or OPERATING_MODE is not NORMAL
          or LOCK_LEVEL indicates unlocked) =>
      RETURN_CODE := NO_ACTION;

  normal
    LOCK_LEVEL := LOCK_LEVEL - 1;
    if (LOCK_LEVEL = 0) then
      ask for process scheduling;
      -- The current process may be preempted
    end if;
    RETURN_CODE := NO_ERROR;

end UNLOCK_PREEMPTION;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
NO_ACTION	LOCK_LEVEL indicates unlocked or calling process is error handler process or OPERATING_MODE is not NORMAL

3.0 SERVICE REQUIREMENTS

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 ID, e.g., 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_PROCESS_ID Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_MODE	Current process has no ID

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 for a partition to control periodic and aperiodic processes.

For periodic processes, each process within a partition is able to specify an amount of time (i.e., elapsed time) that specifies the maximum length of time the process is allowed to consume in satisfying its processing requirements. 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.

3.0 SERVICE REQUIREMENTS

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.

```
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) =>
      RETURN_CODE := INVALID_PARAM;
    when (DELAY_TIME is infinite) =>
      RETURN_CODE := INVALID_PARAM;

  normal
    if (DELAY_TIME = 0) then
      Set the current process state to READY;
      Ask for process scheduling
    else
      Set the current process state to WAITING;
      Initiate a time counter with duration DELAY_TIME;
      Ask for process scheduling;
      -- The current process is blocked and will return to the READY state by
      -- expiration of the delay time (except if another process suspended it)
    end if;
    RETURN_CODE := NO_ERROR;
  end TIMED_WAIT;
```

The return codes for this service are explained below.

<u>TIMED_WAIT</u>	<u>Return Code Value</u>	<u>Commentary</u>
	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

3.0 SERVICE REQUIREMENTS

3.4.2.2 PERIODIC_WAIT

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

Processes need to suspend their execution for a minimum amount of elapsed time. Processes also need to suspend their execution until periodic release points in the processor time line.

```

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 (requesting process is not periodic) =>
      RETURN_CODE := INVALID_MODE;

  normal
    Set the requesting 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 requesting process is blocked and on the next release point
    -- the process will return to the ready state
    RETURN_CODE := NO_ERROR;

end PERIODIC_WAIT;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_MODE	Preemption is disabled or process is error handler process
INVALID_MODE	The calling process is not periodic

3.4.2.3 GET_TIME

The service GET_TIME requests the value of the system clock. The system clock is the value of a clock common to all processors in the module.

```

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

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

end GET_TIME;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion

3.0 SERVICE REQUIREMENTS

3.4.2.4 REPLENISH

The REPLENISH service request updates the deadline of the requesting process with a specified BUDGET_TIME value. It is not allowed to postpone a periodic process' deadline 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 (requesting process is periodic and new deadline will exceed next release point) =>
      RETURN_CODE := INVALID_MODE;
    when (BUDGET_TIME is out of range) =>
      RETURN_CODE := INVALID_PARAM;

  normal
    Set a new DEADLINE_TIME value for the current process
      (current system clock + BUDGET_TIME):
    -- if 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	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	BUDGET_TIME is out of range
INVALID_MODE	The new deadline time would exceed the next release point
NO_ACTION	calling 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 at system configuration time. It is expected that the memory space is checked either at build time or before the first application process becomes running.

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 continuous 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);

```


3.0 SERVICE REQUIREMENTS

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
```

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. Each new occurrence of a message overwrites the previous one. Messages may have a variable length. A refresh period attribute applies to reception 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

3.0 SERVICE REQUIREMENTS

REFRESH_PERIOD attribute indicates the maximum acceptable age of a valid message, from the time it was received in the port. A port must be created during the 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)
      i.e., 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) =>
      RETURN_CODE := INVALID_CONFIG;
    when (the port named SAMPLING_PORT_NAME is already created) =>
      RETURN_CODE := NO_ACTION;
    when (MAX_MESSAGE_SIZE is out of range or MAX_MESSAGE_SIZE is not equal to
      the value specified in the configuration data) =>
      RETURN_CODE := INVALID_CONFIG;
    when (PORT_DIRECTION is invalid or PORT_DIRECTION is not equal to the
      value specified in the configuration data) =>
      RETURN_CODE := INVALID_CONFIG;
    when (REFRESH_PERIOD is out of range or REFRESH_PERIOD is not equal to the
      value specified in the configuration data) =>
      RETURN_CODE := INVALID_CONFIG;
    when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

  normal
    SAMPLING_PORT_ID := 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.

<u>CREATE_SAMPLING_PORT</u>	<u>Return Code Value</u>	<u>Commentary</u>
	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

3.0 SERVICE REQUIREMENTS

INVALID_CONFIG	SAMPLING_PORT_NAME does not identify a sampling port known by the configuration
INVALID_CONFIG	MAX_MESSAGE_SIZE is out of range or is not equal to the value specified in the configuration data
INVALID_CONFIG	PORT_DIRECTION is invalid or is not equal to the value specified in the configuration data.
INVALID_CONFIG	REFRESH_PERIOD is out of range or is not equal to the value specified in the configuration data
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 (the LENGTH is greater than MAX_MESSAGE_SIZE for the port) =>
      RETURN_CODE := INVALID_CONFIG;
    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	
<u>Return Code Value</u>	<u>Commentary</u>
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 port
INVALID_MODE	SAMPLING_PORT_ID 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 in 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.

```

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

```

3.0 SERVICE REQUIREMENTS

```

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
    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

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
NO_ACTION	Sampling port is empty
INVALID_PARAM	SAMPLING_PORT_ID does not identify an existing sampling port
INVALID_MODE	SAMPLING_PORT_ID is not configured to operate as a Destination

3.6.2.1.4 GET_SAMPLING_PORT_ID

The GET_SAMPLING_PORT_ID service allows a process to obtain a sampling port identifier by specifying the 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 sampling port named SAMPLING_PORT_NAME) =>
    RETURN_CODE := INVALID_CONFIG;

normal
  SAMPLING_PORT_ID := (ID of the sampling port named SAMPLING_PORT_NAME)
  RETURN_CODE      := NO_ERROR;

end GET_SAMPLING_PORT_ID;

```

3.0 SERVICE REQUIREMENTS

The return codes for this service are explained below.

<u>GET_SAMPLING_PORT_ID</u>	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current 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.

<u>GET_SAMPLING_PORT_STATUS</u>	
<u>Return Code Value</u>	<u>Commentary</u>
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 have 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 is set in accordance.

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 port must be created during the initialization mode before it can be used.

3.6.2.2.1 CREATE_QUEUEING_PORT

The CREATE_QUEUEING_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 QUEUEING_DISCIPLINE attribute indicates whether blocked processes are queued in FIFO, or in priority order.

3.0 SERVICE REQUIREMENTS

```

procedure CREATE_QUEUEING_PORT
  (QUEUEING_PORT_NAME : in QUEUEING_PORT_NAME_TYPE;
   MAX_MESSAGE_SIZE   : in MESSAGE_SIZE_TYPE;
   MAX_NB_MESSAGE     : in MESSAGE_RANGE_TYPE;
   PORT_DIRECTION     : in PORT_DIRECTION_TYPE;
   QUEUEING_DISCIPLINE : in QUEUEING_DISCIPLINE_TYPE;
   QUEUEING_PORT_ID   : out QUEUEING_PORT_ID_TYPE;
   RETURN_CODE        : out RETURN_CODE_TYPE) is
error
  when (implementation-defined limit to queueing port creation is exceeded)
    i.e., Insufficient storage capacity for the creation of the specified
    port or maximum number of queueing ports have been created =>
    RETURN_CODE := INVALID_CONFIG;
  when (no queueing port of the partition is named QUEUEING_PORT_NAME in the
    configuration) =>
    RETURN_CODE := INVALID_CONFIG;
  when (the port named QUEUEING_PORT_NAME is already created) =>
    RETURN_CODE := NO_ACTION;
  when (MAX_MESSAGE_SIZE is out of range or MAX_MESSAGE_SIZE is not equal to
    the value specified in the configuration data) =>
    RETURN_CODE := INVALID_CONFIG;
  when (MAX_NB_MESSAGE is out of range or MAX_NB_MESSAGE is not equal to the
    value specified in the configuration data) =>
    RETURN_CODE := INVALID_CONFIG;
  when (PORT_DIRECTION is invalid or PORT_DIRECTION is not equal to the
    value specified in the configuration data) =>
    RETURN_CODE := INVALID_CONFIG;
  when (the QUEUEING_DISCIPLINE is invalid) =>
    RETURN_CODE := INVALID_CONFIG;
  when (operating mode is NORMAL) =>
    RETURN_CODE := INVALID_MODE;

normal
  QUEUEING_PORT_ID := identifier assigned by the O/S to the port named
    QUEUEING_PORT_NAME;
  RETURN_CODE      := NO_ERROR;

end CREATE_QUEUEING_PORT;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
NO_ACTION	Port named QUEUEING_PORT_NAME is already created
INVALID_CONFIG	Implementation-defined limit to queueing port creation is exceeded
INVALID_CONFIG	QUEUEING_PORT_NAME does not identify a queueing port known by the configuration
INVALID_CONFIG	MAX_MESSAGE_SIZE is out of range or is not equal to the value specified in the configuration data
INVALID_CONFIG	MAX_NB_MESSAGE is out of range or is not equal to the value specified in the configuration data
INVALID_CONFIG	PORT_DIRECTION is invalid or is not equal to the value specified in the configuration data
INVALID_CONFIG	QUEUEING_DISCIPLINE is invalid
INVALID_MODE	Operating mode is NORMAL

3.0 SERVICE REQUIREMENTS

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 (the TIME_OUT is out of range) =>
      RETURN_CODE := INVALID_PARAM;
    when (the LENGTH is greater than MAX_MESSAGE_SIZE for the port) =>
      RETURN_CODE := INVALID_CONFIG;
    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 process is error handler process) then
      RETURN_CODE := INVALID_MODE;
    else
      if (TIME_OUT is not infinite) then
        initiate a time counter with duration TIME_OUT;
      end if;
      set the current process state to WAITING;
      insert the process in the 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;
      end if;
    end if;
  end normal;
end procedure;

```

3.0 SERVICE REQUIREMENTS

```

    RETURN_CODE := NO_ERROR;
end if;
-- the process is removed from the sending process queue and is moved
-- from the WAITING to the READY state (except if another process
-- suspended it) the process scheduling occurs if preemption is enabled
end if;

```

```
end SEND_QUEUEING_MESSAGE;
```

The return codes for this service are explained below.

SEND_QUEUEING_MESSAGE

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	QUEUEING_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_MODE	QUEUEING_PORT_ID is not configured to operate as a source
NOT_AVAILABLE	Insufficient space in the QUEUEING_PORT_ID to accept a new message
INVALID_MODE	(Preemption is disabled or process is error handler process) and time-out is not zero
TIMED_OUT	Specified time-out is expired

3.6.2.2.3 RECEIVE_QUEUEING_MESSAGE

The RECEIVE_QUEUEING_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 indicates the queue overflow.

```

procedure RECEIVE_QUEUEING_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 (QUEUEING_PORT_ID does not identify an existing queuing port) =>
    RETURN_CODE := INVALID_PARAM;
  when (the TIME_OUT is out of range) =>
    RETURN_CODE := INVALID_PARAM;
  when (the specified port is not configured to operate as a destination
  port) =>
    RETURN_CODE := INVALID_MODE;

```


3.0 SERVICE REQUIREMENTS

```

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
    RETURN_CODE := NOT_AVAILABLE;
  elsif (preemption is disabled or process is error handler process) then
    RETURN_CODE := INVALID_MODE;
  else
    if (TIME_OUT is not infinite) then
      initiate a time counter with duration TIME_OUT;
    end if;
    set the current process state to WAITING;
    insert the process in the 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
      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 valid message of 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 receiving process queue and is moved
    -- from the WAITING to the READY state (except if another process
    -- suspended it) the process scheduling occurs if preemption is enabled
  end if;
  -- if no process is waiting in the receiving process queue of the port then
  -- the new valid message is put in the port's message queue

end RECEIVE_QUEUING_MESSAGE;

```

3.0 SERVICE REQUIREMENTS

The return codes for this service are explained below.

RECEIVE_QUEUEING_MESSAGE	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
TIMED_OUT	Specified time-out is expired
INVALID_PARAM	QUEUEING_PORT_ID does not identify an existing queueing port
INVALID_PARAM	TIME_OUT is out of range
INVALID_MODE	QUEUEING_PORT_ID is not configured to operate as a destination
INVALID_CONFIG	The queue has overflowed since it was last read
NOT_AVAILABLE	There is no message in the QUEUEING_PORT_ID
INVALID_MODE	(Preemption is disabled or process is error handler process) and time-out is not zero

3.6.2.2.4 GET_QUEUEING_PORT_ID

The GET_QUEUEING_PORT_ID service allows a process to obtain a queueing port identifier by specifying the queueing port name.

```

procedure GET_QUEUEING_PORT_ID
  (QUEUEING_PORT_NAME   : in  QUEUEING_PORT_NAME_TYPE;
   QUEUEING_PORT_ID     : out QUEUEING_PORT_ID_TYPE;
   RETURN_CODE          : out RETURN_CODE_TYPE) is

  error
    when (there is no current queueing port named QUEUEING_PORT_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

  normal
    QUEUEING_PORT_ID := (ID of the queueing port named QUEUEING_PORT_NAME);
    RETURN_CODE      := NO_ERROR;

end GET_QUEUEING_PORT_ID;
```

The return codes for this service are explained below.

GET_QUEUEING_PORT_ID	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current queueing port named QUEUEING_PORT_NAME

3.6.2.2.5 GET_QUEUEING_PORT_STATUS

The GET_QUEUEING_PORT_STATUS service returns the current status of the specified queueing port.

```

procedure GET_QUEUEING_PORT_STATUS
  (QUEUEING_PORT_ID      : in  QUEUEING_PORT_ID_TYPE;
   QUEUEING_PORT_STATUS  : out QUEUEING_PORT_STATUS_TYPE;
   RETURN_CODE           : out RETURN_CODE_TYPE) is

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

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

end GET_QUEUEING_PORT_STATUS;
```

3.0 SERVICE REQUIREMENTS

The return codes for this service are explained below.

GET_QUEUING_PORT_STATUS	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	QUEUING_PORT_ID does not identify an existing queuing port

3.7 Intrapartition Communication

Two communication mechanisms are available for intrapartition communication. The first one allows process communication via buffers and blackboards. The second one is only for process synchronization, it is realized by mean of 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;

```

3.0 SERVICE REQUIREMENTS

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. In buffers, the messages are queued in FIFO order. The buffer message size is variable, but a maximum size value is given at buffer creation. A buffer must be created during the initialization mode 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.

3.0 SERVICE REQUIREMENTS

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 out of range) =>
      RETURN_CODE := INVALID_PARAM;
    when (MAX_NB_MESSAGE is out of range) =>
      RETURN_CODE := INVALID_PARAM;
    when (QUEUING_DISCIPLINE is not valid) =>
      RETURN_CODE := INVALID_PARAM;
    when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

  normal
    BUFFER_ID := (ID of unallocated buffer control block);
    Set the process buffer 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.

<u>CREATE_BUFFER</u>	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_CONFIG	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 out of range
INVALID_PARAM	MAX_NB_MESSAGE is out of range
INVALID_PARAM	QUEUING_DISCIPLINE is not valid
INVALID_MODE	Operating mode is NORMAL

3.0 SERVICE REQUIREMENTS

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 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 too long for the specified buffer) =>
      RETURN_CODE := INVALID_PARAM;
    when (TIME_OUT is out of range) =>
      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 process from the process queue;
        Copy the message represented by MESSAGE_ADDR and LENGTH into the
          receiving process's message area of the RECEIVE service request done
          by this receiving process;
        if (this process is waiting on an empty buffer with a time-out) then
          stop the affected time counter;
        end if;
        move receiving process from the WAITING to the READY state (except if
          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 process is 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 process queue at the position specified
        by the queuing discipline;
      ask for process scheduling;
      -- The current process is blocked, and will be removed from that queue and
      -- return in READY state by a RECEIVE service request on that buffer from
      -- another process (except if 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

```

3.0 SERVICE REQUIREMENTS

```

else
  set the process state to WAITING;
  insert this process in the buffer process queue at the position specified
    by the queuing discipline;
  initiate a time counter with duration TIME_OUT;
  ask for process scheduling;
  -- Current process is blocked and will be removed from queue and return in
  -- READY state by expiration of time-out or by RECEIVE service request on
  -- the buffer from other process (unless other 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		Message is too long
INVALID_PARAM		TIME_OUT is out of range
INVALID_MODE		(Preemption is disabled or process is 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) =>
      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 processes waiting on this full buffer) then
        remove the first waiting process from the process queue;
        put the message sent by this waiting process in the FIFO message queue;
      end if;
    end if;
  end procedure;

```

3.0 SERVICE REQUIREMENTS

```

    if (this process is waiting on this full buffer with a time-out) then
        stop the affected time counter;
    end if;
    move this sending process from the WAITING to the READY state (except
    if 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 process is 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 process queue at the position specified
    by the queuing discipline;
    ask for process scheduling;
    -- The current process is blocked and will be removed from that queue and
    -- return in READY state by a SEND service request on that buffer from
    -- another process (except if another process suspended it)
    RETURN_CODE := NO_ERROR;
    -- the next SEND service request will copy the sent message into the
    -- the location represented by MESSAGE_ADDR and LENGTH
else
    set the current process state to WAITING;
    insert this process in the buffer process queue at the position specified
    by the queuing discipline;
    initiate a time counter with duration TIME_OUT;
    ask for process scheduling;
    -- Current process is blocked and will be removed from queue and return in
    -- READY state by expiration of time-out or by SEND service request on
    -- buffer from other process (unless other process suspended it)
    if (expiration of the time-out) then
        RETURN_CODE := TIMED_OUT;
    else
        RETURN_CODE := NO_ERROR;
        -- The next SEND service request will stop the time counter and will
        -- copy the sent message into the location represented by
        -- MESSAGE_ADDR and LENGTH
    end if;
end if;
end RECEIVE_BUFFER;

```

The return codes for this service are explained below.

<u>RECEIVE_BUFFER</u>	
<u>Return Code Value</u>	<u>Commentary</u>
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 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 is expired

3.0 SERVICE REQUIREMENTS

3.7.2.1.4 GET_BUFFER_ID

The GET_BUFFER_ID service request allows the current process to get the identifier of a buffer giving its 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 (there is no current partition buffer named BUFFER_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

  normal
    BUFFER_ID := (ID of the buffer named BUFFER_NAME);
    RETURN_CODE := NO_ERROR;
end GET_BUFFER_ID;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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  => maximum number of messages inside the buffer,
       MAX_MESSAGE_SIZE => maximum size of messages,
       WAITING_PROCESSES => the number of waiting processes);
    RETURN_CODE := NO_ERROR;

end GET_BUFFER_STATUS;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	BUFFER_ID does not identify an existing buffer

3.0 SERVICE REQUIREMENTS

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 the current one. The blackboard message size is variable, but a maximum size value is given at blackboard creation. A blackboard must be created during the initialization mode 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.

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 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) =>
      RETURN_CODE := INVALID_CONFIG;
    when (the blackboard named BLACKBOARD_NAME has already been created) =>
      RETURN_CODE := NO_ACTION;
    when (MAX_MESSAGE_SIZE is out of range) =>
      RETURN_CODE := INVALID_PARAM;
    when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

  normal
    BLACKBOARD_ID := (ID of unallocated blackboard control block);
    Set the empty indicator to TRUE;
    RETURN_CODE := NO_ERROR;

end CREATE_BLACKBOARD;

```

The return codes for this service are explained below.

<u>CREATE_BLACKBOARD</u>	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_CONFIG	Not enough available storage space for the creation of the specified blackboard
NO_ACTION	The blackboard named BLACKBOARD_NAME has already been created
INVALID_PARAM	MAX_MESSAGE_SIZE is out of range
INVALID_MODE	Operating mode is NORMAL

3.0 SERVICE REQUIREMENTS

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.

```

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 (message is too long) =>
      RETURN_CODE := INVALID_PARAM;

  normal
    set the 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;
      move them from the WAITING to the READY state (except if 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;

end DISPLAY_BLACKBOARD;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	BLACKBOARD_ID not identify an existing blackboard
INVALID_PARAM	Message is too long

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 a time-out maximum duration.

COMMENTARY

When processes are waiting on an empty blackboard and another process writes a message on that blackboard, then all the waiting processes on that blackboard become ready and will read the last available message on that blackboard in any case. After that, even if a process clears the blackboard, the now ready processes (previously waiting processes on that blackboard) will read the last available message. If a process writes another message in that blackboard, the now ready processes will read this new available message, even if the blackboard is cleared again.

3.0 SERVICE REQUIREMENTS

```

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) =>
      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
      RETURN_CODE := NOT_AVAILABLE;
    elsif (preemption is disabled or process is error handler process) then
      RETURN_CODE := INVALID_MODE;
    elsif (TIME_OUT is infinite) then
      set the process state to WAITING;
      ask for process scheduling;
      -- The current process is blocked and will return to READY state by a
      -- DISPLAY service request on that blackboard from another process (except
      -- if another process suspended it)
      copy the last available message of the blackboard to the
        location represented by MESSAGE_ADDR;
      LENGTH := Length of the copied message;
      RETURN_CODE := NO_ERROR;
    else
      set the process state to WAITING;
      initiate a time counter with duration TIME_OUT;
      ask for process scheduling;
      -- The current process is blocked and will return to READY state by
      -- expiration of time-out or by a DISPLAY service request on that
      -- blackboard from another process (except if another process suspended
      -- it)
      if (expiration of the time-out) then
        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 service request will stop the time counter
      end if;
    end if;
  end if;

end READ_BLACKBOARD;

```

3.0 SERVICE REQUIREMENTS

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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 process is error handler process) and time-out is not zero
NOT_AVAILABLE	No message in the blackboard
TIMED_OUT	The specified time-out is 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.

```

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.

<u>Return Code Value</u>	<u>Commentary</u>
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 allows the current process to get the identifier of a blackboard giving its 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 (there is no current partition blackboard named BLACKBOARD_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

  normal
    BLACKBOARD_ID := (ID of the blackboard named BLACKBOARD_NAME);
    RETURN_CODE   := NO_ERROR;

end GET_BLACKBOARD_ID;
```

3.0 SERVICE REQUIREMENTS

The return codes for this service are explained below.

<u>GET_BLACKBOARD_ID</u>	
<u>Return Code Value</u>	<u>Commentary</u>
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 empty indicator,
       MAX_MESSAGE_SIZE => maximum size of messages,
       WAITING_PROCESSES => the number of waiting processes);
    RETURN_CODE := NO_ERROR;

end GET_BLACKBOARD_STATUS;

```

The return codes for this service are explained below.

<u>GET_BLACKBOARD_STATUS</u>	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	BLACKBOARD_ID does not identify an existing blackboard

3.7.2.3 Semaphore Services

A counting semaphore is a synchronization object commonly used to provide access to partition resources. A semaphore must be created during the initialization mode 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.

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. 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 that semaphore.

3.0 SERVICE REQUIREMENTS

```

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 (the 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 > MAXIMUM_VALUE) =>
    RETURN_CODE := INVALID_PARAM;
  when (QUEUING_DISCIPLINE is not valid) =>
    RETURN_CODE := INVALID_PARAM;
  when (operating mode is NORMAL) =>
    RETURN_CODE := INVALID_MODE;

normal
  SEMAPHORE_ID := (ID of unallocated semaphore control block);
  Set the process semaphore queuing discipline to QUEUING_DISCIPLINE;
  -- This information will be then used by the O/S to order processes in queues
  -- (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.

<u>CREATE_SEMAPHORE</u>	<u>Commentary</u>
<u>Return Code Value</u>	
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 > MAXIMUM_VALUE
INVALID_PARAM	QUEUING_DISCIPLINE is not valid
INVALID_MODE	Operating mode is NORMAL

3.7.2.3.2 WAIT_SEMAPHORE

The WAIT_SEMAPHORE service request 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. The process goes on executing if the current value of the specified semaphore is positive and the semaphore 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

```

3.0 SERVICE REQUIREMENTS

```

error
  when (SEMAPHORE_ID does not identify an existing semaphore) =>
    RETURN_CODE := INVALID_PARAM;
  when (TIME_OUT is out of range) =>
    RETURN_CODE := INVALID_PARAM;

normal
  if (CURRENT_VALUE > 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 process is error handler process) =>
    RETURN_CODE := INVALID_MODE;
  elsif (TIME_OUT is infinite) then
    set the current process state to WAITING;
    insert this process in the semaphore process queue at the position
      specified by the queuing discipline;
    ask for process scheduling;
    -- The current process is blocked and returns in READY state by a
    -- SIGNAL_SEMAPHORE service request on that semaphore from another
    -- process (except if another process
    -- suspended it)
    RETURN_CODE := NO_ERROR;
  else
    -- ((TIME_OUT > 0) and (Current value = 0))
    set the current process state to WAITING;
    insert this process in the semaphore process queue at the position
      specified by the queuing discipline;
    initiate a time counter with duration TIME_OUT;
    ask for process scheduling;
    -- The current process is blocked and will be removed from that queue and
    -- return in READY state by expiration of time-out or by a
    -- SIGNAL_SEMAPHORE service request on that semaphore from another process
    -- (except if another process suspended it)
    if (expiration of the time-out) then
      RETURN_CODE := TIMED_OUT;
    else
      RETURN_CODE := NO_ERROR;
      -- The next SIGNAL 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

<u>Return Code</u>	<u>Value</u>	<u>Commentary</u>
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 process is 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 is expired

3.0 SERVICE REQUIREMENTS

3.7.2.3.3 SIGNAL_SEMAPHORE

The SIGNAL_SEMAPHORE service request increments the current value of the specified semaphore. If processes are waiting on that semaphore the first process of the queue is moved from the waiting state to the ready state. 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 (all the semaphores are already available) =>
      -- (current value = maximum value) for the specified semaphore
      RETURN_CODE := NO_ACTION;

  normal
    if (no processes are waiting on that semaphore) then
      increment the current value of the specified semaphore;
      RETURN_CODE := NO_ERROR;
    else
      remove the first process from the semaphore queue;
      if (this process is waiting on that semaphore with a time-out) then
        Stop the affected time counter;
      end if;
      move it from the WAITING to the READY state (except if 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.

<u>SIGNAL_SEMAPHORE</u>	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	SEMAPHORE_ID does not identify an existing semaphore
NO_ACTION	Current value of SEMAPHORE_ID = maximum value

3.7.2.3.4 GET_SEMAPHORE_ID

The GET_SEMAPHORE_ID service request allows the current process to get the identifier of a semaphore giving its 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;
```

3.0 SERVICE REQUIREMENTS

```

normal
    SEMAPHORE_ID := (ID of the semaphore named SEMAPHORE_NAME);
    RETURN_CODE  := NO_ERROR;

end GET_SEMAPHORE_ID;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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 waiting processes);
    RETURN_CODE := NO_ERROR;

end GET_SEMAPHORE_STATUS;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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 which may wait for it. An event must be created during the initialization mode 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.

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

```

3.0 SERVICE REQUIREMENTS

```

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 := (ID of unallocated event);
  Set the event state to DOWN;
  RETURN_CODE := NO_ERROR;

end CREATE_EVENT;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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 in up state. All the processes waiting on that event are moved from the waiting state to the ready state. A scheduling takes place.

```

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 in 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;
      move each of these processes from the WAITING state to READY state (except
        if 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 SET_EVENT;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	EVENT_ID does not identify an existing event

3.0 SERVICE REQUIREMENTS

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
  Sets the specified event in 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).

```

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) =>
    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 process is error handler process) =>
    RETURN_CODE := INVALID_MODE;
  elsif (TIME_OUT is infinite) then
    set the current process state to WAITING;
    ask for process scheduling;
    -- It will return in READY state by a SET_EVENT service request from
    -- another process (except if another process suspended it)
    RETURN_CODE := NO_ERROR;
  else
    -- ((TIME_OUT) > 0 and (event is DOWN))
    set the current process state to WAITING;
    initiate a time counter with duration TIME_OUT;
    ask for process scheduling;
    -- It will return in READY state by expiration of time-out or by a
    -- SET_EVENT service request from another process (except if another

```

3.0 SERVICE REQUIREMENTS

```

-- 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.

<u>Return Code Value</u>	<u>Commentary</u>
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 process is error handler process) and time-out is not zero
NOT_AVAILABLE	The event is in the DOWN state
TIMED_OUT	The specified time-out is expired

3.7.2.4.5 GET_EVENT_ID

The GET_EVENT_ID SERVICE allows the current process to get the identifier of an event giving its 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 := (ID of the event named NAME);
    RETURN_CODE := NO_ERROR;

end GET_EVENT_ID;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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;

```

3.0 SERVICE REQUIREMENTS

```

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

end GET_EVENT_STATUS;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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 partition level error are specified in the Module HM and Partition HM tables by the system integrator. 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 implementation 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;

```

The RETURN_CODE_TYPE is common to all APEX services.

3.0 SERVICE REQUIREMENTS

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 if it detects an erroneous behavior. 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.

<u>Return Code Value</u>	<u>Commentary</u>
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 started by the O/S in case of a process level error (e.g., deadline missed, numeric error, stack overflow). It is executed during the partition window. 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 (lock level not equal to zero). It stops itself by 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 cannot be used to correct an error (e.g., limit a value in case of overflow). The faulty process can continue its execution only in 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

3.0 SERVICE REQUIREMENTS

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 (the error handler process is already created) =>
      RETURN_CODE := NO_ACTION;
    when (insufficient storage capacity for the creation of the specified
      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 with the highest priority,
    ENTRY_POINT and STACK_SIZE attributes;
    -- This special process will be started by the O/S in case of error
    RETURN_CODE := NO_ERROR;

end CREATE_ERROR_HANDLER;

```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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 faulty, this service must be called in a loop in the error handler until there are no more processes in error.

3.0 SERVICE REQUIREMENTS

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 put 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;
    -- the error handler process cannot be restarted for the same error
    RETURN_CODE := NO_ERROR;

end GET_ERROR_STATUS;
```

The return codes for this service are explained below.

<u>Return Code Value</u>	<u>Commentary</u>
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.

```

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 out of range) =>
      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
```

3.0 SERVICE REQUIREMENTS

```
    handler process is not created) then 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	
<u>Return Code Value</u>	<u>Commentary</u>
NO_ERROR	Successful completion
INVALID_PARAM	LENGTH is out of range
INVALID_PARAM	ERROR_CODE is not APPLICATION_ERROR

COMMENTARY

This service was defined in Supplement 1 (dated October 16, 2003). This specification allows only APPLICATION_ERROR to be passed by the health monitor process to the partition health manager. Other errors must be identified via potentially non-portable use of the message parameter. The resolution of this issue will be discussed in the development of Supplement 3. Readers of this document should be aware of this development.

4.0 COMPLIANCE TO APEX INTERFACE

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.

5.0 XML CONFIGURATION TABLES

5.1 XML Configuration Specification.

Section 2.5.2 of this document states that: “Configuration tables are static data areas accessed by the operating system. They cannot be accessed directly by applications; however they are not built as part of the operating system.” This section defines the mechanism for specifying configuration data, in a standard format, that does not depend on the implementation of the ARINC 653 OS. 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 systems 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 the form for the loaded configuration table. The core software implementer may choose to do this, but it is not required.

The ARINC 653 XML-Schema defines the structure of the data needed to specify any ARINC 653 configuration. The XML-Schema is extensible; therefore, the ARINC 653 OS implementers can extend the schema for a particular implementation.

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. In doing so, the core system should function in a manner that is complaint with this standard. Optional configuration items are defined, and they may be used to further tailor, or extend the functionality of the core system. It should not be expected that an ARINC 653 compliant application require any optional configuration information in order to function in a manner that is complaint with this standard. This definition covers only the configuration of the ARINC 653 compliant core module. The larger system may require additional configuration data (e.g. sensor/effector configuration, data bus communications configuration, etc.), which is 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.

An ARINC 653 XML-Schema as defined in Appendix H defines the structure of the XML instance file in which the configuration data parameters are specified. Graphical documentation of the XML-Schema is provided in Appendix G and an example XML instance file is shown in Appendix I.

These following sections will provide an overview of the XML-Schema for ARINC 653 and then discuss how the schema is used to produce and verify the ARINC 653 configuration instance.

5.2 XML-Schema

The XML-Schema is the reference standard to which instance files are defined. The XML-Schema defines the structure of the data. The XML-Schema consists of tagged pairs that describe the attributes and their relationship to the whole.

The ARINC 653 XML Schema has been constructed based on Figure 5.2-1. The square boxes represent the schema elements and the attributes are represented by the text inside the box. This diagram represents the major elements and their attributes. The figure shows the major element relationships used to define the ARINC 653 XML-Schema. The complete details of the schema are found in the detailed schema definition (Appendix G and H).

5.0 XML CONFIGURATION TABLES

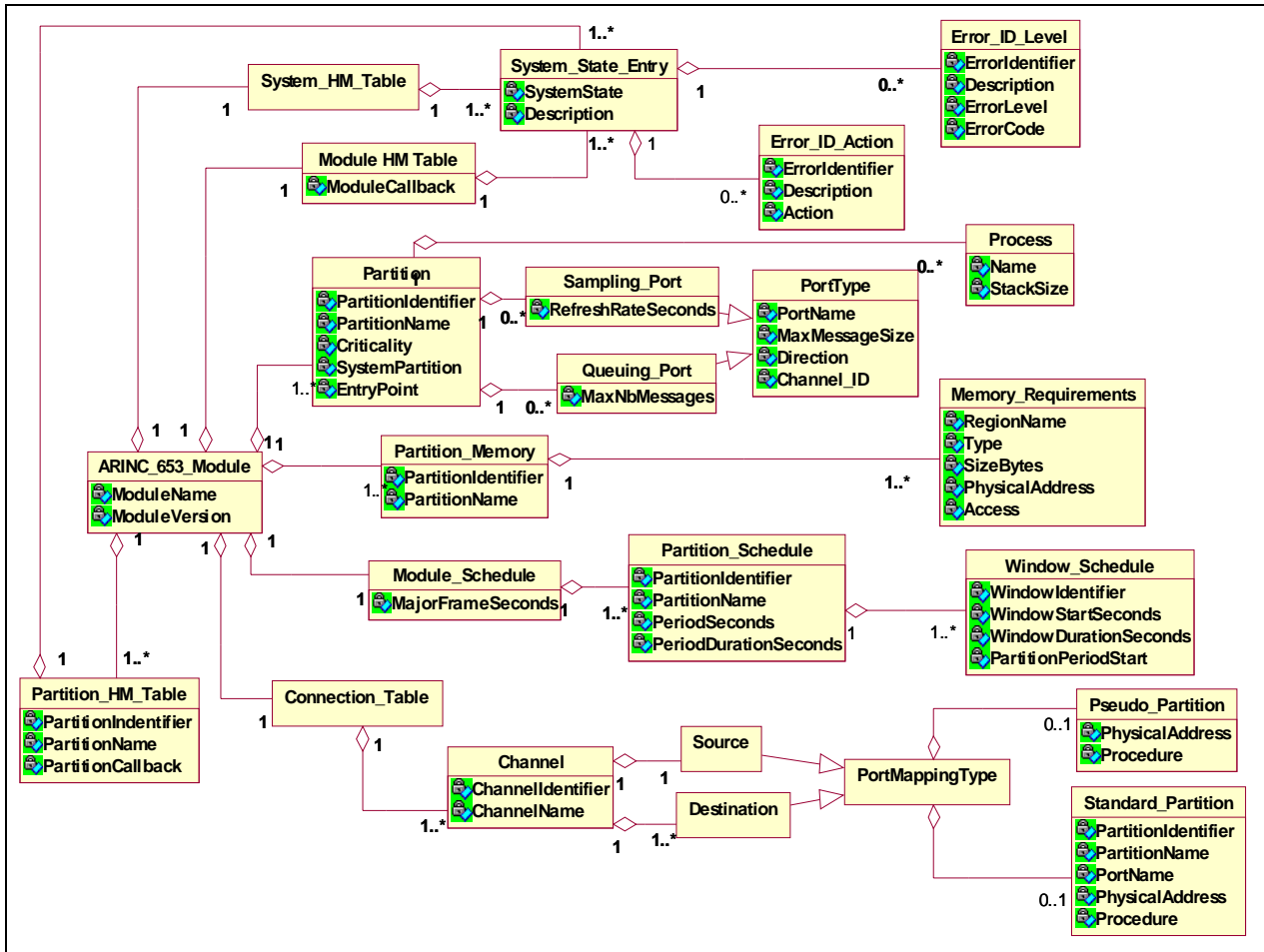


Figure 5.2-1 XML Schema Element Relationships

As can be seen by the figure the following statements can be made:

1. An ARINC 653 Module has seven major elements one or more Partitions, a System HM Table, a Module HM Table, a Connection Table, one or more Partition HM Tables, one or more Partition Memory elements, and a Module Schedule.
2. Each partition has five attributes and zero or more sampling and queuing port elements. In addition there is an optional Process element and its associated attributes.
3. Sampling or queuing ports each have a unique attribute and share common attributes defined by PortType.
4. The System HM Table is made up of 1 or more System State Entries that in turn is made up of 1 or more Error ID Levels. The id levels define the combination as a module, partition, or process level error. In addition process level errors are mapped to the predefined ARINC 653 error codes.
5. The Module and Partition HM Tables are similar to the System HM table with the exception being there is no Error ID Levels but rather there are Error ID Actions. These actions define what the core OS software is to do when the error occurs in a specific system state.
6. The Module Schedule is made up of Partition Schedule elements, which in turn are made up of Window schedule elements. The module schedule attribute defines the major frame rate for scheduling each partition and the mapping of each partition to one or more partition windows (Window Schedule).

5.0 XML CONFIGURATION TABLES

7. The connection table defines the global channel identifications to their port connections. The Channel has one Source and 1 or more Destinations. The Source and Destinations can be either Standard Partitions with a set of attributes or a Pseudo Partition (external to the module) with a set of attributes.
8. The Partition Memory defines the regions of memory that each partition has access to.

This set of relationships forms the basis used to define the detailed XML-Schema defined in the Appendices.

5.3 XML Processes

This section describes an example how the XML-Schema is developed and used for defining XML based ARINC 653 Configuration files. There are five major steps involved in defining and using an XML-Schema that are shown graphically in Figure 5.3-1 and described below.

1. The ARINC 653 Working group defines the ARINC 653 XML-Schema. The ARINC 653 XML Schema is the output produced by this group.
2. The ARINC 653 core OS implementer may develop a translator tool. The translator tool uses the ARINC 653 XML-Schema as the reference for converting an instance file into the configuration table format used by the core OS implementation.
3. The system integrator and the application developer may define the XML instance file for the configuration being integrated. The XML instance is based on the ARINC 653 XML-Schema.
4. The system Integrator and the application developer may validate the XML instance file. A validation tool can be used to verify the instance file is well formed and valid against the ARINC 653 XML-Schema.
5. The system integrator may use the ARINC 653 implementer's translator to convert the XML instance file into the configuration table format used by the core OS implementation.

```

    usecaseDiagram
        actor ARINC653WG as ARINC 653 WG
        actor ARINC653Impl as ARINC 653 Implementers
        actor SysIntegrator as System Integrator Application Developer

        usecase 1 as 1 Define XML-Schema
        usecase 2 as 2 Provide Translator
        usecase 3 as 3 Define XML Instance
        usecase 4 as 4 Validate Instance
        usecase 5 as 5 Translate XML Instance

        ARINC653WG --> 1
        ARINC653Impl --> 2
        1 --> 2
        1 --> Schema[<<XML Schema>> ARINC 653 XML-Schema]
        2 --> Schema
        Schema --> 3
        SysIntegrator --> 3
        SysIntegrator --> 4
        3 --> InstanceFile[<<XML Instance>> ARINC 653 Instance File]
        InstanceFile --> 4
        InstanceFile --> 5
        4 --> 5
        5 --> ConfigTable[ARINC 653 Implementer's Format Configuration Table]
    
```

The diagram illustrates the workflow for ARINC 653 XML Schema and Instance processing. It involves three main actors: ARINC 653 WG, ARINC 653 Implementers, and System Integrator Application Developer. The process consists of five numbered use cases: 1. Define XML-Schema, 2. Provide Translator, 3. Define XML Instance, 4. Validate Instance, and 5. Translate XML Instance. The workflow starts with ARINC 653 WG defining the XML Schema (1), which is then provided to the ARINC 653 Implementers (2). The ARINC 653 Implementers provide the translator to the ARINC 653 XML-Schema (represented by a yellow box). The ARINC 653 XML-Schema is then used to define the XML Instance (3). The System Integrator Application Developer defines the XML Instance (3) and validates it (4). The validated XML Instance is then translated (5) into the ARINC 653 Implementer's Format Configuration Table. A note indicates that the XML Schema is defined for each unique 653 module.

Figure 5.3-1 Defining and Using XML-Schema and XML Instance Files

**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).

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)

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)

**APPENDIX A
GLOSSARY****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)

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.

Core Processor

A Module that contains at least the processing resources and memory. (ARINC Report 651)

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.

**APPENDIX A
GLOSSARY**

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 period of 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)

**APPENDIX A
GLOSSARY****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)

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)

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)

**APPENDIX A
GLOSSARY**

Interchangeability

When either the current element or its replacement satisfies the same functional and interface requirements.

Interpartition

Refers to any communication conducted between partitions.

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 Frame (MAF)

The major time frame is defined as a multiple of the least common multiple of all partition periods in the module. Each major time frame contains identical partition scheduling windows.

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.

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 core 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 Windows

Periods of time during which resources are allocated to particular processes.

Periodic

Occurs cyclically with a fixed time period.

**APPENDIX A
GLOSSARY****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)

Pseudo-Partition

A device, subsystem, or software, which is not an ARINC 653 hosted application, that is capable of communicating with an ARINC 653 hosted application via ARINC 653 channels. From a module's perspective, a partition external to the module can be considered a Pseudo-partition.

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)

Reliability

Reliability at a time t is given by R(t), where:

$R(t) = \text{Prob (continuous service from 0 to t)}$. (ARINC Report 652)

**APPENDIX A
GLOSSARY****Reliability, Software**

The probability of failure-free operation of a software component or system in a specified environment for a specified time. Size and complexity appear to be the main factors influencing reliability, as well as the amount of testing performed. Components of software reliability are measurement, estimation, and prediction:

Measurement - uses failure interval data obtained by running the program in its actual operating environment.

Estimation - uses failure interval data from a test environment; used to determine present or future reliability.

Prediction - uses program characteristics, no failure intervals, to determine software reliability. Takes factors such as size and complexity into account. This is the only reliability component that can be performed during phases prior to test. (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 core processor, 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 period of execution.

A software partition should be allowed to consume shared I/O resources only during its period of execution.

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.

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.

**APPENDIX A
GLOSSARY****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 who configures 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).

Test Coverage

The degree to which a test plan exercises the software, subdivided into (a) requirements test coverage - the degree to which the tests show compliance with the requirements and (b) software structure test coverage - the degree to which the tests exercise the actual implementation. (ARINC Report 652)

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 Electronics 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 Performance 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

C-1.0 Purpose

This section describes specification grammar for a structured language to describe the functional requirements of APEX services.

C-2.0 Functional Requirements

C-2.1 Proposed Method

The description of functional requirements is composed of:

- A specification of the APEX service interface.
- A description of each service.

C-2.1.1 Specification of the Service Interface

The APEX services are spread over several modules called packages, each including services which are logically related i.e. which use common data types.

Specifying the service interface consists in specifying each package successively. The name of the package is chosen so that it suggests the name of the object or group of objects which are managed by the services of the package (for example Partition, Processes, Resources, Buffers, Events).

The interface is entirely specified in a structured language.

Intentionally, this structured language is very simple. Each package specification includes definitions of:

- Numbers.
- Data types.
- Constants.
- Services.

However, definitions which are needed by several packages may be gathered in a separate package.

A service definition contains the name of the service (ex: Create, Send, Receive) and a list of formal parameters. A mode is associated to each formal parameter; the mode indicates how the actual parameter is used by the service:

- Mode input: read only
- Mode input-output: read and updated
- Mode output: written only

C-2.1.2 Description of Each Service

The description of a service consists of:

- A short definition of the functionality of the service.
- A full description of its semantics.

APPENDIX C

APEX SERVICES SPECIFICATION GRAMMAR

The semantic description gives the algorithm of the treatment performed by the executive, independently of any implementation consideration. However, relying the semantic description upon a conceptual model may sometimes improve understandability. This underlying model must be introduced before starting the description of the services which need it.

The algorithm is composed of two parts:

- An error part which describes error handling due to incorrect values of actual input or input-output parameters.
- The order of the error test is not defined by this standard.

A normal part which describes the treatment to be performed when no error is detected by the service.

The whole semantic description uses a structured language. This language may be useful to obtain an easy to read description.

Note: 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.

C-2.1.3 Definition of a Structured Language

The structured language described in this paper is based on the Ada 83 language syntax with some simplifications and additional rules to correctly specify the functional requirements of the APEX. The goal of this paragraph is to give an entire description of the proposed language instead of listing differences from Ada in order to help people not familiar with Ada to understand this structured language.

The language is described with YACC and LEX syntax. LEX is a program generator designed for lexical processing of character input streams. YACC is a parser generator. It provides a general tool for describing the structure of an input to a computer program, together with code to be invoked as each item is recognized. The YACC specification file is given below. It can be used to define in a first step the structured language proposed for APEX services. In a second step, it could be used to write a tool for verifying the structure of the APEX specification. (this verification can be very large, e.g. verify that every used type is already defined).

C-2.1.3.1 Structure of the YACC File

A full YACC specification file looks like:

- declarations
- %%
- rules
- %%
- programs

APPENDIX C

APEX SERVICES SPECIFICATION GRAMMAR

The declaration section and the programs section may be empty. Spaces, tabs and newlines are ignored.

A grammar rule has the form

A : BODY ;

where A represents a nonterminal name and BODY represents a sequence of zero or more names and literals. The colon and the semicolon are YACC punctuation. The names used in the body of a grammar rule may represent tokens or nonterminal names.

If there are several grammar rules with the same left hand side, the vertical bar `|' can be used to avoid rewriting the left hand side. In addition, the semicolon at the end of a rule can be dropped before a vertical bar. Thus, the grammar rules:

```
A :      B      C      D      ;
A :      E      F      ;
A :      G      ;
```

can be given as:

```
A :      B      C      D
  |      E      F
  |      G
  ;
```

Names representing tokens must be declared; this is most simply done by writing

%token name

in the declarations section. Every name not defined in the declarations section is assumed to represent a nonterminal symbol. Every nonterminal symbol must appear on the left-hand side of at least one rule. Of all the nonterminal symbols, one, called the start symbol, has particular importance. The parser is designed to recognize the start symbol; thus, this symbol represents the largest, most general structure described by the grammar rules. By default, the start symbol is taken to be the left hand sided of the first grammar rule in the rules section. It is possible, and in fact desirable, to declare the start symbol explicitly in the declarations section using:

%start symbol

C-2.1.3.2 Grammar Rules (YACC File)

```
/*      DECLARATIONS                                */
/*                                                    */
/*      Characters                                    */
%token SEMICOLON      /* ; */
%token COMMA          /* , */
%token DOT            /* . */
%token LEFT_PARENTH   /* ( */
%token RIGHT_PARENTH  /* ) */
%token COLON          /* : */
%token ANYCHAR        /* Any other Character */
```

APPENDIX C

APEX SERVICES SPECIFICATION GRAMMAR

```

/*      Set of characters                                     */
%token ARROW          /* => */
%token IS_EQUAL_TO    /* := */
%token IDENTIFIER     /* A letter followed by letters, digits or _ */
%token ANYSTRING      /* Any characters between quotation marks */

/*      List of Keywords (can be written in upper or lower case) */
%token ARRAY
%token CASE CONSTANT
%token DELTA DIGIT
%token ELSE ELSIF END ERROR
%token FOR
%token IF IN IS
%token LOOP
%token NORMAL
%token OF OUT
%token PACKAGE PROCEDURE
%token RANGE RECORD REVERSE
%token SELECT
%token THEN TYPE
%token UNTIL
%token WHEN
%token WHILE

/*      GRAMMAR RULES                                     */
%start specification
%%

specification      :
                    | package_declarations
                    | semantic_descriptions
                    | package_declarations semantic_descriptions
                    ;

/* THE RULES BELOW ARE ONLY USED FOR THE SPECIFICATION OF THE INTERFACE */

package_declarations : package_declaration
                    | package_declarations package_declaration
                    ;

package_declaration  : PACKAGE package_id IS basic_declaration_list END package_id
                    ;
                    SEMICOLON

basic_declaration_list : basic_declaration
                    | basic_declaration_list basic_declaration
                    ;

basic_declaration    : constant_declaration
                    | type_declaration
                    | service_declaration
                    ;

constant_declaration : constant_id COLON CONSTANT type_indication IS_EQUAL_TO expression
                    ;
                    SEMICOLON
                    | constant_id COLON CONSTANT IS_EQUAL_TO expression SEMICOLON

constant_id          : IDENTIFIER
                    ;

type_declaration      : TYPE type_id IS type_definition SEMICOLON
                    ;

service_declaration   : service_spec SEMICOLON
                    ;

```

APPENDIX C

APEX SERVICES SPECIFICATION GRAMMAR

```

type_definition      : enum_type_definition
                    | integer_type_definition
                    | fixed_point_type_definition
                    | array_type_definition
                    | record_type_definition
                    | miscellaneous_type_definition
                    ;

enum_type_definition : LEFT_PARENTH enum_literal_list RIGHT_PARENTH
                    ;

enum_literal_list    : enum_literal
                    | enum_literal_list COMMA enum_literal
                    ;

enum_literal         : IDENTIFIER
                    ;

integer_type_definition : RANGE range
                    ;

fixed_point_type_definition : DELTA fixed_point_precision RANGE range
                    ;

fixed_point_precision : expression
                    ;

array_type_definition : ARRAY LEFT_PARENTH index_type_indication_list RIGHT_PARENTH OF
                    component_type_indication
                    ;

index_type_indication_list : index_type_indication
                    | index_type_indication_list COMMA index_type_indication
                    ;

index_type_indication : type_indication
                    ;

component_type_indication : type_indication
                    ;

record_type_definition : RECORD component_list END RECORD
                    ;

component_list       : component_declaration
                    | component_list component_declaration
                    ;

component_declaration : component_id COLON component_type_indication SEMICOLON
                    ;

component_id         : IDENTIFIER
                    ;

miscellaneous_type_definition : ANYSTRING
                    ;

```

APPENDIX C

APEX SERVICES SPECIFICATION GRAMMAR

/* THE RULES BELOW ARE ONLY USED FOR THE SPECIFICATION OF THE SEMANTIC DESCRIPTION */

```

semantic_descriptions      : semantic_description
                           | semantic_descriptions semantic_description
                           ;

semantic_description       : service_spec IS ERROR alternative_list NORMAL sequence_of_statements
                           END service_id
                           SEMICOLON
                           | service_spec IS NORMAL sequence_of_statements END service_id
                           SEMICOLON
                           ;

alternative_list           : alternative
                           | alternative_list alternative
                           ;

alternative                : WHEN condition ARROW sequence_of_statements
                           ;

sequence_of_statements     : statement
                           | sequence_of_statements statement
                           ;

statement                  : simple_statement
                           | compound_statement
                           ;

simple_statement            : expression SEMICOLON
                           | expression IS_EQUAL_TO expression SEMICOLON
                           ;

compound_statement         : if_statement
                           | case_statement
                           | simple_loop_statement
                           | counter_loop_statement
                           | while_loop_statement
                           | blocked_wait_statement
                           ;

if_statement               : IF condition THEN then_statements END IF SEMICOLON
                           | IF condition THEN then_statements ELSE sequence_of_statements END IF
                           SEMICOLON
                           ;

then_statements            : sequence_of_statements
                           | sequence_of_statements elsif_statements
                           ;

elsif_statements           : ELSIF condition THEN sequence_of_statements
                           | elsif_statements ELSIF condition THEN sequence_of_statements
                           ;

case_statement             : CASE expression IS alternative_list END CASE SEMICOLON
                           ;

simple_loop_statement       : LOOP sequence_of_statements END LOOP SEMICOLON
                           ;

counter_loop_statement     : FOR loop_parameter_spec LOOP sequence_of_statements END LOOP
                           SEMICOLON
                           ;

while_loop_statement       : WHILE condition LOOP sequence_of_statements END LOOP SEMICOLON
                           ;

```

APPENDIX C APEX SERVICES SPECIFICATION GRAMMAR

```

condition                : LEFT_PARENTH expression RIGHT_PARENTH
                           ;

blocked_wait_statement   : SELECT alternative_list END SELECT SEMICOLON
                           ;

loop_parameter_spec      : IDENTIFIER IN range
                           | IDENTIFIER IN REVERSE range
                           ;

/* THE RULES BELOW ARE USED BOTH IN THE SPECIFICATION OF THE INTERFACE AND THE SEMANTIC
DESCRIPTION */

expression               : ANYCHAR
                           | ANYSTRING
                           | IDENTIFIER
                           | expression ANYCHAR
                           | expression ANYSTRING
                           | expression IDENTIFIER
                           ;

service_spec             : PROCEDURE service_id
                           | PROCEDURE service_id formal_part
                           ;

service_id               : IDENTIFIER
                           ;

formal_part              : LEFT_PARENTH parameter_spec_list RIGHT_PARENTH
                           ;

parameter_spec_list      : parameter_spec
                           | parameter_spec_list SEMICOLON parameter_spec
                           ;

parameter_spec           : in_parameter_spec
                           | inout_parameter_spec
                           | out_parameter_spec
                           ;

in_parameter_spec        : parameter_id COLON IN type_indication
                           ;

inout_parameter_spec     : parameter_id COLON IN OUT type_indication
                           ;

out_parameter_spec       : parameter_id COLON OUT type_indication
                           ;

parameter_id             : IDENTIFIER
                           ;

type_indication          : type_id
                           | package_id DOT type_id
                           ;

package_id               : IDENTIFIER
                           ;

type_id                  : IDENTIFIER
                           ;

range                    : expression DOT DOT expression
                           ;

%%

```

APPENDIX C

APEX SERVICES SPECIFICATION GRAMMAR

C-2.1.3.3 Lexical Rules (Lex File)

```
%{
/* ===== */
/*           DEFINITIONS           */
/* ===== */
int line_count = 1;
}%
COMMENT      --.*\n
IDENTIFIER   [A-Za-z][A-Za-z0-9_]*
ANYSTRING    \"^[^\"]*\"
ARRAY        [Aa][Rr][Rr][Aa][Yy]
CASE         [Cc][Aa][Ss][Ee]
CONSTANT     [Cc][Oo][Nn][Ss][Tt][Aa][Nn][Tt]
DELTA        [Dd][Ee][Ll][Tt][Aa]
DIGIT        [Dd][Ii][Gg][Ii][Tt]
ELSE         [Ee][Ll][Ss][Ee]
ELSIF        [Ee][Ll][Ss][Ii][Ff]
END          [Ee][Nn][Dd]
ERROR        [Ee][Rr][Rr][Oo][Rr]
FOR          [Ff][Oo][Rr]
IF           [Ii][Ff]
IN           [Ii][Nn]
IS           [Ii][Ss]
LOOP         [Ll][Oo][Oo][Pp]
NORMAL       [Nn][Oo][Rr][Mm][Aa][Ll]
OF           [Oo][Ff]
OUT          [Oo][Uu][Tt]
PACKAGE      [Pp][Aa][Cc][Kk][Aa][Gg][Ee]
PROCEDURE    [Pp][Rr][Oo][Cc][Ee][Dd][Uu][Rr][Ee]
RANGE        [Rr][Aa][Nn][Gg][Ee]
RECORD       [Rr][Ee][Cc][Oo][Rr][Dd]
REVERSE      [Rr][Ee][Vv][Ee][Rr][Ss][Ee]
SELECT       [Ss][Ee][Ll][Ee][Cc][Tt]
THEN         [Tt][Hh][Ee][Nn]
TYPE         [Tt][Yy][Pp][Ee]
UNTIL        [Uu][Nn][Tt][Ii][Ll]
WHEN         [Ww][Hh][Ee][Nn]
WHILE        [Ww][Hh][Ii][Ll][Ee]
/* ===== */
/*           RULES           */
/* ===== */
%%
{COMMENT}      ;
{ANYSTRING}    { return ( ANYSTRING); }
{ARRAY}        { return ( ARRAY); }
{CASE}         { return ( CASE); }
{CONSTANT}     { return ( CONSTANT); }
{DELTA}        { return ( DELTA); }
{DIGIT}        { return ( DIGIT); }
{ELSE}         { return ( ELSE); }
{ELSIF}        { return ( ELSIF); }
{END}          { return ( END); }
{ERROR}        { return ( ERROR); }
{FOR}          { return ( FOR); }
{IF}           { return ( IF); }
{IN}           { return ( IN); }
{IS}           { return ( IS); }
{LOOP}         { return ( LOOP); }
{NORMAL}       { return ( NORMAL); }
{OF}           { return ( OF); }
{OUT}          { return ( OUT); }
{PACKAGE}      { return ( PACKAGE); }
{PROCEDURE}    { return ( PROCEDURE); }
{RANGE}        { return ( RANGE); }
{RECORD}       { return ( RECORD); }
```


APPENDIX C

APEX SERVICES SPECIFICATION GRAMMAR

```

{REVERSE}      { return ( REVERSE); }
{SELECT}       { return ( SELECT); }
{THEN}         { return ( THEN); }
{TYPE}         { return ( TYPE); }
{UNTIL}        { return ( UNTIL); }
{WHEN}         { return ( WHEN); }
{WHILE}        { return ( WHILE); }
{IDENTIFIER}   { return ( IDENTIFIER); }
\n             { line_count++; }
[ \t]*         ;
=>             { return (ARROW); }
:=             { return (IS_EQUAL_TO); }
\;             |
\:             |
\.             |
\,             |
\(             |
\)             { return (yytext[0]); }
.             { return (ANYCHAR); }
%%
/* ===== */
/*          ROUTINES          */
/* ===== */
main()
{
    return (yyparse());
}
yyerror(s)
char *s;
{
    fprintf(stderr,"%s at line %d\n",s,line_count);
}

```

APPENDIX D ADA INTERFACE SPECIFICATION

- **The Ada bindings**

This appendix provides two Ada bindings. Appendix D.1 is a version that is compatible with both Ada 83 and Ada 95. Appendix D.2 is an Ada 95 binding that provides a number of usability improvements. It includes wrapper packages to provide backward-compatibility to the Ada 83 binding in D.1.

- **Rationale for Ada and C binding modification proposal**

The objective is to have the same definitions and representations of the interface in both the Ada and C languages.

The two bindings have been realigned in the same order, and define the same items.

The first item resolves the current inconsistencies with timer representation, and proposes a common format compatible with both Ada and C implementations.

The second item explains why the MESSAGE_AREA_TYPE does not convey the correct meaning of the required type, and proposes a more appropriate terminology.

The third item addresses the dissimilarity of string representation in the Ada and C languages, and proposes a method of handling the APEX NAME type in both languages.

The final item addresses the dissimilarity in the storage of data types between not only the Ada and C language but also potentially between different development environments. For the Ada APEX interface, this is addressed using representation clauses to define the layout and storage requirements for pertinent APEX data types, and then, for those data types designated as APEX base types, using these base types to derive the remaining APEX types.

- **Time representation**

In the Ada interface specification (appendix D), time is represented as:

```
type SYSTEM_TIME_TYPE is new DURATION; -- implementation dependent
```

```
INFINITE_TIME_VALUE : constant SYSTEM_TIME_TYPE := SYSTEM_TIME'FIRST -- i.e. 0
```

In the C interface specification (appendix E), time is represented as:

```
typedef APEX_INTEGER SYSTEM_TIME_TYPE
```

```
define MAX_TIME_VALUE 8640000
```

```
#define INFINITE_TIME_VALUE MAX_TIME_VALUE + 1 -- i.e. 8640001
```

Assuming that the Ada 'implementation dependent' type is on 32 bits, INFINITE_TIME_VALUE is not the same for Ada and C. For Ada a TIMED_WAIT (0) means infinite wait, but for C it means 'ask for process scheduling'.

There is now one time representation:

A new representation for SYSTEM_TIME_TYPE using a signed 64-bit value with a resolution of 1 nanosecond.

APPENDIX D ADA INTERFACE SPECIFICATION

- **Message area definition**

MESSAGE_AREA_TYPE is currently declared as follows:

In Ada : MESSAGE_AREA_TYPE is new SYSTEM.ADRRESS;

In C : typedef APEX_CHAR MESSAGE_AREA_TYPE;

The MESSAGE_AREA_TYPE does not convey the correct meaning: the 'area' referred to is neither an address (as Ada defines it) nor is it limited to a byte (as C defines it). Moreover, the use of MESSAGE parameters defined using this type causes confusion because it is not the full array that is being referred to but only its address.

Proposal

Either in Ada or C, the parameters passed to the OS for message manipulation are its ADDRESS and its LENGTH.

So the proposal is to change the current definitions of MESSAGE_AREA_TYPE to the following:

In Ada : MESSAGE_ADDR_TYPE is new SYSTEM.ADRRESS;

In C : typedef APEX_BYTE * MESSAGE_ADDR_TYPE;

For example the READ_SAMPLING_MESSAGE service is defined as :

```
procedure READ_SAMPLING_MESSAGE
(SAMPLING_PORT_ID : in SAMPLING_PORT_ID_TYPE;
 MESSAGE          : out MESSAGE_AREA_TYPE;      -- in fact, the parameter is an address
                                                    -- that is passed in (the application give
                                                    -- to the OS the address it wants the OS
                                                    -- writes the data).

LENGTH           : out MESSAGE_SIZE_TYPE;
VALIDITY         : out BOOLEAN;
RETURN_CODE      : out RETURN_CODE_TYPE) is
```

With the new definition the meaning of the parameters become more clear:

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

- **String representation (NAME_TYPE)**

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 shall treat the contents of NAME_TYPE objects as case insensitive.

The use of 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.

APPENDIX D
ADA INTERFACE SPECIFICATION

- **Datatype representation:**

Dissimilar languages, and even dissimilar implementations of the same language, may yield different data storage representations for the same datatype. This is an especially significant issue when attempting to bind the OS with a partition, where both are written in a different implementation language. For the Ada APEX interface, this is resolved by explicitly defining the storage layout and usage via Ada representation clauses. In particular, for those APEX data types declared as enumerated types, the representation clause explicitly denotes the order and values assigned to the enumeration literals within that type, thus providing the required mapping onto the C APEX equivalents.

The introduction of base APEX types (e.g. APEX_INTEGER) with their defined storage representation allows subsequent scalar APEX types to be derived from these base types without the need to apply further representation clauses. It may also be noted that the use of Ada derived types, as compared to Ada subtypes, provides a higher level of language safety due to Ada's protective strong typing. This is because, without explicit type conversion, the inadvertent mixing of derived types within an expression would raise a compile-time error, whereas for subtypes, implicit type conversion to the base subtype would automatically occur and hence such inadvertent mixing would not be easily detected (NB. notwithstanding the error detection of any 'out-of-range' values at runtime).

APPENDIX D ADA INTERFACE SPECIFICATION

APPENDIX D.1

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.
-- The declarations of the services given below are taken from the
-- standard, as are the enumerated types and the names of the others types.
-- However, the definitions given for these others types, and the
-- names and values given below for constants, are all implementation
-- specific.
-- All types have a defined representation clause to enable compatibility
-- between the C and Ada bindings.

-- -----
--
-- Global constant and type definitions
--
-- -----

with SYSTEM;

package APEX_TYPES is

    -- -----
    -- Domain limits
    -- -----

    -- Domain 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
    -- -----

    -- 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
```

APPENDIX D
ADA INTERFACE SPECIFICATION

```
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
```

APPENDIX D **ADA INTERFACE SPECIFICATION**

```
-- -----
-- 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 1..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;
```

APPENDIX D ADA INTERFACE SPECIFICATION

```

-----
--
-- 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 := 63;
```

```
    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;
```

```

    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

```


APPENDIX D

ADA INTERFACE SPECIFICATION

```

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 );

```

**APPENDIX D
ADA INTERFACE SPECIFICATION**

```
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;
```

APPENDIX D

ADA INTERFACE SPECIFICATION

```

-----
--
-- 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 ADA 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 );

```

APPENDIX D
ADA INTERFACE SPECIFICATION

```
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 ADA 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 );

```

APPENDIX D
ADA INTERFACE SPECIFICATION

```
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 );

end APEX_QUEUING_PORTS;
```

APPENDIX D ADA INTERFACE SPECIFICATION

```

-----
--
-- 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 );

```


APPENDIX D

ADA INTERFACE SPECIFICATION

```

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 );

```

APPENDIX D
ADA INTERFACE SPECIFICATION

```
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;
```

APPENDIX D **ADA INTERFACE SPECIFICATION**

```

-- -----
--
-- 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 ADA 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

ADA 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 := 64;

    type ERROR_MESSAGE_SIZE_TYPE is new APEX_TYPES.APEX_INTEGER
        range 1..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 );

```

**APPENDIX D
ADA INTERFACE SPECIFICATION**

```
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;
```

-- =====

APPENDIX D ADA INTERFACE SPECIFICATION

APPENDIX D.2

ADA 95 INTERFACE SPECIFICATION

- **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 ensuing sections.

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 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,

APPENDIX D ADA INTERFACE SPECIFICATION

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 C interfaces and type definitions of Appendix E are the basis for the Ada 95 representation.**

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 ADA INTERFACE SPECIFICATION

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.
-- The declarations of the services given below are taken from the
-- standard, as are the enumerated types and the names of the others types.
-- However, the definitions given for these others types, and the
-- names and values given below for constants, are all implementation
-- specific.
-- 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 --
  -- -----

  -- Domain 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_Queueing_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 --
  -- -----

  -- 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;
  type APEX_Integer is new Interfaces.C.long;
  type APEX_Unsigned is new Interfaces.C.unsigned_long;
```

APPENDIX D ADA INTERFACE SPECIFICATION

```

type APEX_Long_Integer is new Interfaces.Integer_64;
-- 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
    1 .. 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;
Aperiodic           : constant System_Time_Type;
Zero_Time_Value     : constant System_Time_Type;

private
    Infinite_Time_Value : constant System_Time_Type := -1;
    Aperiodic           : constant System_Time_Type := 0;
    Zero_Time_Value     : constant System_Time_Type := 0;
end APEX;

```

APPENDIX D
ADA 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;
```

APPENDIX D ADA INTERFACE SPECIFICATION

```

-- -----
-- --
-- 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 := 0;
    Max_Priority_Value : constant := 249;
    Max_Lock_Level : constant := 32;

    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;

```

APPENDIX D

ADA INTERFACE SPECIFICATION

```

    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;

```

APPENDIX D ADA INTERFACE SPECIFICATION

```

-----
-- --
-- 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;

```

APPENDIX D

ADA INTERFACE SPECIFICATION

```

-----
-- --
-- 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 ADA 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);

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;

```


APPENDIX D

ADA INTERFACE SPECIFICATION

```

-- -----
-- --
-- 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 ADA 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;

```

APPENDIX D **ADA INTERFACE SPECIFICATION**

```

-- -----
-- --
-- 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;

```

APPENDIX D ADA INTERFACE SPECIFICATION

```

-- -----
-- --
-- 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;

```

APPENDIX D **ADA INTERFACE SPECIFICATION**

```

-----
-- --
-- ERROR constant and type definitions and management services --
-- --
-----

```

```

with APEX.Processes;
package APEX.Health_Monitoring is

    Max_Error_Message_Size : constant := 64;

    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 ADA INTERFACE SPECIFICATION

```

-- -----
-- --
-- MODULE_SCHEDULES constant and type definitions and management services --
-- --
-- -----

package APEX.Module_Schedules is

  type Schedule_Id_Type is private;
  Null_Schedule_Id : constant Schedule_Id_Type;

  subtype Schedule_Name_Type is Name_Type;

  type Schedule_Status_Type is record
    Time_Of_Last_Schedule_Switch      : System_Time_Type;
    Current_Schedule                   : Schedule_Id_Type;
    Next_Schedule                      : Schedule_Id_Type;
  end record;

  procedure Set_Module_Schedule
    (Schedule_Id      : in Schedule_Id_Type;
     Return_Code      : out Return_Code_Type);

  procedure Get_Module_Schedule_Status
    (Schedule_Status  : out Schedule_Status_Type;
     Return_Code      : out Return_Code_Type);

  procedure Get_Module_Schedule_Id
    (Schedule_Name    : in Schedule_Name_Type;
     Schedule_Id      : out Schedule_Id_Type;
     Return_Code      : out Return_Code_Type);

private
  Type Schedule_Id_Type is new APEX_Integer;
  Null_Schedule_Id : constant Schedule_Id_Type := 0;

  pragma Convention (C, Schedule_Status_Type);
end APEX.Module_Schedules;

```

APPENDIX D
ADA 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.Module_Schedules;  
package APEX_Module_Schedules renames APEX.Module_Schedules;  
  
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 C INTERFACE SPECIFICATION

• Rationale for ADA and C binding modification proposal

The objective is to have the same definitions and representations of the interface in both the Ada and C languages.

The two bindings have been realigned in the same order, and define the same items.

The first item resolves the current inconsistencies with timer representation, and proposes a common format compatible with both Ada and C implementations

The second item explains why the MESSAGE_AREA_TYPE does not convey the correct meaning of the required type, and proposes a more appropriate terminology.

The third item addresses the dissimilarity of string representation in the Ada and C languages, and proposes a method of handling the APEX NAME type in both languages.

The final item addresses the dissimilarity in the storage of datatypes between not only the Ada and C language but also potentially between different development environments. For the Ada APEX interface, this is addressed using representation clauses to define the layout and storage requirements for pertinent APEX datatypes, and then, for those datatypes designated as APEX base types, using these base types to derive the remaining APEX types.

• Time representation

In the Ada interface specification (appendix D), time is represented as:

```
type SYSTEM_TIME_TYPE is new DURATION; -- implementation dependent
```

```
INFINITE_TIME_VALUE : constant SYSTEM_TIME_TYPE := SYSTEM_TIME'FIRST -- i.e. 0
```

In the C interface specification (appendix E), time is represented as:

```
typedef APEX_INTEGER SYSTEM_TIME_TYPE
```

```
define MAX_TIME_VALUE 8640000
```

```
#define INFINITE_TIME_VALUE MAX_TIME_VALUE + 1 -- i.e. 8640001
```

Assuming that the Ada 'implementation dependent' type is on 32 bits, INFINITE_TIME_VALUE is not the same for Ada and C. For Ada a TIMED_WAIT (0) means infinite wait, but for C it means 'ask for process scheduling'.

There is now one time representation:

A new representation for SYSTEM_TIME_TYPE using a signed a 64-bit value with a resolution of 1 nanosecond.

• Message area definition

MESSAGE_AREA_TYPE is currently declared as follows:

In Ada : MESSAGE_AREA_TYPE is new SYSTEM.ADRESS;

APPENDIX E C INTERFACE SPECIFICATION

In C : typedef APEX_CHAR MESSAGE_AREA_TYPE;

The MESSAGE_AREA_TYPE does not convey the correct meaning: the 'area' referred to is neither an address (as Ada defines it) nor is it limited to a byte (as C defines it). Moreover, the use of MESSAGE parameters defined using this type causes confusion because it is not the full array that is being referred to but only its address.

Proposal

Either in Ada or C, the parameters passed to the OS for message manipulation are its ADDRESS and its LENGTH.

So the proposal is to change the current definitions of MESSAGE_AREA_TYPE to the following:

In Ada : MESSAGE_ADDR_TYPE is new SYSTEM.ADRRESS;

In C : typedef APEX_BYTE * MESSAGE_ADDR_TYPE;

For example the READ_SAMPLING_MESSAGE service is defined as :

```
procedure READ_SAMPLING_MESSAGE
(SAMPLING_PORT_ID : in SAMPLING_PORT_ID_TYPE;
 MESSAGE          : out MESSAGE_AREA_TYPE; -- in fact, the parameter is an address
                                           -- that is passed in (the application give
                                           -- to the OS the address it wants the OS
                                           -- writes the data)

LENGTH           : out MESSAGE_SIZE_TYPE;
VALIDITY         : out BOOLEAN;
RETURN_CODE      : out RETURN_CODE_TYPE) is
```

With the new definition the meaning of the parameters becomes clearer:

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

- **String representation (NAME_TYPE)**

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 shall treat the contents of NAME_TYPE objects as case insensitive.

The use of 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.

- **Datatype representation:**

Dissimilar languages, and even dissimilar implementations of the same language, may yield different data storage representations for the same datatype. This is an especially significant issue when attempting to bind the OS with a partition, where both are written in a different implementation

APPENDIX E
C INTERFACE SPECIFICATION

language. For the Ada APEX interface, this is resolved by explicitly defining the storage layout and usage via Ada representation clauses. In particular, for those APEX datatypes declared as enumerated types, the representation clause explicitly denotes the order and values assigned to the enumeration literals within that type, thus providing the required mapping onto the C APEX equivalents.

The introduction of base APEX types (e.g. APEX_INTEGER) with their defined storage representation allows subsequent scalar APEX types to be derived from these base types without the need to apply further representation clauses. It may also be noted that the use of Ada derived types, as compared to Ada subtypes, provides a higher level of language safety due to Ada's protective strong typing. This is because, without explicit type conversion, the inadvertent mixing of derived types within an expression would raise a compile-time error, whereas for subtypes, implicit type conversion to the base subtype would automatically occur and hence such inadvertent mixing would not be easily detected (NB. notwithstanding the error detection of any 'out-of-range' values at runtime).

APPENDIX E

C INTERFACE SPECIFICATION

```

/* This is a compilable ANSI C specification of the APEX interface.      */
/* The declarations of the services given below are taken from the        */
/* standard, as are the enum types and the names of the others types.    */
/* However, the definitions given for these others types, and the        */
/* names and values given below for constants, are all implementation    */
/* specific.                                                                */

/* This ANSI C specification follows the same structure (package) as     */
/* the Ada specification.                                                 */

/*-----*/
/*                                                                */
/* Global constant and type definitions                                  */
/*                                                                */
/*-----*/

#ifndef APEX_TYPES
#define APEX_TYPES

/*-----*/
/* Domain limits                                                    */
/*-----*/

/* Domain 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_QUEUEING_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 */

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/* Base APEX types */
/*-----*/

/* 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

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* 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        63

#define MAX_LOCK_LEVEL            16

typedef NAME_TYPE                  PROCESS_NAME_TYPE;

typedef APEX_INTEGER               PROCESS_ID_TYPE;

typedef APEX_INTEGER               LOCK_LEVEL_TYPE;

typedef APEX_UNSIGNED              STACK_SIZE_TYPE;

typedef APEX_INTEGER               WAITING_RANGE_TYPE;

typedef APEX_INTEGER               PRIORITY_TYPE;

```

APPENDIX E

C INTERFACE SPECIFICATION

```

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 ();

extern void STOP (
    /*in */ PROCESS_ID_TYPE           PROCESS_ID,
    /*out*/ RETURN_CODE_TYPE          *RETURN_CODE );

```

APPENDIX E

C INTERFACE SPECIFICATION

```

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

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* 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

```


APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* 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

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* QUEUING PORT constant and type definitions and management services */
/*
/*-----*/

#ifndef APEX_QUEUEING
#define APEX_QUEUEING

#define MAX_NUMBER_OF_QUEUEING_PORTS    SYSTEM_LIMIT_NUMBER_OF_QUEUEING_PORTS

typedef NAME_TYPE            QUEUEING_PORT_NAME_TYPE;

typedef APEX_INTEGER         QUEUEING_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;
    } QUEUEING_PORT_STATUS_TYPE;

extern void CREATE_QUEUEING_PORT (
    /*in */ QUEUEING_PORT_NAME_TYPE    QUEUEING_PORT_NAME,
    /*in */ MESSAGE_SIZE_TYPE          MAX_MESSAGE_SIZE,
    /*in */ MESSAGE_RANGE_TYPE         MAX_NB_MESSAGE,
    /*in */ PORT_DIRECTION_TYPE         PORT_DIRECTION,
    /*in */ QUEUEING_DISCIPLINE_TYPE    QUEUEING_DISCIPLINE,
    /*out*/ QUEUEING_PORT_ID_TYPE       *QUEUEING_PORT_ID,
    /*out*/ RETURN_CODE_TYPE            *RETURN_CODE );

extern void SEND_QUEUEING_MESSAGE (
    /*in */ QUEUEING_PORT_ID_TYPE       QUEUEING_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_QUEUEING_MESSAGE (
    /*in */ QUEUEING_PORT_ID_TYPE       QUEUEING_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_QUEUEING_PORT_ID (
    /*in */ QUEUEING_PORT_NAME_TYPE     QUEUEING_PORT_NAME,
    /*out*/ QUEUEING_PORT_ID_TYPE       *QUEUEING_PORT_ID,
    /*out*/ RETURN_CODE_TYPE            *RETURN_CODE );

extern void GET_QUEUEING_PORT_STATUS (
    /*in */ QUEUEING_PORT_ID_TYPE       QUEUEING_PORT_ID,
    /*out*/ QUEUEING_PORT_STATUS_TYPE   *QUEUEING_PORT_STATUS,
    /*out*/ RETURN_CODE_TYPE            *RETURN_CODE );

#endif

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* 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

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* 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

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* 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

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* 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

```

APPENDIX E

C INTERFACE SPECIFICATION

```

/*-----*/
/*
/* ERROR constant and type definitions and management services */
/*
/*-----*/

#ifndef APEX_ERROR
#define APEX_ERROR

#define MAX_ERROR_MESSAGE_SIZE 64

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

This Appendix was added in Supplement 1 to describe the XML-Schema code file.

Appendix G is divided into 2 sections. The first G.1 is a graphical view of the ARINC 653 XML-Schema. The second section G.2 depicts the HM table's mappings to the XML-Schema.




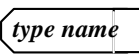
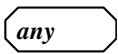
Section G.1 XML-Schema Documentation

The actual XML-Schema code is shown in Appendix H and an example instance file is shown in Appendix I. A tool was used to generate graphics for the documentation shown in this Appendix. There are many commercial off the shelf XML tools which provide a graphical interface for building instance files from the XML-Schema, for example XML Spy 4.4 was used in this appendix to generated graphics.

Figures 1 through 10 graphically capture the ARINC 653 XML-Schema and will be used as references. The figures include annotations on the meaning of each tag 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 XML Spy 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.
	Corresponds to a created type definition. Used to build up the schema and to reuse common definitions.
	Corresponds to the XML definition 'xsd:any'. Allows user defined extensions to the schema.

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

The following type definition, 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.

Simple Type	Name Type
type	restriction of xs:string
facets	minLength 1 maxLength 30

Table 2 NameType Definition

The DecOrHexValueType allows numbers to be entered as integers or hex values. Hex values are preceded by “0x”.

Simple Type	DecOrHexValueType
type	restriction of xs:string
facets	pattern [+-]{0,1}[0-9]+ [+-]{0,1}0x[0-9a-fA-F]+

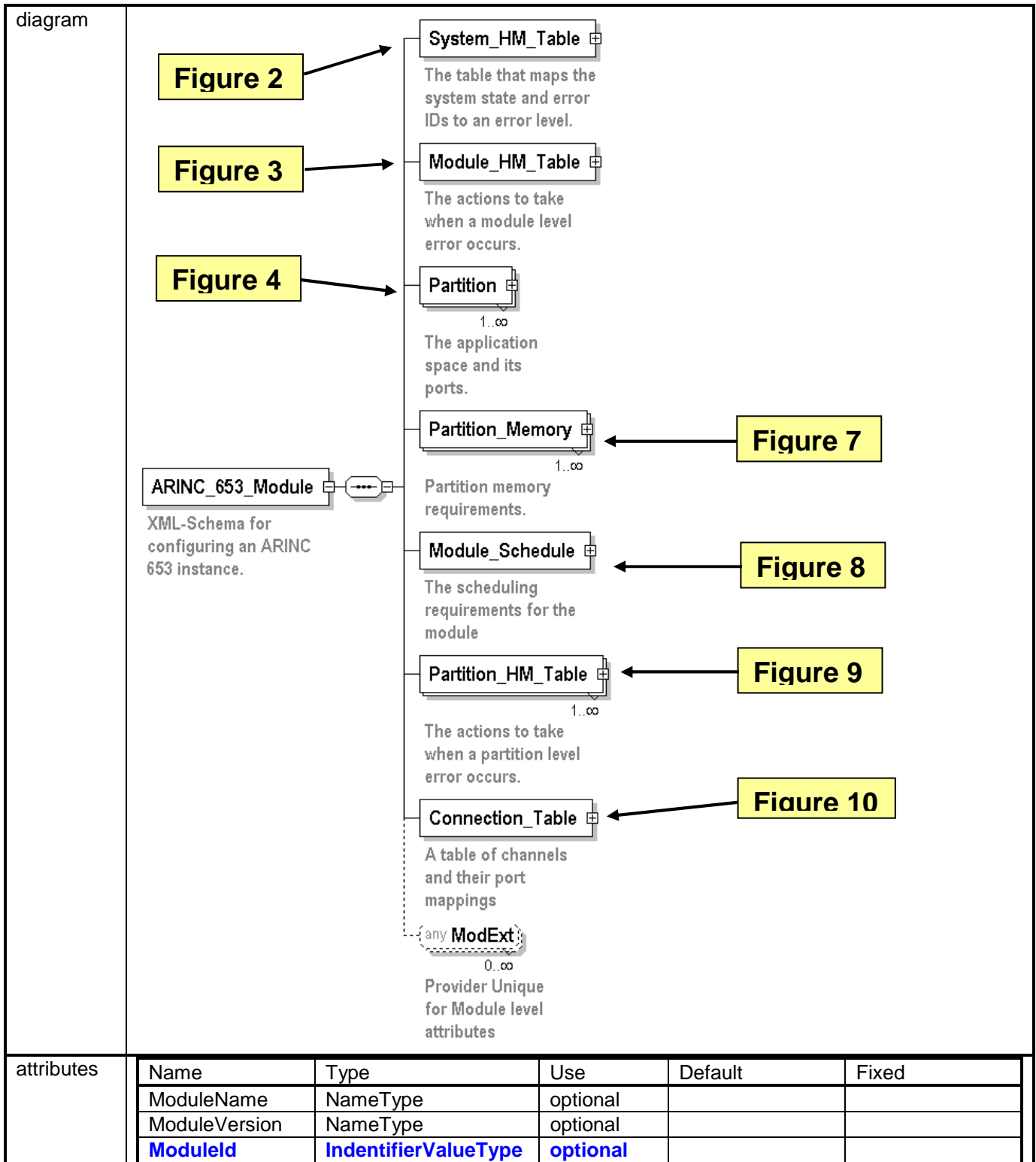
Table 3 DecOrHexValueType Definition

The IdentifierValueType allows identifiers to be either hex or integer values.

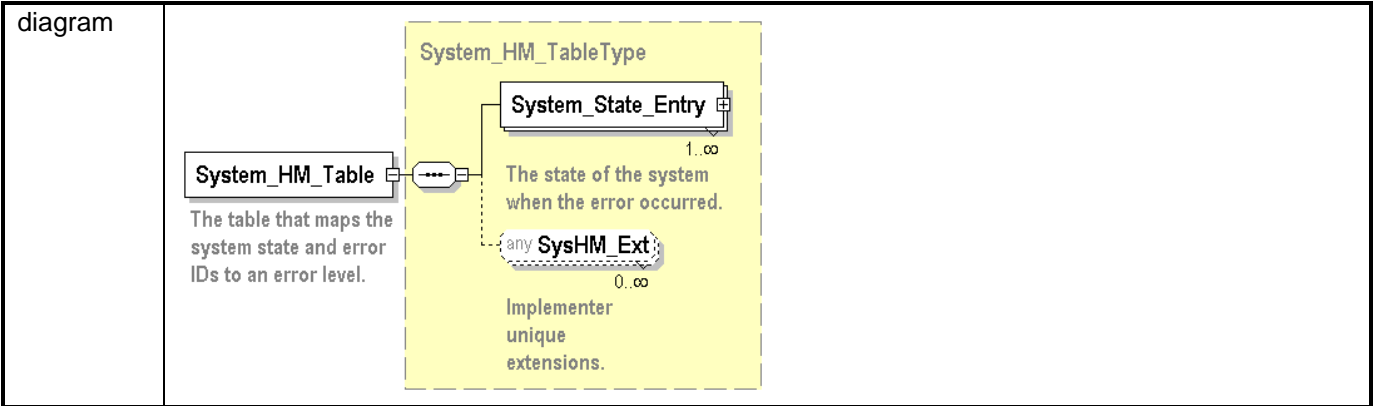
Simple Type	IdentifierValueType
type	DexOrHexValueType
facets	pattern [+-]{0,1}[0-9]+ [+-]{0,1}0x[0-9a-fA-F]+

Table 4 IdentifierValueType Definition

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

Element ARINC_653_Module**Figure 1 - ARINC_653_Module Definitions**

element ARINC_653_Module/System_HM_Table



element System_HM_TableType/System_State_Entry

diagram	<div><div>System_State_Entry</div><div>The state of the system when the error occurred.</div></div> <div><div>Error_ID_Level</div><div>1..∞</div><div>The mapping of error IDs to the module, partition or process level.</div></div>				
attributes	Name	Type	Use	Default	Fixed
	SystemState	xs:integer IdentifierValue Type	required		
	Description	NameType	optional		

element System_HM_TableType/System_State_Entry/Error_ID_Level

diagram	Error_ID_Level				
attributes	Name	Type	Use	Default	Fixed
	ErrorIdentifier	IdentifierValue Type xs:integer	required		
	Description	NameType	optional		
	ErrorLevel	ErrorLevelType	required		
	ErrorCode	ErrorCodeType	optional		

simpleType ErrorLevelType

type	restriction of xs:string	
facets	enumeration	MODULE
	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_FAILURE

Figure 2 - System_HM_Table Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

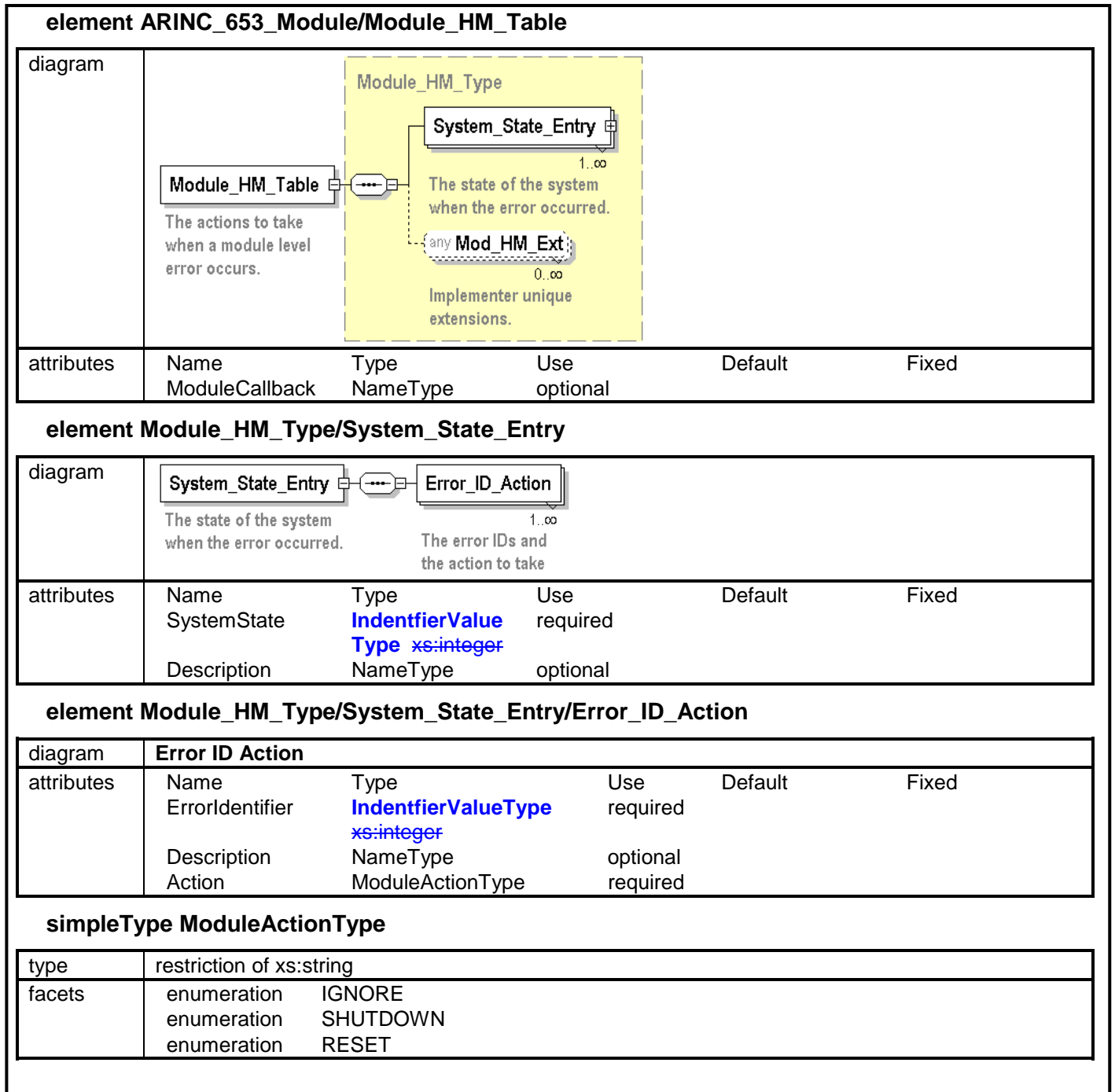


Figure 3 – Module_HM Table Definitions

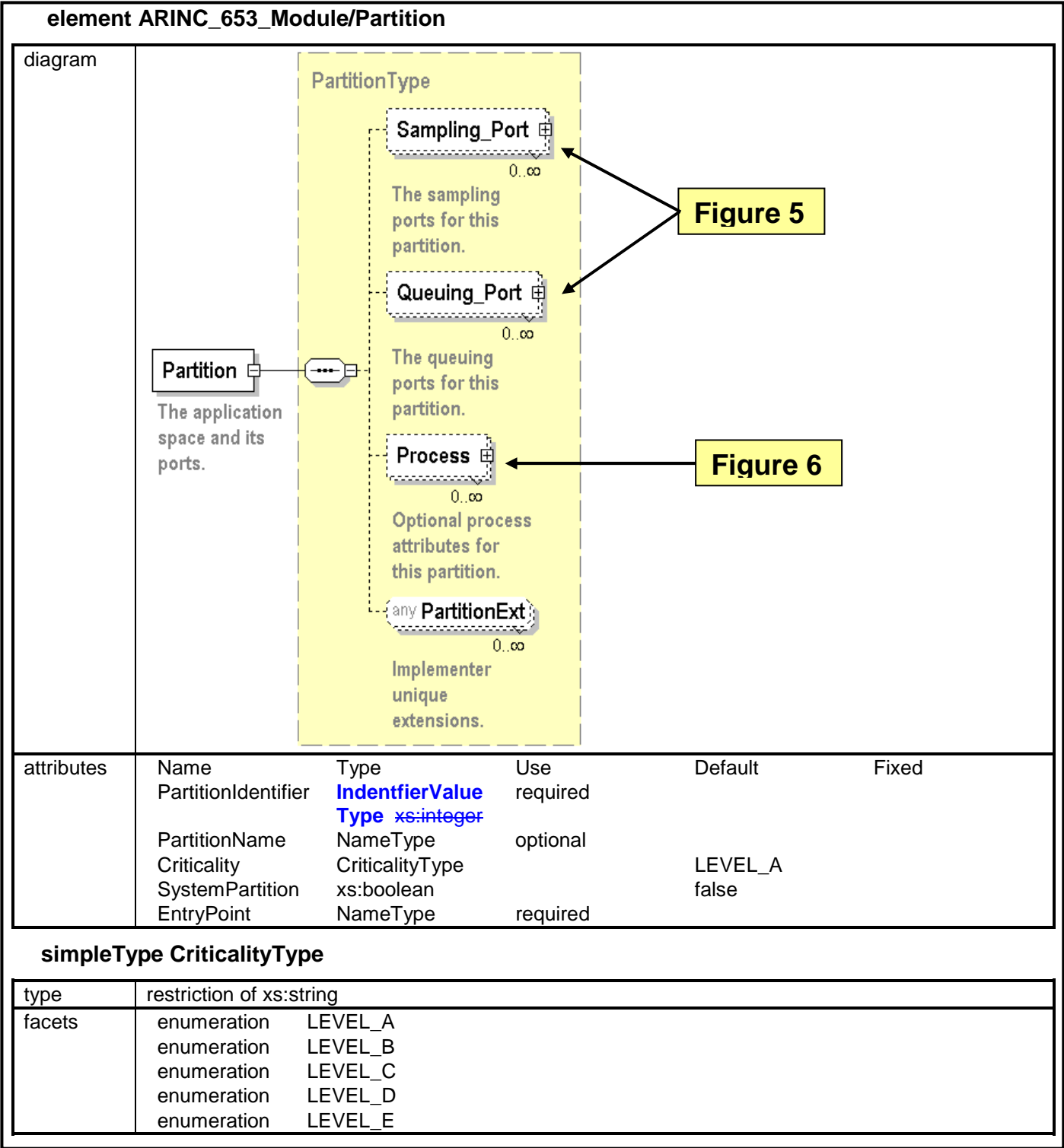


Figure 4 - Partition Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

element PartitionType/Sampling_Port					
diagram					
type	<i>SamplingPortType</i>				
attributes	Name	Type	Use	Default	Fixed
	PortName	NameType	required		
	MaxMessageSize	xs:hexBinary	required		
	Direction	DirectionType	required		
	RefreshRateSeconds	xs:float	required		
element PartitionType/Queuing_Port					
diagram					
type	<i>QueuingPortType</i>				
attributes	Name	Type	Use	Default	Fixed
	PortName	NameType	required		
	MaxMessageSize	xs:hexBinary	required		
	Direction	DirectionType	required		
	MaxNbMessages	xs:integer	required		
simpleType DirectionType					
type	restriction of xs:string				
facets	enumeration	SOURCE			
	enumeration	DESTINATION			

Figure 5 – Queuing and Sampling Port Definitions

Note: RefreshRateSeconds is intentionally a float value (instead of SYSTEM_TIME_TYPE) to aid correct entry by users.

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

element PartitionType/Process					
diagram	<p>Optional process attributes for this partition.</p>				
type	<i>ProcessType</i>				
attributes	Name	Type	Use	Default	Fixed
	Name	NameType	optional		
	StackSize	xs:hexBinary DecOrHexValueType	optional		

Figure 6 – Process Definitions

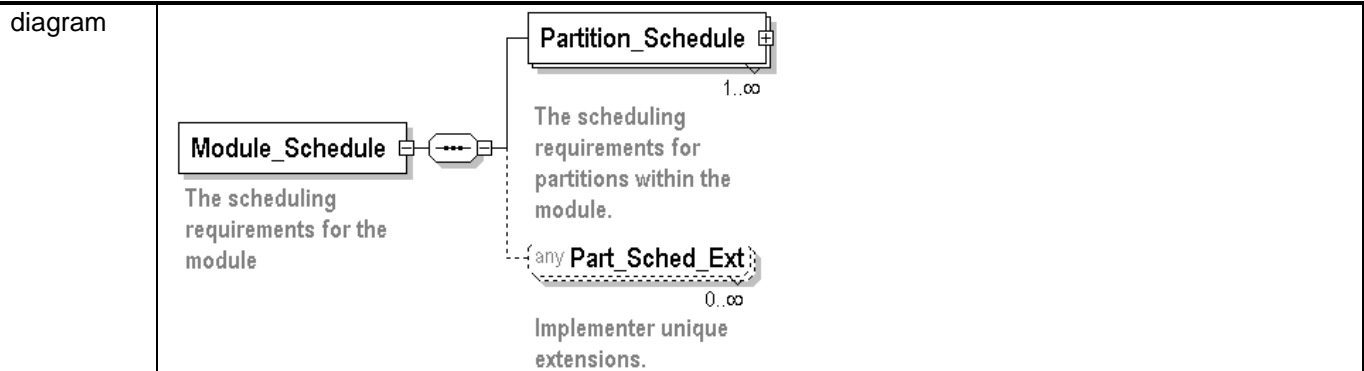
element ARINC_653_Module/Partition_Memory					
diagram	<p>Partition memory requirements.</p> <p>A single partition can have multiple mapping requirements. Defines memory bounds of the partition, with appropriate code/data segregation.</p>				
attributes	Name	Type	Use	Default	Fixed
	PartitionIdentifier	xs:integerIdentifier Value Type	required		
	PartitionName	NameType	optional		

element ARINC_653_Module/Partition_Memory/Memory_Requirements					
diagram					
attributes	Name	Type	Use	Default	Fixed
	RegionName	NameType	optional		
	Type	xs:string	required		
	SizeBytes	DecOrHexValueType xs:hexBinary	required		
	PhysicalAddress	DecOrHexValueType xs:hexBinary	optional		
	Access	xs:string	required		

Figure 7 – Partition Memory Definitions

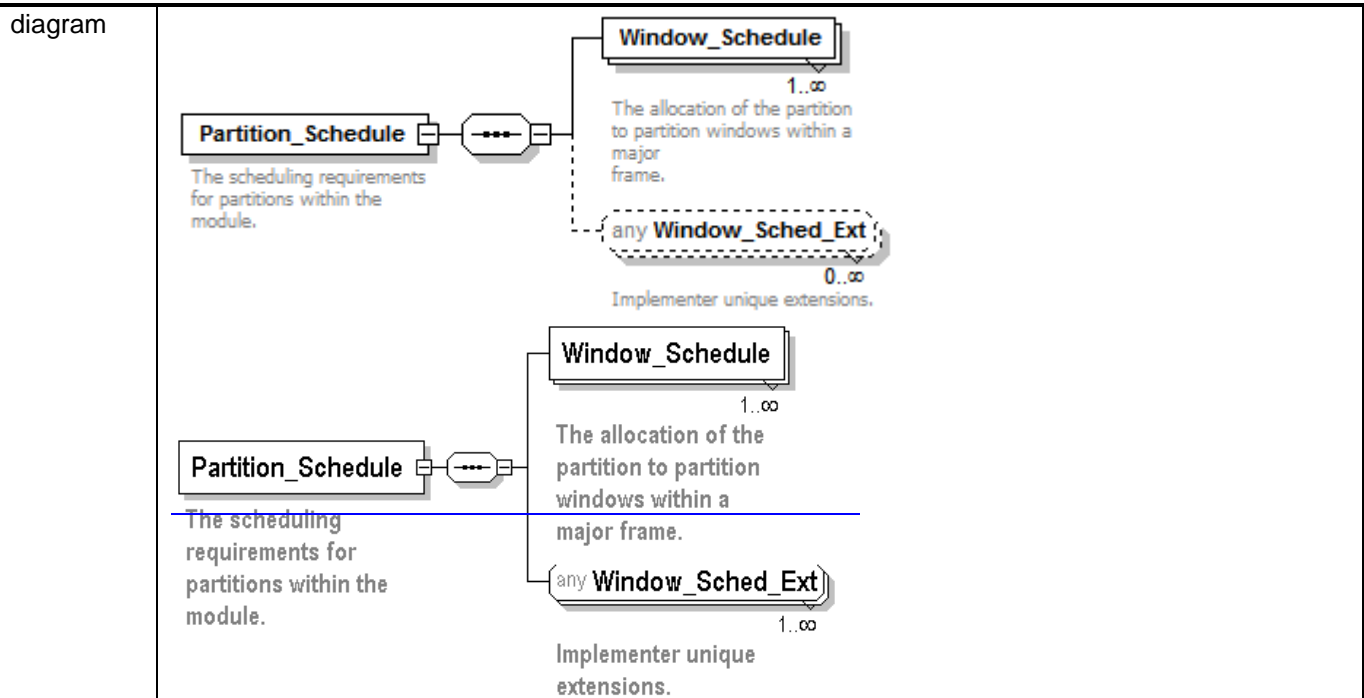
APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

element ARINC_653_Module/Partition_Schedule



attributes	Name	Type	Use	Default	Fixed
	MajorFrameSeconds	xs:float	required		

element ARINC_653_Module/Module_Schedule/Partition_Schedule



attributes	Name	Type	Use	Default	Fixed
	PartitionIdentifier	xs:integerIdentifierValueType	required		
	PartitionName	NameType	optional		
	PeriodSeconds	xs:float	required		
	PeriodDurationSeconds	xs:float	required		

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

element ARINC_653_Module/Module_Schedule/Partition_Schedule/Window_Schedule

diagram	<div>Window_Schedule</div> <p>The allocation of the partition to partition windows within a major frame.</p>				
attributes	Name	Type	Use	Default	Fixed
	WindowIdentifier	IdentifierValueTypes:positiveInteger	required		
	WindowStartSeconds	xs:float	required		
	WindowDurationSeconds	xs:float	required		
	PartitionPeriodStart	xs:boolean		false	

Figure 8 - Partition Schedule Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

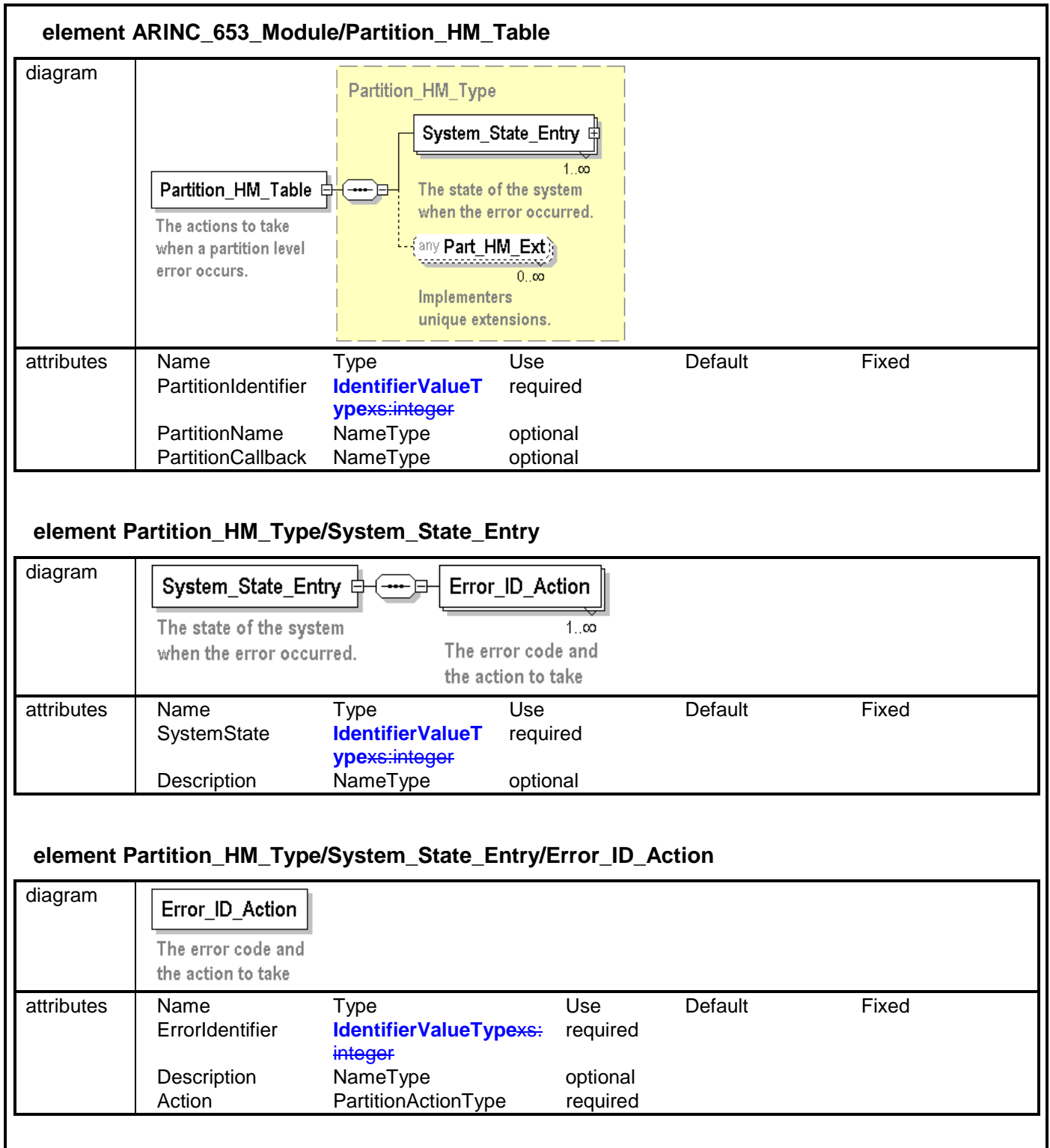


Figure 9 - Partition HM Table Definition

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

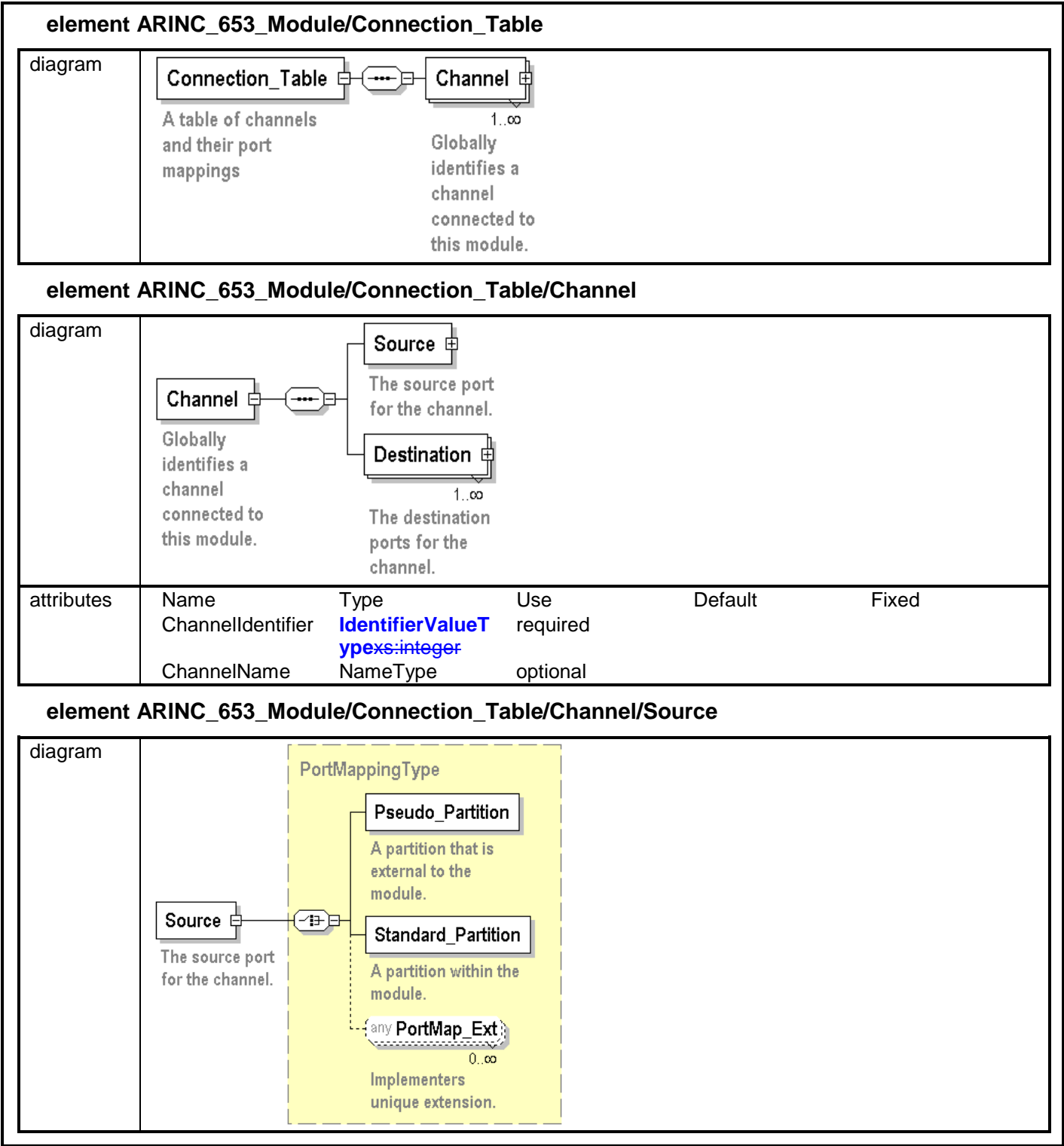


Figure 10-1 - Connection Table Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

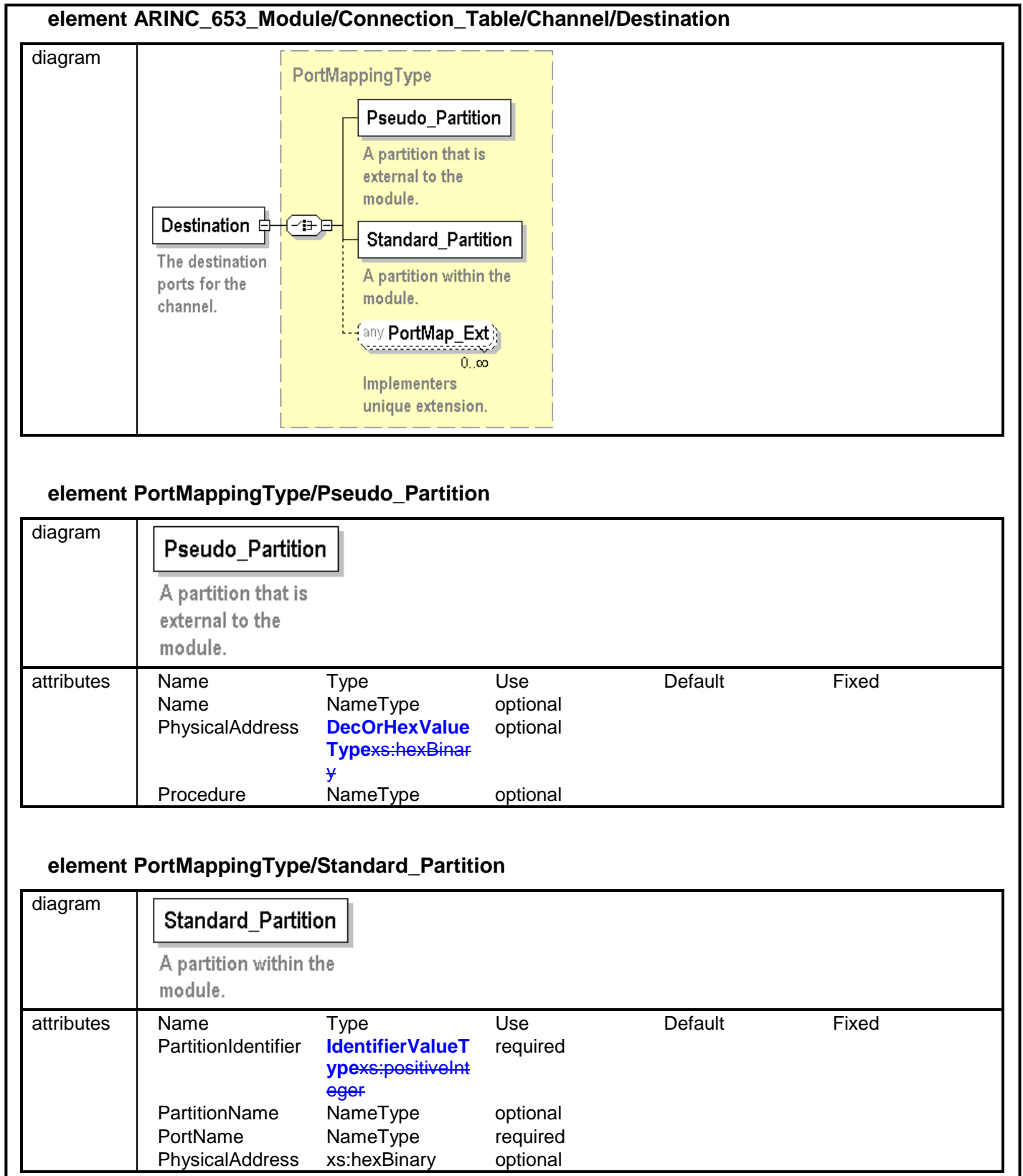


Figure 10-2 - Connection Table Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

Section G.2 HM Tables Mapping to Schema

The following graphics shows **examples of** how the different health monitoring (HM) tables are mapped to the ARINC 653 Schema and how these mappings would appear in an XML instance file. The 10 figures are briefly described in Table 1 below.

Table 1 - HM Table of Figures

Fig	Title	Notes
1	System HM Table	An example of a System HM table as defined by 2.5.2.3.
2	System HM Table Entries	The entries that define the module and partition level errors.
3	Module HM Table	An example of a Module HM table with its action entries.
4	Partition HM Table	An example of a Partition HM table with its action entries.
5	System HM Table Mapped to Schema	The System HM table entries mapped to XML-Schema.
6	System HM Table XML Example	An example XML instance of a System HM table.
7	Module Actions Mapped to Schema	The Module HM table entries mapped to the XML-Schema.
8	Module HM Table XML Example	An example XML instance of a Module HM table.
9	Partition Actions Mapped to Schema	The Partition HM table entries mapped to the XML-Schema.
10	Partition HM Table XML Example	An example XML instance of a Partition HM table.

The following pages contain the figures.

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

“The System HM table defines the level of an error (Module, Partition, Process) according to the detected error and the state of the system. For process level errors only, the error codes are indicated in this table. These error codes are implementation independent.” (ARINC 653 Section 2.5.2.3) – (Also, the state of the system and the error id are dependent on the implementation)

System States		System HM Table Mappings									
→	State Symbolic	ML = Module Level Error PL = Partition Level Error									
→	State Codes										
↓ Detected Error ↓		module init	Module Function	partition switching	partition init	partition process management	partition error handler	Process Execution	OS Execution		
Symbolic Name	Error ID	state 1	state 2	state 3	state 4	state 5	state 6	state 7	state 8	...	
configuration error	1	ML									
module config error	2	ML									
partition config error	3				PL						
partition init error	4				PL						
segmentation error	5	ML	ML	ML	PL	PL	PL	MEMORY_VIOLATION	PL		
time duration exceeded	6	ML	ML	ML	PL	PL	PL	DEADLINE_MISSED	PL		
invalid OS call	7	ML	ML	ML	PL	PL	PL	ILLEGAL_REQUEST			
divide by 0	8	ML	ML	ML	PL	PL	PL	NUMERIC_ERROR	PL		
overflow	9	ML	ML	ML	PL	PL	PL	NUMERIC_ERROR	PL		
floating point error	10	ML	ML	ML	PL	PL	PL	NUMERIC_ERROR	PL		
application reports error	11							APPLICATION_ERROR			
stack overflow	12	ML	ML	ML	PL	PL	PL	STACK_OVERFLOW	PL		
parity error	13	ML	ML	ML	PL	PL	PL	HARDWARE_FAULT	PL		
io access error	14	ML	ML	ML	PL	PL	PL	HARDWARE_FAULT	PL		
supervisor priv. violation	15				ML	PL	PL	MEMORY_VIOLATION	PL		
power interrupt	16	ML	ML	ML	ML	PL	PL	POWER_FAILURE	PL		
power failure	17	ML	ML	ML	ML	ML	ML	ML	ML		
...	...										

Figure 1 – System HM Table

APPENDIX G GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

The entries in the System HM Table tell the Core OS software where to look for the actions when an error occurs in a particular state. For entries that map to ML, the Module HM Table is referenced and for entries marked PL the Partition HM Table for the currently active partition is referenced. For the predefined errors the error handler process for the currently active partition is invoked with the predefined error codes.

System States		System HM Table Mappings									
⇒ State Symbolic		ML = Module Level Error									
⇒ State Codes		PL = Partition Level Error									
↓ Detected Error ↓		module init	Module Function	partition switching	partition init	partition process management	partition error handler	Process Execution		OS Execution	
Symbolic Name	Error ID	state 1	state 2	state 3	state 4	state 5	state 6	state 7	state 8	...	
configuration error	1	ML									
module config error	2	ML									
partition config error	3				PL						
partition init error	4				PL						
segmentation error	5	ML	ML	ML	PL	PL	PL	MEMORY VIOLATION		PL	
time duration exceeded	6	ML	ML	ML	PL	PL	PL	DEADLINE MISSED		PL	
invalid OS call	7	ML	ML	ML	PL	PL	PL	ILLEGAL_REQUEST			
divide by 0	8	ML	ML	ML	PL	PL	PL	NUMERIC_ERROR		PL	
overflow	9	ML	ML	ML	PL	PL	PL	NUMERIC_ERROR		PL	
floating point error	10	ML	ML	ML	PL	PL	PL	NUMERIC_ERROR		PL	
application reports error	11							APPLICATION_ERROR			
stack overflow	12	ML	ML	ML	PL	PL	PL	STACK_OVERFLOW		PL	
parity error	13	ML	ML	ML	PL	PL	PL	HARDWARE_FAULT		PL	
io access error	14	ML	ML	ML	PL	PL	PL	HARDWARE_FAULT		PL	
supervisor priv. violation	15				ML	PL	PL	MEMORY VIOLATION		PL	
power interrupt	16	ML	ML	ML	ML	PL		POWER_FAILURE		PL	
power failure	17	ML	ML	ML	ML	ML	ML	ML		ML	
...	...										

Partition HM Tables define the actions

Module HM Table defines the actions

Figure 2 – System HM Table Entries

The Module HM Table contains the actions when a module level error occurs. The module actions are reset, shutdown, or ignore. The module HM callback is also defined in this table.

System States		Module HM Table Mappings									
⇒ State Symbolic											
⇒ State Codes											
↓ Detected Error ↓		module init	Module Function	partition switching	partition init	partition process management	partition error handler	Process Execution		OS Execution	
Symbolic Name	Error ID	state 1	state 2	state 3	state 4	state 5	state 6	state 7	state 8	...	
configuration error	1	SH									
module config error	2	SH									
partition config error	3										
partition init error	4										
segmentation error	5	SH	SH	SH							
time duration exceeded	6	IG	IG	IG							
invalid OS call	7	IG	IG	IG							
divide by 0	8	IG	IG	IG							
overflow	9	IG	IG	IG							
floating point error	10	IG	IG	IG							
application reports error	11										
stack overflow	12	RS	RS	RS							
parity error	13	RS	RS	RS							
io access error	14	IG	IG	IG							
supervisor priv. violation	15				RS						
power interrupt	16	IG	IG	IG	IG						
power failure	17	RS	RS	RS	RS	RS	RS	RS	RS	RS	
...	...										
Module HM Callback	callback name / addr										

Module Actions
RS = RESET
SH = SHUTDOWN
IG = IGNORE

Figure 3 – Module HM Table

APPENDIX G

GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

The Partition HM Table contains the actions when a partition level error occurs. The possible partition actions are shown below. There is a unique partition HM table for each partition. The process level errors need an action entry for two cases: no process level error handler or failure in the process level error handler.

System States		Partition 1 HM Table Mappings									
⇒	State Symbolic										
⇒	State Codes										
↓ Detected Error ↓		module init	system function execution	partition switching	partition init	partition process management	partition error handler	Process Execution	OS Execution		
Symbolic Name	Error ID	state 1	state 2	state 3	state 4	state 5	state 6	state 7	state 8		
configuration error	1										
module config error	2										
partition config error	3				ID						
partition init error	4				CS						
segmentation error	5				CS	ID	ID	ID	ID		
time duration exceeded	6				IG	IG	IG	WS	IG		
invalid OS call	7				IG	WS	ID	IG	IG		
divide by 0	8				IG	IG	WS	IG	ID		
overflow	9				IG	IG	WS	ID	ID		
floating point error	10				IG	IG	WS	ID	ID		
application reports error	11							IG	IG		
stack overflow	12				CS	ID	ID	ID	ID		
parity error	13				CS	IG	IG	ID	ID		
io access error	14				CS	IG	IG	ID	ID		
supervisor priv. violation	15					ID	ID	ID	ID		
power interrupt	16					WS	WS	ID	ID		
power failure	17										
...	...										
Partition 1 HM Callback		callback name									

Figure 4 – Partition HM Table

System States		System HM Table Mappings									
⇒	State Symbolic										
⇒	State Codes										
↓ Detected Error ↓		Module init	Module Function	partition switching	partition init	partition process management	partition error handler	Process Level Errors Predefined			
Symbolic Name	Error ID	state 1	state 2	state 3	state 4	state 5	state 6	state 7	state 8		
configuration error	1	ML									
module config error	2	ML									
partition config error	3										
partition init error	4				PL						
segmentation error	5	ML			PL						
time duration exceeded	6	ML	ML	ML	PL	PL	PL	MEMORY VIOLATION	PL		
invalid OS call	7	ML	ML	ML	PL	PL	PL	DEADLINE MISSED	PL		
divide by 0	8	ML	ML	ML	PL	PL	PL	ILLEGAL REQUEST	PL		
overflow	9	ML						NUMERIC ERROR	PL		
floating point error	10	ML									
application reports error	11										
stack overflow	12	ML									
parity error	13	ML									
io access error	14	ML									
supervisor priv. violation	15										
power interrupt	16	ML									
power failure	17	ML									
...	...										

System HM Table Mappings

ML = Module Level Error
PL = Partition Level Error

System HM TableType has the required attribute **SystemState**. The system states and

System_State_Entry has the required attribute **SystemState**. The state of the system when the error occurred.

Error_ID_Level has the required attributes **ErrorIdentifier**, and **ErrorLevel**. For process level errors the additional predefined **ErrorCode** is required. Figure 6 is an example of the System HM Table XML.

Figure 5 – System HM Table Mapped to Schema

Figure 6 – System HM Table XML Example



APPENDIX G

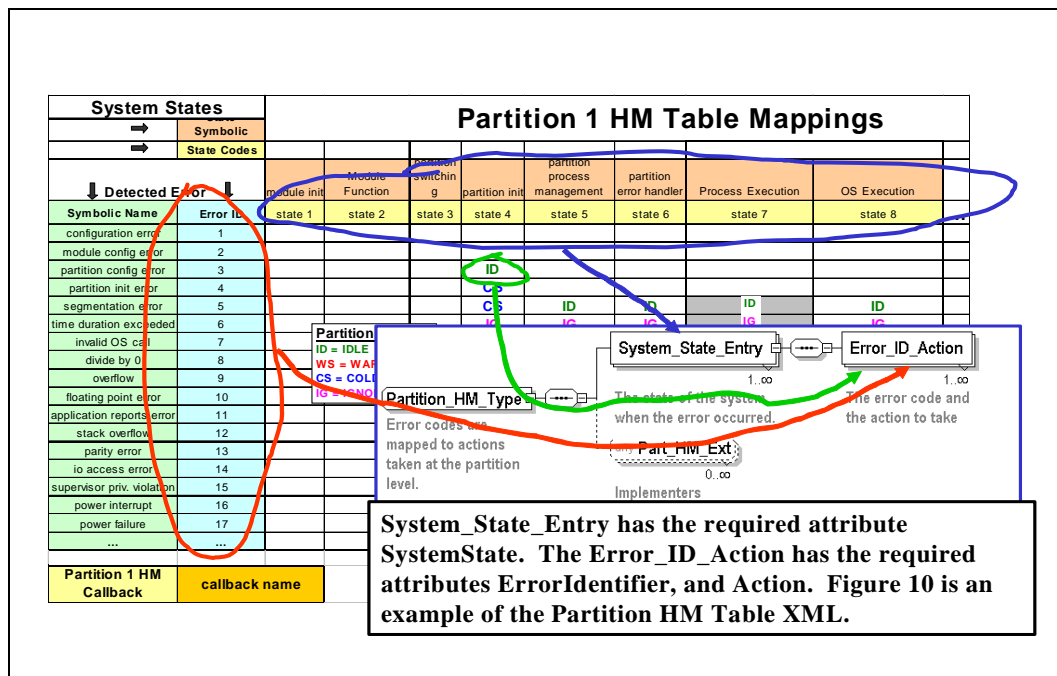
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

```

<Module_HM_Table ModuleCallback="module_HM_callback">
  <System_State_Entry SystemState="1" Description="module init">
    <Error_ID_Action ErrorIdentifier="1" Action="SHUTDOWN"/>
    <Error_ID_Action ErrorIdentifier="2" Action="SHUTDOWN"/>
    <Error_ID_Action ErrorIdentifier="5" Action="SHUTDOWN"/>
    <Error_ID_Action ErrorIdentifier="6" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Action="IGNORE"/>
  </System_State_Entry>
  <System_State_Entry SystemState="2" Description="system function execution">
    <Error_ID_Action ErrorIdentifier="5" Action="SHUTDOWN"/>
    <Error_ID_Action ErrorIdentifier="6" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="8" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="9" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="10" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="12" Action="RESET"/>
  </System_State_Entry>
</Module_HM_Table>

```

Figure 8 – Module HM Table XML Example



APPENDIX G

GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA

```

<Partition_HM_Table PartitionIdentifier="2" PartitionName="flight controls" PartitionCallback="partition_HM_callback">
  <System_State_Entry SystemState="4" Description="partition init">
    <Error_ID_Action ErrorIdentifier="3" Description="partition config error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="4" Description="partition init error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IGNORE"/>
  </System_State_Entry>
  <System_State_Entry SystemState="6" Description="partition error handler">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="WARM_START"/>
  </System_State_Entry>
  <System_State_Entry SystemState="7" Description="process execution">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="IDLE"/>
  </System_State_Entry>
</Partition_HM_Table>

```

Figure 10 – Partition HM Table XML Example

APPENDIX H

ARINC 653 XML-SCHEMA

This Appendix was added in Supplement 1 to include the actual XML-Schema code file.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="ARINC_653_Module">
    <xs:annotation>
      <xs:documentation>XML-Schema for configuring an ARINC 653 instance.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="System_HM_Table" type="System_HM_TableType">
          <xs:annotation>
            <xs:documentation>The table that maps the system state and error IDs to an error level.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="Module_HM_Table" type="Module_HM_Type">
          <xs:annotation>
            <xs:documentation>The actions to take when a module level error occurs.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="Partition" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>The application space and its ports.</xs:documentation>
            <xs:documentation>Where the Applications resides. Uses ARINC 653 API</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="PartitionType"/>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="Partition_Memory" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>Partition memory requirements.</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Memory_Requirements" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>A single partition can have multiple mapping requirements. Defines memory
bounds of the
partition, with appropriate code/data segregation.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:attribute name="RegionName" type="NameType" use="optional"/>
                  <xs:attribute name="Type" type="xs:string" use="required"/>
                  <xs:attribute name="SizeBytes" type="DecOrHexValueType" use="required"/>
                  <xs:attribute name="PhysicalAddress" type="DecOrHexValueType" use="optional"/>
                  <xs:attribute name="Access" type="xs:string" use="required"/>
                </xs:complexType>
              </xs:element>
              <xs:any namespace="Memory_Ext" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Implementer unique extensions.</xs:documentation>
                </xs:annotation>
              </xs:any>
            </xs:sequence>
            <xs:attribute name="PartitionIdentifier" type="IdentifierValueType" use="required"/>
            <xs:attribute name="PartitionName" type="NameType" use="optional"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="Module_Schedule">
          <xs:annotation>

```

APPENDIX H

ARINC 653 XML-SCHEMA

```

    <xs:documentation>The scheduling requirements for the module</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Partition_Schedule" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>The scheduling requirements for partitions within the
module.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Window_Schedule" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>The allocation of the partition to partition windows within a major
frame.</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:sequence>
                  <xs:annotation>
                    <xs:complexType>
                      <xs:attribute name="WindowIdentifier" type="DecOrHexValueType" use="required"/>
                      <xs:attribute name="WindowStartSeconds" type="xs:float" use="required"/>
                      <xs:attribute name="WindowDurationSeconds" type="xs:float" use="required"/>
                      <xs:attribute name="PartitionPeriodStart" type="xs:boolean" default="false"/>
                    </xs:complexType>
                  </xs:sequence>
                  <xs:element>
                    <xs:any namespace="Window_Sched_Ext" minOccurs="0" maxOccurs="unbounded">
                      <xs:annotation>
                        <xs:documentation>Implementer unique extensions.</xs:documentation>
                      </xs:annotation>
                    </xs:any>
                  </xs:sequence>
                  <xs:attribute name="PartitionIdentifier" type="IdentifierValueType" use="required"/>
                  <xs:attribute name="PartitionName" type="NameType" use="optional"/>
                  <xs:attribute name="PeriodSeconds" type="xs:float" use="required"/>
                  <xs:attribute name="PeriodDurationSeconds" type="xs:float" use="required"/>
                </xs:complexType>
              </xs:element>
            <xs:any namespace="Part_Sched_Ext" minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Implementer unique extensions.</xs:documentation>
              </xs:annotation>
            </xs:any>
          </xs:sequence>
          <xs:attribute name="MajorFrameSeconds" type="xs:float" use="required"/>
        </xs:complexType>
      </xs:element>
    <xs:element name="Partition_HM_Table" type="Partition_HM_Type" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The actions to take when a partition level error occurs.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Connection_Table">
      <xs:annotation>
        <xs:documentation>A table of channels and their port mappings</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Channel" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>Globally identifies a channel connected to this module.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Source" type="PortMappingType">
                  <xs:annotation>
                    <xs:documentation>The source port for the channel. </xs:documentation>
                  </xs:annotation>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:element>

```

APPENDIX H

ARINC 653 XML-SCHEMA

```

        </xs:annotation>
      </xs:element>
      <xs:element name="Destination" type="PortMappingType" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>The destination ports for the channel. </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="ChannelIdentifier" type="IdentifierValueType" use="required"/>
    <xs:attribute name="ChannelName" type="NameType" use="optional"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:any namespace="ModExt" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>Provider Unique for Module level attributes</xs:documentation>
  </xs:annotation>
</xs:any>
</xs:sequence>
<xs:attribute name="ModuleName" type="NameType" use="optional"/>
<xs:attribute name="ModuleVersion" type="NameType" use="optional"/>
<xs:attribute name="ModuleId" type="IdentifierValueType" use="optional"/>
</xs:complexType>
</xs:element>
<xs:complexType name="PortType">
  <xs:annotation>
    <xs:documentation>The base port attributes of both sampling and queuing ports</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:any namespace="PortExt" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Implementer unique extensions.</xs:documentation>
      </xs:annotation>
    </xs:any>
  </xs:sequence>
  <xs:attribute name="Name" type="NameType" use="required"/>
  <xs:attribute name="MaxMessageSize" type="DecOrHexValueType" use="required"/>
  <xs:attribute name="Direction" type="DirectionType" use="required"/>
</xs:complexType>
<xs:complexType name="SamplingPortType">
  <xs:annotation>
    <xs:documentation>The configurable attributes of the sampling port.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="PortType">
      <xs:attribute name="RefreshRateSeconds" type="xs:float" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="QueuingPortType">
  <xs:annotation>
    <xs:documentation>The configurable attributes of queuing ports.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="PortType">
      <xs:attribute name="MaxNbMessages" type="xs:integer" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ProcessType">
  <xs:annotation>
    <xs:documentation>Optional process configuration parameters</xs:documentation>
  </xs:annotation>

```

APPENDIX H

ARINC 653 XML-SCHEMA

```

</xs:annotation>
<xs:sequence>
  <xs:any namespace="Proc_Ext" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
      <xs:documentation>Implementer Unique</xs:documentation>
    </xs:annotation>
  </xs:any>
</xs:sequence>
<xs:attribute name="Name" type="NameType" use="optional"/>
<xs:attribute name="StackSize" type="DecOrHexValueType" use="optional"/>
</xs:complexType>
<xs:complexType name="PartitionType">
  <xs:annotation>
    <xs:documentation>The system and application partition type definition.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Sampling_Port" type="SamplingPortType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The sampling ports for this partition.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Queuing_Port" type="QueuingPortType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The queuing ports for this partition.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="Process" type="ProcessType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Optional process attributes for this partition.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:any namespace="PartitionExt" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Implementer unique extensions.</xs:documentation>
      </xs:annotation>
    </xs:any>
  </xs:sequence>
  <xs:attribute name="PartitionIdentifier" type="IdentifierValueType" use="required"/>
  <xs:attribute name="PartitionName" type="NameType" use="optional"/>
  <xs:attribute name="Criticality" type="CriticalityType" default="LEVEL_A"/>
  <xs:attribute name="SystemPartition" type="xs:boolean" default="false"/>
  <xs:attribute name="EntryPoint" type="NameType" use="required"/>
</xs:complexType>
<xs:complexType name="Partition_HM_Type">
  <xs:annotation>
    <xs:documentation>Error codes are mapped to actions taken at the partition level.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="System_State_Entry" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The state of the system when the error occurred.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Error_ID_Action" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>The error code and the action to take</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attribute name="ErrorIdentifier" type="IdentifierValueType" use="required"/>
              <xs:attribute name="Description" type="NameType" use="optional"/>
              <xs:attribute name="Action" type="PartitionActionType" use="required"/>
              <xs:anyAttribute namespace="##any"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>

```


APPENDIX H

ARINC 653 XML-SCHEMA

```

        </xs:element>
    </xs:sequence>
    <xs:attribute name="SystemState" type="IdentifierValueType" use="required"/>
    <xs:attribute name="Description" type="NameType" use="optional"/>
</xs:complexType>
</xs:element>
<xs:any namespace="Part_HM_Ext" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
        <xs:documentation>Implementers unique extensions.</xs:documentation>
    </xs:annotation>
</xs:any>
</xs:sequence>
<xs:attribute name="PartitionIdentifier" type="IdentifierValueType" use="required"/>
<xs:attribute name="PartitionName" type="NameType" use="optional"/>
<xs:attribute name="PartitionCallback" type="NameType" use="optional"/>
</xs:complexType>
<xs:complexType name="Module_HM_Type">
    <xs:annotation>
        <xs:documentation>System state and error codes are mapped to actions taken at the module level.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="System_State_Entry" maxOccurs="unbounded">
            <xs:annotation>
                <xs:documentation>The state of the system when the error occurred.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Error_ID_Action" maxOccurs="unbounded">
                        <xs:annotation>
                            <xs:documentation>The error IDs and the action to take</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:attribute name="ErrorIdentifier" type="xs:integer" use="required"/>
                            <xs:attribute name="Description" type="NameType" use="optional"/>
                            <xs:attribute name="Action" type="ModuleActionType" use="required"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="SystemState" type="IdentifierValueType" use="required"/>
                <xs:attribute name="Description" type="NameType" use="optional"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="ModuleCallback" type="NameType" use="optional"/>
</xs:complexType>
<xs:simpleType name="DecOrHexValueType">
    <xs:annotation>
        <xs:documentation>Allows hex and decimal numbers. Hex start 0x</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
        <xs:pattern value="[+-]{0,1}[0-9]+[+-]{0,1}0x[0-9a-fA-F]+" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="IdentifierValueType">
    <xs:annotation>
        <xs:documentation>Restricts identifiers to hex or decimal numbers.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="DecOrHexValueType"/>
</xs:simpleType>

```

APPENDIX H

ARINC 653 XML-SCHEMA

```

<xs:simpleType name="NameType">
  <xs:annotation>
    <xs:documentation>A 1..30 character string.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
    <xs:maxLength value="30"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CriticalityType">
  <xs:annotation>
    <xs:documentation>Level A ... Level E</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="LEVEL_A"/>
    <xs:enumeration value="LEVEL_B"/>
    <xs:enumeration value="LEVEL_C"/>
    <xs:enumeration value="LEVEL_D"/>
    <xs:enumeration value="LEVEL_E"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DirectionType">
  <xs:annotation>
    <xs:documentation>SOURCE or DESTINATION</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="SOURCE"/>
    <xs:enumeration value="DESTINATION"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="System_HM_TableType">
  <xs:annotation>
    <xs:documentation>The system states and error codes are mapped to the Module, Partition or Process
level.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="System_State_Entry" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The state of the system when the error occurred.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Error_ID_Level" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>The mapping of error IDs to the module, partition or process
level.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attribute name="ErrorIdentifier" type="IdentifierValueType" use="required"/>
              <xs:attribute name="Description" type="NameType" use="optional"/>
              <xs:attribute name="ErrorLevel" type="ErrorLevelType" use="required"/>
              <xs:attribute name="ErrorCode" type="ErrorCodeType" use="optional"/>
            </xs:complexType>
          </xs:element>
          <xs:sequence>
            <xs:attribute name="SystemState" type="IdentifierValueType" use="required"/>
            <xs:attribute name="Description" type="NameType" use="optional"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    <xs:any namespace="SysHM_Ext" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Implementer unique extensions.</xs:documentation>
      </xs:annotation>
    </xs:any>
  </xs:sequence>
</xs:complexType>

```

APPENDIX H

ARINC 653 XML-SCHEMA

```

    </xs:sequence>
</xs:complexType>
<xs:simpleType name="ErrorLevelType">
  <xs:annotation>
    <xs:documentation>MODULE, PARTITION, or PROCESS level error.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="MODULE"/>
    <xs:enumeration value="PARTITION"/>
    <xs:enumeration value="PROCESS"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ErrorCodeType">
  <xs:annotation>
    <xs:documentation>The predefined ARINC 653 process errors.</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_FAILURE"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ModuleActionType">
  <xs:annotation>
    <xs:documentation>The 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:restriction>
</xs:simpleType>
<xs:simpleType name="PartitionActionType">
  <xs:annotation>
    <xs:documentation>The actions to take when partition level errors occur.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="IGNORE"/>
    <xs:enumeration value="IDLE"/>
    <xs:enumeration value="WARM_START"/>
    <xs:enumeration value="COLD_START"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="PortMappingType">
  <xs:annotation>
    <xs:documentation> The port communication mapping.</xs:documentation>
  </xs:annotation>
  <xs:choice>
    <xs:element name="Pseudo_Partition">
      <xs:annotation>
        <xs:documentation>A partition that is external to the module. </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:attribute name="Name" type="NameType" use="optional"/>
        <xs:attribute name="PhysicalAddress" type="DecOrHexValueType" use="optional"/>
        <xs:attribute name="Procedure" type="NameType" use="optional"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Standard_Partition">

```

APPENDIX H

ARINC 653 XML-SCHEMA

```

<xs:annotation>
  <xs:documentation>A partition within the module.</xs:documentation>
</xs:annotation>
<xs:complexType>
  <xs:attribute name="PartitionIdentifier" type="IdentifierValueType" use="required"/>
  <xs:attribute name="PartitionName" type="NameType" use="optional"/>
  <xs:attribute name="PortName" type="NameType" use="required"/>
  <xs:attribute name="PhysicalAddress" type="DecOrHexValueType" use="optional"/>
</xs:complexType>
</xs:element>
<xs:any namespace="PortMap_Ext" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>Implementers unique extension.</xs:documentation>
  </xs:annotation>
</xs:any>
</xs:choice>
</xs:complexType>
</xs:schema>

```

APPENDIX I EXAMPLE ARINC 653 XML INSTANCE FILE

This Appendix was added in Supplement 1 to include examples of ARINC 653 XML instance files.

The instance file shown in this Appendix is built from the example shown in Figure 1 and 2. The instance file also contains the example HM table mappings shown in Appendix G.

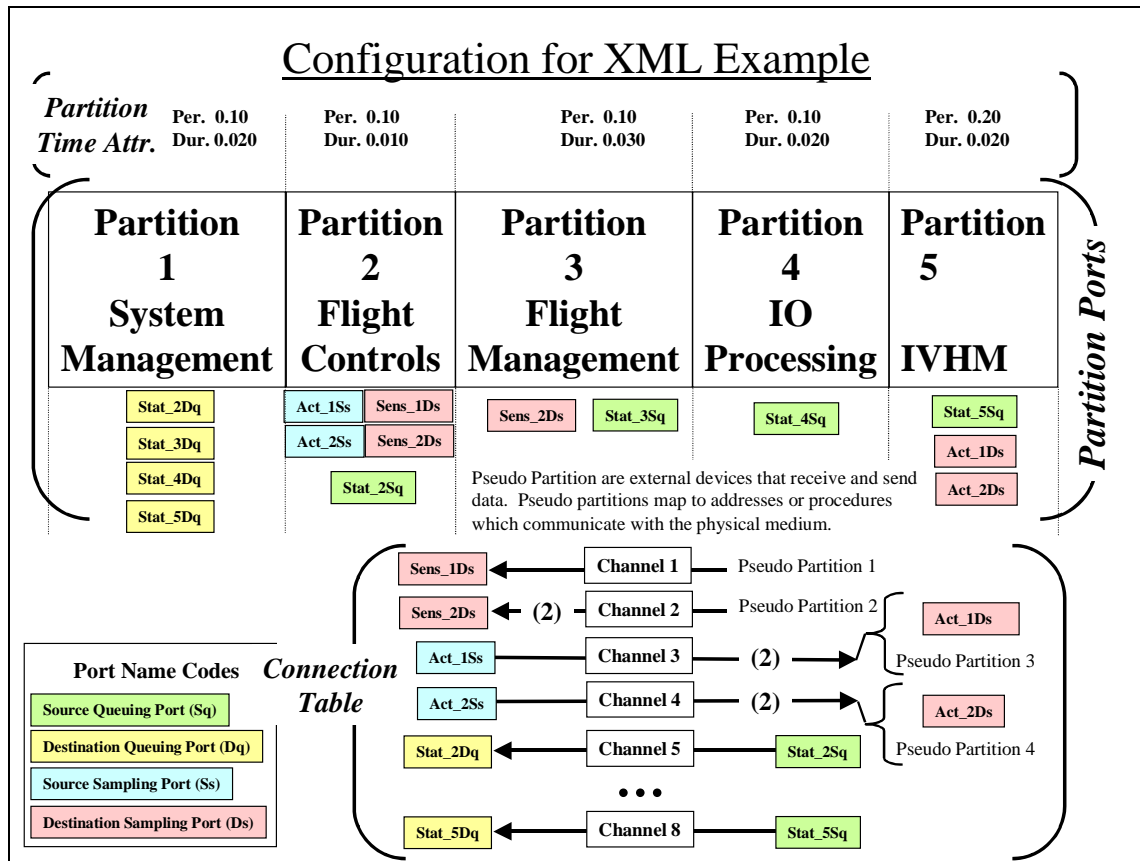


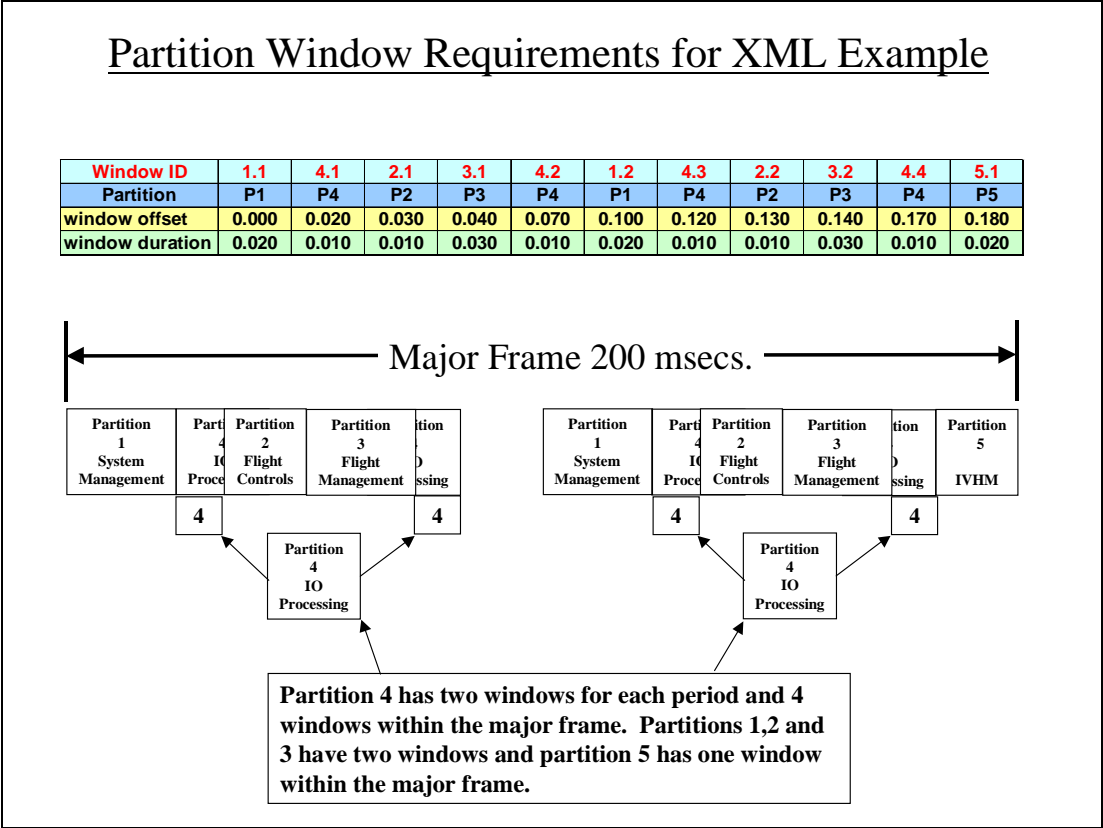
Figure 1 - Configuration for XML Example

Figure 1 shows a five-partition module. The time attributes of the module and the sampling and queuing ports of the module are shown in the figure and completely defined in the XML instance.

The partition's ports are shown in the connection table with their channel identifiers. The Pseudo Partitions, which are sources and destinations for Channel 1, 2, 3 and 4, are external IO ports and are mapped in the XML instance file as either addresses or procedure (driver) attributes. All the other connections are within the **partition module**. There are 8 queuing ports and 4-7 sampling ports in the XML configuration instance example.

The allocation of these five partitions to partition windows is shown in Figure 2. Since the lowest frequency partition defines the major frame rate the major frame rate for this module example would be 200 milliseconds. The window requirements would define the start time (offset) of each window and its duration for the entire major frame.

APPENDIX I
EXAMPLE ARINC 653 XML INSTANCE FILE



• Figure 2 - Partition Window Time Requirements

The XML Instance example for these figures follows.

APPENDIX I

EXAMPLE ARINC 653 XML INSTANCE FILE

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) -->
<!-- Sample XML file generated by XML Spy v4.4 U (http://www.xmlspy.com)-->
<ARINC_653_Module xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".\\A653_Part1_revA.xsd" Clearwater-Version-1.1 .xsd ModuleName="Example ARINC 653
XML Instance" ModuleVersion="18-Mar-2003">
  <System_HM_Table>
    <System_State_Entry SystemState="1" Description="Module Init">
      <Error_ID_Level ErrorIdentifier="1" Description="Configuration Error" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="2" Description="Module Config Error" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="5" Description="segmentation error" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="6" Description="time duration exceeded" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="7" Description="invalid OS call" ErrorLevel="MODULE"/>
    </System_State_Entry>
    <System_State_Entry SystemState="2" Description="System Function Execution">
      <Error_ID_Level ErrorIdentifier="5" Description="Segmentation Violation" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="6" Description="time duration exceeded" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="7" Description="invalid OS call" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="8" Description="divide by 0" ErrorLevel="MODULE"/>
      <Error_ID_Level ErrorIdentifier="9" Description="floating point error" ErrorLevel="MODULE"/>
    </System_State_Entry>
    <System_State_Entry SystemState="4" Description="Partition Init">
      <Error_ID_Level ErrorIdentifier="3" Description="partition config error" ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="4" Description="partition init error" ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="5" Description="segmentation violation" ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="6" Description="time duration exceeded" ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="7" Description="invalid OS call" ErrorLevel="PARTITION"/>
    </System_State_Entry>
    <System_State_Entry SystemState="7" Description="Process Execution">
      <Error_ID_Level ErrorIdentifier="5" Description="segmentation error" ErrorLevel="PROCESS"
      ErrorCode="MEMORY_VIOLATION"/>
      <Error_ID_Level ErrorIdentifier="6" Description="time duration exceeded" ErrorLevel="PROCESS"
      ErrorCode="DEADLINE_MISSED"/>
      <Error_ID_Level ErrorIdentifier="7" Description="invalid OS call" ErrorLevel="PROCESS"
      ErrorCode="ILLEGAL_REQUEST"/>
      <Error_ID_Level ErrorIdentifier="8" Description="divide by 0" ErrorLevel="PROCESS"
      ErrorCode="NUMERIC_ERROR"/>
      <Error_ID_Level ErrorIdentifier="9" Description="overflow" ErrorLevel="PROCESS"
      ErrorCode="NUMERIC_ERROR"/>
    </System_State_Entry>
  </System_HM_Table>
  <Module_HM_Table ModuleCallback="module_HM_callback">
    <System_State_Entry SystemState="1" Description="module init">
      <Error_ID_Action ErrorIdentifier="1" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="2" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="5" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="6" Action="IGNORE"/>
      <Error_ID_Action ErrorIdentifier="7" Action="IGNORE"/>
    </System_State_Entry>
    <System_State_Entry SystemState="2" Description="system function execution">
      <Error_ID_Action ErrorIdentifier="5" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="6" Action="IGNORE"/>
      <Error_ID_Action ErrorIdentifier="7" Action="IGNORE"/>
      <Error_ID_Action ErrorIdentifier="8" Action="IGNORE"/>
      <Error_ID_Action ErrorIdentifier="9" Action="IGNORE"/>
      <Error_ID_Action ErrorIdentifier="10" Action="IGNORE"/>
      <Error_ID_Action ErrorIdentifier="12" Action="RESET"/>
    </System_State_Entry>
  </Module_HM_Table>
  <Partition PartitionIdentifier="1" PartitionName="system management" Criticality="LEVEL_A" SystemPartition="true"
  EntryPoint="Initial">
    <Queueing_Port PortName="Stat_2Dq" MaxMessageSize="30" Direction="DESTINATION" MaxNbMessages="30"/>
    <Queueing_Port PortName="Stat_3Dq" MaxMessageSize="30" Direction="DESTINATION" MaxNbMessages="30"/>
    <Queueing_Port PortName="Stat_4Dq" MaxMessageSize="30" Direction="DESTINATION" MaxNbMessages="30"/>
    <Queueing_Port PortName="Stat_5Dq" MaxMessageSize="30" Direction="DESTINATION" MaxNbMessages="30"/>
  </Partition>
  <Partition PartitionIdentifier="2" PartitionName="flight controls" Criticality="LEVEL_A" SystemPartition="false"
  EntryPoint="Initial">
    <Sampling_Port PortName="Act_1Ss" MaxMessageSize="20" Direction="SOURCE" RefreshRateSeconds="0.100"/>
    <Sampling_Port PortName="Act_2Ss" MaxMessageSize="20" Direction="SOURCE" RefreshRateSeconds="0.100"/>
    <Sampling_Port PortName="Sens_1Ds" MaxMessageSize="40" Direction="DESTINATION"
    RefreshRateSeconds="0.100"/>
  </Partition>

```


APPENDIX I EXAMPLE ARINC 653 XML INSTANCE FILE

```

    <Sampling_Port PortName="Sens_2Ds" MaxMessageSize="40" Direction="DESTINATION"
RefreshRateSeconds="0.100"/>
    <Queuing_Port PortName="Stat_2Sq" MaxMessageSize="30" Direction="SOURCE" MaxNbMessages="30"/>
  </Partition>
  <Partition PartitionIdentifier="3" PartitionName="flight management" Criticality="LEVEL_A" SystemPartition="false"
EntryPoint="Initial">
    <Sampling_Port PortName="Sens_2Ds" MaxMessageSize="40" Direction="DESTINATION"
RefreshRateSeconds="0.100"/>
    <Queuing_Port PortName="Stat_3Sq" MaxMessageSize="30" Direction="SOURCE" MaxNbMessages="30"/>
  </Partition>
  <Partition PartitionIdentifier="4" PartitionName="IO Processing" Criticality="LEVEL_A" SystemPartition="true"
EntryPoint="Initial">
    <Queuing_Port PortName="Stat_4Sq" MaxMessageSize="30" Direction="SOURCE" MaxNbMessages="30"/>
  </Partition>
  <Partition PartitionIdentifier="5" PartitionName="IHVM" Criticality="LEVEL_B" SystemPartition="false"
EntryPoint="Initial">
    <Sampling_Port PortName="Act_1Ds" MaxMessageSize="20" Direction="DESTINATION"
RefreshRateSeconds="0.100"/>
    <Sampling_Port PortName="Act_2Ds" MaxMessageSize="20" Direction="DESTINATION"
RefreshRateSeconds="0.100"/>
    <Queuing_Port PortName="Stat_5Sq" MaxMessageSize="30" Direction="SOURCE" MaxNbMessages="30"/>
  </Partition>
  <Partition_Memory PartitionIdentifier="1" PartitionName="system management">
    <Memory_Requirements Type="CODE" SizeBytes="20000" Access="READ_ONLY"/>
    <Memory_Requirements Type="DATA" SizeBytes="20000" Access="READ_WRITE"/>
    <Memory_Requirements Type="INPUT_OUTPUT" SizeBytes="128000" PhysicalAddress="0xFF000000"
Access="READ_WRITE"/>
  </Partition_Memory>
  <Partition_Memory PartitionIdentifier="2" PartitionName="flight controls">
    <Memory_Requirements Type="CODE" SizeBytes="25000" Access="READ_ONLY"/>
    <Memory_Requirements Type="DATA" SizeBytes="25000" Access="READ_WRITE"/>
  </Partition_Memory>
  <Partition_Memory PartitionIdentifier="3" PartitionName="flight management">
    <Memory_Requirements Type="CODE" SizeBytes="35000" Access="READ_ONLY"/>
    <Memory_Requirements Type="DATA" SizeBytes="25000" Access="READ_WRITE"/>
  </Partition_Memory>
  <Partition_Memory PartitionIdentifier="4" PartitionName="IO Processing">
    <Memory_Requirements Type="CODE" SizeBytes="50000" Access="READ_ONLY"/>
    <Memory_Requirements Type="DATA" SizeBytes="75000" Access="READ_WRITE"/>
    <Memory_Requirements Type="INPUT_OUTPUT" SizeBytes="256000" PhysicalAddress="0x50000000"
Access="READ_WRITE"/>
    <Memory_Requirements Type="INPUT_OUTPUT" SizeBytes="512000" PhysicalAddress="0x80000000"
Access="READ_WRITE"/>
  </Partition_Memory>
  <Partition_Memory PartitionIdentifier="5" PartitionName="IHVM">
    <Memory_Requirements Type="CODE" SizeBytes="50000" Access="READ_ONLY"/>
    <Memory_Requirements Type="DATA" SizeBytes="100000" Access="READ_WRITE"/>
  </Partition_Memory>
  <Module_Schedule MajorFrameSeconds="0.200">
    <Partition_Schedule PartitionIdentifier="1" PartitionName="system management" PeriodSeconds="0.100"
PeriodDurationSeconds="0.020">
      <Window_Schedule WindowIdentifier="101" WindowStartSeconds="0.0" WindowDurationSeconds="0.020"
PartitionPeriodStart="true"/>
      <Window_Schedule WindowIdentifier="102" WindowStartSeconds="0.1" WindowDurationSeconds="0.020"
PartitionPeriodStart="true"/>
    </Partition_Schedule>
    <Partition_Schedule PartitionIdentifier="2" PartitionName="flight controls" PeriodSeconds="0.100"
PeriodDurationSeconds="0.010">
      <Window_Schedule WindowIdentifier="201" WindowStartSeconds="0.030" WindowDurationSeconds="0.010"
PartitionPeriodStart="true"/>
      <Window_Schedule WindowIdentifier="202" WindowStartSeconds="0.130" WindowDurationSeconds="0.010"
PartitionPeriodStart="true"/>
    </Partition_Schedule>
    <Partition_Schedule PartitionIdentifier="3" PartitionName="flight management" PeriodSeconds="0.100"
PeriodDurationSeconds="0.030">
      <Window_Schedule WindowIdentifier="301" WindowStartSeconds="0.040" WindowDurationSeconds="0.030"
PartitionPeriodStart="true"/>
      <Window_Schedule WindowIdentifier="302" WindowStartSeconds="0.140" WindowDurationSeconds="0.030"
PartitionPeriodStart="true"/>
    </Partition_Schedule>
    <Partition_Schedule PartitionIdentifier="4" PartitionName="IO Processing" PeriodSeconds="0.100"
PeriodDurationSeconds="0.020">

```


APPENDIX I EXAMPLE ARINC 653 XML INSTANCE FILE

```

    <Window_Schedule WindowIdentifier="401" WindowStartSeconds="0.020" WindowDurationSeconds="0.010"
PartitionPeriodStart="true"/>
    <Window_Schedule WindowIdentifier="402" WindowStartSeconds="0.070" WindowDurationSeconds="0.010"
PartitionPeriodStart="false"/>
    <Window_Schedule WindowIdentifier="403" WindowStartSeconds="0.120" WindowDurationSeconds="0.010"
PartitionPeriodStart="true"/>
    <Window_Schedule WindowIdentifier="404" WindowStartSeconds="0.170" WindowDurationSeconds="0.010"
PartitionPeriodStart="false"/>
  </Partition_Schedule>
  <Partition_Schedule PartitionIdentifier="5" PartitionName="IHVM" PeriodSeconds="0.200"
PeriodDurationSeconds="0.020">
    <Window_Schedule WindowIdentifier="501" WindowStartSeconds="0.180" WindowDurationSeconds="0.020"
PartitionPeriodStart="true"/>
  </Partition_Schedule>
</Module_Schedule>
<Partition_HM_Table PartitionIdentifier="2" PartitionName="flight controls" PartitionCallback="partition_HM_callback">
  <System_State_Entry SystemState="4" Description="partition init">
    <Error_ID_Action ErrorIdentifier="3" Description="partition config error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="4" Description="partition init error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IGNORE"/>
  </System_State_Entry>
  <System_State_Entry SystemState="6" Description="partition error handler">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="WARM_START"/>
  </System_State_Entry>
  <System_State_Entry SystemState="7" Description="process execution">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="IDLE"/>
  </System_State_Entry>
</Partition_HM_Table>
<Partition_HM_Table PartitionIdentifier="3" PartitionName="flight management"
PartitionCallback="partition_HM_callback">
  <System_State_Entry SystemState="4" Description="partition init">
    <Error_ID_Action ErrorIdentifier="3" Description="partition config error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="4" Description="partition init error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IGNORE"/>
  </System_State_Entry>
  <System_State_Entry SystemState="6" Description="partition error handler">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="WARM_START"/>
  </System_State_Entry>
  <System_State_Entry SystemState="7" Description="process execution">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="IDLE"/>
  </System_State_Entry>
</Partition_HM_Table>
<Partition_HM_Table PartitionIdentifier="4" PartitionName="IO Processing" PartitionCallback="partition_IO_callback">
  <System_State_Entry SystemState="4" Description="partition init">
    <Error_ID_Action ErrorIdentifier="3" Description="partition config error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="4" Description="partition init error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IGNORE"/>
  </System_State_Entry>
  <System_State_Entry SystemState="6" Description="partition error handler">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>

```

APPENDIX I EXAMPLE ARINC 653 XML INSTANCE FILE

```

    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="WARM_START"/>
  </System_State_Entry>
  <System_State_Entry SystemState="7" Description="process execution">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="IDLE"/>
  </System_State_Entry>
</Partition_HM_Table>
<Partition_HM_Table PartitionIdentifier="5" PartitionName="IHVM" PartitionCallback="partition_HM_callback">
  <System_State_Entry SystemState="4" Description="partition init">
    <Error_ID_Action ErrorIdentifier="3" Description="partition config error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="4" Description="partition init error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="COLD_START"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IGNORE"/>
  </System_State_Entry>
  <System_State_Entry SystemState="6" Description="partition error handler">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="IGNORE"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="WARM_START"/>
  </System_State_Entry>
  <System_State_Entry SystemState="7" Description="process execution">
    <Error_ID_Action ErrorIdentifier="5" Description="segmentation error" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="6" Description="time duration exceeded" Action="WARM_START"/>
    <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="8" Description="divide by 0" Action="IDLE"/>
    <Error_ID_Action ErrorIdentifier="9" Description="overflow" Action="IDLE"/>
  </System_State_Entry>
</Partition_HM_Table>
<Connection_Table>
  <Channel ChannelIdentifier="1" ChannelName="Sensor 1">
    <Source>
      <Pseudo_Partition Name="Pseudo Partition 1" Procedure="receive_485_1" PhysicalAddress="0x80000000"/>
    </Source>
    <Destination>
      <Standard_Partition PartitionIdentifier="2" PartitionName="Flight Controls" PortName="Sens_1Ds"
PhysicalAddress="0x80000000"/>
    </Destination>
  </Channel>
  <Channel ChannelIdentifier="2" ChannelName="Sensor 2">
    <Source>
      <Pseudo_Partition Name="Pseudo Partition 2" Procedure="receive_485_2" PhysicalAddress="0x80000100"/>
    </Source>
    <Destination>
      <Standard_Partition PartitionIdentifier="2" PartitionName="Flight Controls" PortName="Sens_2Ds"
PhysicalAddress="0x80000100"/>
    </Destination>
    <Destination>
      <Standard_Partition PartitionIdentifier="3" PartitionName="Flight Management" PortName="Sens_2Ds"
PhysicalAddress="0x80000100"/>
    </Destination>
  </Channel>
  <Channel ChannelIdentifier="3" ChannelName="Actuator_1">
    <Source>
      <Standard_Partition PartitionIdentifier="2" PartitionName="Flight Controls" PortName="Act_1Ss"/>
    </Source>
    <Destination>
      <Standard_Partition PartitionIdentifier="5" PartitionName="IVHM" PortName="Act_1Ds"/>
    </Destination>
    <Destination>
      <Pseudo_Partition Name="Pseudo Partition 3" Procedure="send_485_1"/>
    </Destination>
  </Channel>
  <Channel ChannelIdentifier="4" ChannelName="Actuator_2">
    <Source>
      <Standard_Partition PartitionIdentifier="2" PartitionName="Flight Controls" PortName="Act_2Ss"/>
    </Source>
  </Channel>

```

APPENDIX I
EXAMPLE ARINC 653 XML INSTANCE FILE

```

    <Destination>
      <Standard_Partition PartitionIdentifier="5" PartitionName="IVHM" PortName="Act_2Ds"/>
    </Destination>
    <Destination>
      <Pseudo_Partition Name="Pseudo Partition 4" Procedure="send_485_2"/>
    </Destination>
  </Channel>
  <Channel ChannelIdentifier="5" ChannelName="Status_2">
    <Source>
      <Standard_Partition PartitionIdentifier="2" PartitionName="Flight Controls" PortName="Stat_2Sq"/>
    </Source>
    <Destination>
      <Standard_Partition PartitionIdentifier="1" PartitionName="System Management" PortName="Stat_2Dq"/>
    </Destination>
  </Channel>
  <Channel ChannelIdentifier="6" ChannelName="Status_3">
    <Source>
      <Standard_Partition PartitionIdentifier="3" PartitionName="Flight Management" PortName="Stat_3Sq"/>
    </Source>
    <Destination>
      <Standard_Partition PartitionIdentifier="1" PartitionName="System Management" PortName="Stat_3Dq"/>
    </Destination>
  </Channel>
  <Channel ChannelIdentifier="7" ChannelName="Status_4">
    <Source>
      <Standard_Partition PartitionIdentifier="4" PartitionName="IO Processing" PortName="Stat_4Sq"/>
    </Source>
    <Destination>
      <Standard_Partition PartitionIdentifier="1" PartitionName="System Management" PortName="Stat_4Dq"/>
    </Destination>
  </Channel>
  <Channel ChannelIdentifier="8" ChannelName="Status_5">
    <Source>
      <Standard_Partition PartitionIdentifier="5" PartitionName="IVHM" PortName="Stat_5Sq"/>
    </Source>
    <Destination>
      <Standard_Partition PartitionIdentifier="1" PartitionName="System Management" PortName="Stat_5Dq"/>
    </Destination>
  </Channel>
</Connection_Table>
</ARINC_653_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 was deleted that says no message is lost in the queuing mode. 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 then 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 clarifying 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 corrects 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 ARINC653 specification section 2.3.1.4.2.2 (2) it 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_QUEUEING_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_QUEUEING_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.

3.6.2.2.3 RECEIVE_QUEUEING_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 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 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 note under Figure 5 clarifying why the units for RefreshPeriodSeconds were selected.

The schema file “A653_Part_1_revA.xsd” has updates 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 PartitionIdentifier
 - 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.

ARINC Standard – Errata Report

1. Document Title

Draft 4 of Supplement 2 to ARINC Specification 653: Avionics Application Software Standard Interface, Part 1 - Required Services – Overview
Published:

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

(State why the correction is necessary.)

6. Submitter (Optional)

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

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.

Please return comments to fax +1 410-266-2047 or standards@arinc.com

ARINC IA Project Initiation/Modification (APIM) Guidelines for Submittal

1. ARINC Industry Activities Projects and Work Program

A project is established in order to accomplish a technical task approved by one or more of the committees (AEEC, AMC, FSEMC) Projects generally but not exclusively result in a new ARINC standard or modify an existing ARINC standard. All projects are typically approved on a calendar year basis. Any project extending beyond a single year will be reviewed annually before being re-authorized. The work program of Industry Activities (IA) consists of all projects authorized by AEEC, AMC, or FSEMC (The Committees) for the current calendar year.

The Committees establish a project after consideration of an ARINC Project Initiation/Modification (APIM) request. This document includes a template which has provisions for all of the information required by The Committees to determine the relative priority of the project in relation to the entire work program.

All recommendations to the committees to establish or reauthorize a project, whether originated by an airline or from the industry, should be prepared using the APIM template. Any field that cannot be filled in by the originator may be left blank for subsequent action.

2. Normal APIM Evaluation Process

Initiation of an APIM

All proposed projects must be formally initiated by filling in the APIM template. An APIM may be initiated by anyone in the airline community, e.g., airline, vendor, committee staff.

Staff Support

All proposed APIMs will be processed by committee staff. Each proposal will be numbered, logged, and evaluated for completeness. Proposals may be edited to present a style consistent with the committee evaluation process. For example, narrative sentences may be changed to bullet items, etc. When an APIM is complete, it will be forwarded to the appropriate Committee for evaluation.

The committee staff will track all ongoing projects and prepare annual reports on progress.

Committee Evaluation and Acceptance or Rejection

The annual work program for each Committee is normally established at its annual meeting. Additional work tasks may be evaluated at other meetings held during the year. Each committee (i.e., AMC, AEEC, FSEMC) has its own schedule of annual and interim meetings.

The committee staff will endeavor to process APIMs and present them to the appropriate Committee at its next available meeting. The Committee will then evaluate the proposal. Evaluation criteria will include:

- Airline support – number and strength of airline support for the project, including whether or not an airline chairman has been identified
- Issues – what technical, programmatic, or competitive issues are addressed by the project, what problem will be solved
- Schedule – what regulatory, aircraft development or modification, airline equipment upgrade, or other projected events drive the urgency for this project
- Accepted proposals will be assigned to a subcommittee for action with one of two priorities:
 - High Priority – technical solution needed as rapidly as possible
 - Routine Priority – technical solution to proceed at a normal pace
- Proposals may have designated coordination with other groups. This means that the final work must be coordinated with the designated group(s) prior to submittal for adoption consideration.
- Proposals that are not accepted may be classified as follows:
 - Deferred for later consideration - the project is not deemed of sufficient urgency to be placed on the current calendar of activities but will be reconsidered at a later date
 - Deferred to a subcommittee for refinement – the subcommittee will be requested to, for example, gain stronger airline support or resolve architectural issues
- Rejected – the proposal is not seen as being appropriate, e.g., out of scope of the committee

3. APIM Template

The following is an annotated outline for the APIM. Proposal initiators are requested to fill in all fields as completely as possible, replacing the italicized explanations in each section with information as available. Fields that cannot be completed may be left blank. When using the Word file version of the following template, update the header and footer to identify the project.

ARINC IA Project Initiation/Modification (APIM)

Name of proposed project

APIM #: _____

Name for proposed project.

Suggested Subcommittee assignment

Identify an existing group that has the expertise to successfully complete the project. If no such group is known to exist, a recommendation to form a new group may be made.

Project Scope

Describe the scope of the project clearly and concisely. The scope should describe “what” will be done, i.e., the technical boundaries of the project. Example: “This project will standardize a protocol for the control of printers. The protocol will be independent of the underlying data stream or page description language but will be usable by all classes of printers.”

Project Benefit

Describe the purpose and benefit of the project. This section should describe “why” the project should be done. Describe how the new standard will improve competition among vendors, giving airlines freedom of choice. This section provides justification for the allocation of both IA and airline resources. Example: “Currently each class of printers implements its own proprietary protocol for the transfer of a print job. In order to provide access to the cockpit printer from several different avionics sources, a single protocol is needed. The protocol will permit automatic determination of printer type and configuration to provide for growth and product differentiation.”

Airlines supporting effort

Name, airline, and contact information for proposed chairman, lead airline, list of airlines expressing interest in working on the project (supporting airlines), and list of airlines expressing interest but unable to support (sponsoring airlines). It is important for airline support to be gained prior to submittal. Other organizations, such as airframe manufacturers, avionics vendors, etc. supporting the effort should also be listed.

Issues to be worked

Describe the major issues to be addressed by the proposed ARINC standard.

Recommended Coordination with other groups

Draft documents may have impact on the work of groups other than the originating group. The APIM writer or, subsequently, The Committee may identify other groups which must be given the opportunity to review and comment upon mature draft documents.

Projects/programs supported by work

If the timetable for this work is driven by a new airplane type, major avionics overhaul, regulatory mandate, etc., that information should be placed in this section. This information is a key factor in assessing the priority of this proposed task against all other tasks competing for subcommittee meeting time and other resources.

Timetable for projects/programs

Identify when the new ARINC standard is needed (month/year).

Documents to be produced and date of expected result

The name and number (if already assigned) of the proposed ARINC standard to be either newly produced or modified.

Comments

Anything else deemed useful to the committees for prioritization of this work.

Meetings

The following table identifies the number of meetings and proposed meeting days needed to produce the documents described above.

Activity	Mtgs	Mtg-Days
<i>Document a</i>	<i># of mtgs</i>	<i># of mtg days</i>
<i>Document b</i>	<i># of mtgs</i>	<i># of mtg days</i>

For IA Staff use

Date Received: _____ **IA Staff Assigned:** _____

Potential impact: _____

(A. Safety B. Regulatory C. New aircraft/system D. Other)

Forward to committee(s) (AEEC, AMC, FSEMC): _____ **Date Forward:** _____

Committee resolution: _____

(0 Withdrawn 1 Authorized 2 Deferred 3 More detail needed 4 Rejected)

Assigned Priority: _____ **Date of Resolution:** _____

A. – High (execute first) B. – Normal (may be deferred for A.)

Assigned to SC/WG: _____