ARINC 653P1-2

Support Guide Document Rev. A

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Release Date | Release/Change Description | Author | Approval |
| A | 04/04/2023 | Initial draft. | J. Urias, O. Perez, D. Ruiz | M. Lopez |

Table of Contents

[1 Introduction 5](#_Toc132179657)

[1.1 Scope 5](#_Toc132179658)

[1.2 Acronyms and Abbreviations 5](#_Toc132179659)

[2 Glossary 6](#_Toc132179660)

[2.1 Chapter 1 6](#_Toc132179661)

[*2.1.1* *O/S* 6](#_Toc132179662)

[*2.1.2* *APEX* 6](#_Toc132179663)

[*2.1.3* *Portability* 6](#_Toc132179664)

[*2.1.4* *Reusability* 6](#_Toc132179665)

[*2.1.5* *Modularity* 6](#_Toc132179666)

[*2.1.6* *Integration of Software of Multiple Criticalities* 6](#_Toc132179667)

[*2.1.7* *Application Partitions* 6](#_Toc132179668)

[*2.1.8* *IMA* 6](#_Toc132179669)

[2.2 CHAPTER 2 7](#_Toc132179670)

[*2.2.1* *Deterministic* 7](#_Toc132179671)

[*2.2.2* *Core* 7](#_Toc132179672)

[*2.2.3* *Partition* 7](#_Toc132179673)

[*2.2.4* *Partition Attributes* 7](#_Toc132179674)

[*2.2.5* *Partition scheduling* 8](#_Toc132179675)

[*2.2.6* *Partition Modes* 9](#_Toc132179676)

[*2.2.7* *Process Attribute* 9](#_Toc132179677)

[*2.2.8* *Process State* 10](#_Toc132179678)

[*2.2.9* *Continuous block* 11](#_Toc132179679)

[*2.2.10* *Intrapartition* 11](#_Toc132179680)

[*2.2.11* *Interpartition* 12](#_Toc132179681)

[*2.2.12* *Message Types* 12](#_Toc132179682)

[*2.2.13* *FIFO* 13](#_Toc132179683)

[*2.2.14* *Channel* 13](#_Toc132179684)

[*2.2.15* *Port* 13](#_Toc132179685)

[*2.2.16* *Sampling Mode* 13](#_Toc132179686)

[*2.2.17* *Queuing Mode* 13](#_Toc132179687)

[*2.2.18* *Buffers* 13](#_Toc132179688)

[*2.2.19* *Blackboards* 13](#_Toc132179689)

[*2.2.20* *Semaphores* 14](#_Toc132179690)

[*2.2.21* *Events* 14](#_Toc132179691)

[*2.2.22* *Health Monitor* 14](#_Toc132179692)

[*2.2.23* *Mutex* 14](#_Toc132179693)

[2.3 VIDEOS 16](#_Toc132179694)

[*2.3.1* *RTOS introduction video* 16](#_Toc132179695)

[*2.3.2* *RTOS* 16](#_Toc132179696)

[*2.3.3* *Task* 16](#_Toc132179697)

[*2.3.4* *Thread* 16](#_Toc132179698)

[*2.3.5* *Process* 16](#_Toc132179699)

[*2.3.6* *Operating system* 16](#_Toc132179700)

[*2.3.7* *ISR* 16](#_Toc132179701)

[*2.3.8* *RTOS Task Scheduling video* 17](#_Toc132179702)

[*2.3.9* *RTOS Schedule* 17](#_Toc132179703)

[*2.3.10* *Memory Management video* 17](#_Toc132179704)

[*2.3.11* *Queue video* 18](#_Toc132179705)

[*2.3.12* *Queue* 18](#_Toc132179706)

[*2.3.13* *Atomic Variable* 19](#_Toc132179707)

[*2.3.14* *Atomic Operation* 19](#_Toc132179708)

[3 ChatGPT Tool 20](#_Toc132179709)

[4 References 24](#_Toc132179710)

List of Figures

[Figure 2.2‑1 Partition Modes diagram 9](https://pinnaclesoftware-my.sharepoint.com/personal/jose_urias_pinnacleaerospace_com/Documents/Microsoft%20Teams%20Chat%20Files/ARINC%20653P1-2%20Support%20Guide%20Document.docx#_Toc132179646)

[Figure 2.2‑2 Process state diagram 11](https://pinnaclesoftware-my.sharepoint.com/personal/jose_urias_pinnacleaerospace_com/Documents/Microsoft%20Teams%20Chat%20Files/ARINC%20653P1-2%20Support%20Guide%20Document.docx#_Toc132179647)

[Figure 2.3‑1 RTOS logic diagram 16](https://pinnaclesoftware-my.sharepoint.com/personal/jose_urias_pinnacleaerospace_com/Documents/Microsoft%20Teams%20Chat%20Files/ARINC%20653P1-2%20Support%20Guide%20Document.docx#_Toc132179648)

[Figure 2.3‑2 Schedule picture assuming single-core processor 17](#_Toc132179649)

[Figure 2.3‑3Memory Block division 18](https://pinnaclesoftware-my.sharepoint.com/personal/jose_urias_pinnacleaerospace_com/Documents/Microsoft%20Teams%20Chat%20Files/ARINC%20653P1-2%20Support%20Guide%20Document.docx#_Toc132179650)

[Figure 2.3‑4Queue in block scheme 19](https://pinnaclesoftware-my.sharepoint.com/personal/jose_urias_pinnacleaerospace_com/Documents/Microsoft%20Teams%20Chat%20Files/ARINC%20653P1-2%20Support%20Guide%20Document.docx#_Toc132179651)

[Figure 2.3‑1 Openai main page 20](#_Toc132179652)

[Figure 2.3‑2 Giving context to ChatGPT 21](#_Toc132179653)

[Figure 2.3‑3 Example about semaphores inside the standard 22](#_Toc132179654)

[Figure 2.3‑4 Example with more concepts inside Partition Modes 23](#_Toc132179655)

[Figure 2.3‑5 Example asking a concept without context. 24](#_Toc132179656)

List of Tables

[Table 1 Acronyms and Abbreviations 4](#_Toc122017418)

# Introduction

## Scope

[1] [2]

The Support Guide shows a glossary related to advance/unusual concepts used in ARINC 653P1-2 chapter 1 and 2, increasing the APEX overview to the standard reader.

## Acronyms and Abbreviations

Table 1 Acronyms and Abbreviations

|  |  |
| --- | --- |
| Term | Definition |
| PC | Personal Computer |
| GUI | Graphical User Interface |
| HM | Health Monitor |
| APEX | Application and Executive |
| RTOS | Real-Time Operating System |
| IMA | Integrated Modular Avionics |
| O/S | Operating System |
| POSIX | Portable Operating System Interface for UNIX |
| API | Application Programming Interface |
| ISR | Interrupt Service Routine |
| PCB | Process Control Block |
| GUI | Graphical User Interface |
| CLI | Command Line Interface |
| DMA | Direct Memory Access |
| HAL | Hardware Abstraction Layer |
| VMM | Virtual Memory Manager |
| BIOS | Basic Input/Output System |
| UEFI | Unified Extensible Firmware Interface |
| ACPI | Advanced Configuration and Power Interface |
| IPC | Inter-Process Communication |
| DMA | Direct Memory Access |
| IDE | Integrated Development Environment |
| JIT | Just-In-Time |
| FIFO | First-in, First-Out |

# Glossary

## Introduction concepts

### *O/S*

Refers to the Operating System, and it is the software that supports a computer’s basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

### *APEX*

It is the combination of Application and Executive (APEX) and refers to the part of the software inside an RTOS, used as an Interface between the O/S of an avionics computer resource and the application software.

It is also the part of the program that defines a set of function calls or APIs that can be used by an application running within the ARINC 653-2 partition to request various operating systems.

### *Portability*

Refers to how minimal rectification of code/Software/Hardware must be done when the system is transported to another aircraft.

### *Reusability*

Refers to the reduction of the amount of customizing required when a component is reused for a new software.

### *Modularity*

Refers to the removing of hardware and software dependencies, reduces the impact on application software from modifications to the overall system.

### *Integration of Software of Multiple Criticalities*

The APEX interface supports the ability to co-locate application software of different levels of criticality.

### *Application Partitions*

It is the portions of software specific to avionics applications supported by the core module.

### *IMA*

The acronym IMA stands for Integrated Modular Avionics. This refers to a design approach for developing avionics systems that involves integrating multiple functions into a single hardware platform.

## System Overview concepts

### *Deterministic*

This means that the scheduling and execution of processes and partitions will always follow a set of well-defined rules, and the response time of the system to requests will be consistent and bound.

### *Core*

This refers to the central processing unit (CPU) of a computer system. It is responsible for executing instructions and performing calculations.

### *Partition*

The unit of partitioning is called a partition, and is basically the same as a program in a single application environment: it comprises data, its own context, configuration attributes, etc.

Is a logical unit of software execution that provides memory and time partitioning between multiple independent applications running on a single physical computing platform or processor.

Each partition has its own memory space and time budget, this means that applications within partition are isolated from other partitions.

### *Partition Attributes*

Inside the partitions are a lot of attributes, there are divided on two categories: **fixed** and **variables** which are going to be mentioned in the following text:

#### *Fixed Attributes*

Fixed attributes are those that cannot be changed once the partition is created. These attributes include the partition identifier, memory requirements, entry point, and lock level. The partition identifier is a unique name that identifies the partition within the system, and the memory requirements indicate the amount of memory required for the partition to run.

##### *Identifier*

Refers A unique identifier used to differentiate the partition from other partitions in the system.

##### *Memory Requirements*

The amount of memory required for the partition to operate.

##### *Period*

The period of the partition, which determines how often the partition is scheduled for execution.

##### *Duration*

The duration of the partition, which determines the maximum amount of time the partition is allowed to execute during each period.

##### *Criticality Level*

A level that determines the priority of the partition relative to other partitions in the system.

##### *Communication Requirements*

The communication requirements of the partition, which determine how it communicates with other partitions in the system.

##### *Partition Health Monitor Table*

A table that stores information about the health of the partition.

##### *Entry Point*

The entry point of the partition, which is the first function executed when the partition is started.

##### *System Partition*

A partition that provides operating system services to other partitions in the system.

#### *Variables Attributes*

They can be changed dynamically during runtime. These attributes include the partition's operating mode, start condition, duration, period, and criticality level.

##### *Lock Level*

It indicates the level of lock associated with the partition. The value of the lock level ranges from 0 to 255, where 0 indicates no locking, and 255 indicates the highest level of locking. A higher lock level indicates that the partition requires more exclusive access to the shared resources.

##### *Operating Mode*

It indicates the operating mode of the partition. There are two operating modes: Normal and Restart. The Normal operating mode indicates that the partition is running, and the Restart operating mode indicates that the partition is restarting.

##### *Start Condition*

It specifies the condition that must be satisfied before the partition can start. The start condition can be based on time, events, or variables. For example, the start condition can be a specific time or the occurrence of an event.

### *Partition scheduling*

It refers to the process of assigning computing resources to different partitions running on the same RTOS. The RTOS must support fixed-priority and preemptive scheduling, and priority inheritance to prevent priority inversion. It must also define a partition bandwidth and maintain a partition schedule table to ensure that each partition meets its timing and performance requirements. Overall, partition scheduling ensures that each partition can meet its timing and performance requirements while sharing the available resources fairly with other partitions.

### *Partition Modes*

It refers to the way in which the processing power and memory resources of a system are allocated to different partitions. The partitions can stay in four modes which are being mentioned in the following text:

Diagram

Description automatically generated

Figure ‑ Partition Modes diagram

#### *IDLE*

In this mode, the partition is not executing and is waiting to be activated.

#### *NORMAL*

In this mode, the partition is executing its normal functionality as defined by its software design and specification.

#### *COLD\_START*

In this mode, the partition is initialized from a cold start, meaning it has no memory contents or context from previous runs. All its variables and data structures are initialized as specified in its software design and specification.

#### *WARM\_START*

In this mode, the partition is initialized from a warm start, meaning it has some memory contents and context from previous runs. Some variables and data structures may retain their values from previous runs, while others may be reset based on the partition's software design and specification.

### *Process Attribute*

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.

#### *Name*

A string that uniquely identifies the process within the system.

#### *Entry Point*

The starting address of the process code.

#### *Stack Size*

The size of the stack allocated for the process.

#### *Base Priority*

The priority assigned to the process. Higher priority processes are given precedence over lower priority processes.

#### *Period*

The time interval between successive releases of the process.

#### *Time Capacity*

The maximum amount of time that a process can run during a single execution cycle.

#### *Deadline*

The time by which a process must complete its execution. The deadline can be categorized into two types: soft and hard deadlines.

##### *Soft deadline*

It is a deadline that is desirable but not mandatory. If a task misses its soft deadline, the system can still function correctly, but its performance may be degraded.

##### *Hard deadline*

It is a deadline that must be met. If a task misses its hard deadline, the system may fail, and the result could be catastrophic. Therefore, it is critical to ensure that tasks with hard deadlines are executed before their deadline expires.

#### *Current Priority*

The actual priority assigned to the process during its execution.

#### *Deadline Time*

The absolute time at which the process must complete its execution.

### *Process State*

The state of a process can change based on the system events and operations that occur, such as I/O completion, signals, and timer expiration. The operating system must keep track of the state of each process in order to schedule and manage them effectively. The process states as viewed by the O/S are as follows:

#### *Dormant*

In this state, the process has been created but is not yet ready to execute. It may be waiting for an event or some other system resource to become available.

#### *Ready*

In this state, the process is waiting to be allocated a processor. It is prepared to execute and is waiting for the system to schedule it.

#### *Running*

In this state, the process has been allocated a processor and is currently executing its instructions.

#### *Waiting*

In this state, the process is waiting for an event to occur or for a system resource to become available. The event could be anything, such as input from the user, completion of a disk read or write operation, or a signal from another process. While waiting, the process is blocked and not executing instructions. Once the event occurs, the process is moved to the ready state and becomes eligible for scheduling.

Diagram

Description automatically generated

Figure ‑ Process state diagram

### *Continuous block*

It 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.

### *Intrapartition*

Intrapartitions refer to the communication and synchronization mechanisms that exist within a single partition of the operating system.

### *Interpartition*

Interpartitions refers to communication and interaction between partitions in an ARINC 653 system. Interpartition communication can be achieved through Interpartition Communication (IPC) ports or a shared memory area. IPC ports allow partitions to exchange messages while maintaining data integrity and preventing unauthorized access. Shared memory areas allow multiple partitions to share data by accessing the same memory space.

### *Message Types*

A message is a piece of data sent from one process to another, containing information such as commands, status updates, or data to be processed. In the following text are some message types in RTOS.

#### *Fixed Messages:*

These messages have a fixed length, and the length is determined at design time. Fixed messages are easy to implement and have a small overhead, making them suitable for real-time systems with strict timing requirements.

#### *Variable Length Messages:*

These messages have a variable length, and the length is determined at runtime. Variable length messages are more flexible than fixed length messages, but they require more overhead because the length of the message must be determined before the message can be processed.

#### *Periodic Messages:*

These messages are generated at regular intervals and are used to communicate time-critical information. They are typically used to update system state or to synchronize system clocks.

#### *Aperiodic Messages:*

These messages are generated in response to an event or a request from another process. Aperiodic messages are used to communicate time-critical information that is not generated at regular intervals. They are typically used to trigger a response or to inform another process of a change in system state.

#### *Unicast messages*

A unicast message is sent from one sender to one receiver. This type of message is useful when there is only one receiver that needs to receive the message.

#### *Multicast messages*

A multicast message is sent from one sender to a group of receivers. This type of message is useful when multiple receivers need to receive the same message.

#### *Broadcast messages*

A broadcast message is sent from one sender to all receivers in the system. This type of message is useful when a message needs to be sent to all tasks in the system.

### *FIFO*

FIFO stands for "first-in, first-out," which means that the process that has been waiting the longest is the first one to be granted access to the resource. When a process is added to the queue, it is added to the end of the line, and when a resource becomes available, the process at the front of the queue is granted access.

### *Channel*

A channel is a method for interprocess communication that allows one process to send a message to another process. It provides a buffer to temporarily store messages that are being sent and received. Channels can be used for both intra- and inter-partition communication, and can be created with different communication properties, such as FIFO or priority queuing.

### *Port*

It is an entity used for communication between partitions or processes in a distributed system**.** Ports can be viewed as endpoints of channels and are responsible for receiving and transmitting messages between partitions or processes.

### *Sampling Mode*

Sampling mode is a way for a process to periodically read a resource without waiting for an external event to trigger it.

*For example, in a system where sensors are collecting data, a process might be set to sample the sensor readings every second. The process can then perform computations or take actions based on the current readings without waiting for an external trigger.*

### *Queuing Mode*

It is a communication mechanism where messages are stored in a queue until they can be processed. It allows for more flexibility in message handling and can handle a larger number of messages.

*For example, in a printer server, Multiple users send to the printer at the same time, but the printer can only handle one job at a time. In this case, the printer server would use a queuing mode to manage the print jobs.*

### *Buffers*

Buffers store messages and process in queues. Those are stored in FIFO order and the messages cannot 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. In the case of priority order, processes with the same priority are queued in FIFO order.

### *Blackboards*

A blackboard is a messaging system that does not allow message queuing. Any message written on a blackboard remains there until it is cleared or overwritten by a new message. Processes can read, display, or clear messages from the blackboard.

### *Semaphores*

Counting semaphores are used to control access to partition resources. A process waits on a semaphore to access a resource and signals the semaphore when finished. The value of a semaphore indicates the number of available resources. Processes waiting on a semaphore are queued in FIFO or priority order, with the queuing discipline defined at creation.

### *Events*

An event is a communication mechanism which permits notification of an occurrence of a condition to processes which may wait for it.

### *Health Monitor*

Health Monitor is a service that monitors the health of the partitions, processes, and other system resources. It uses a table to keep track of the health of each partition, process, and other system resources. If a partition or process fails to respond or perform as expected, the Health Monitor takes corrective action, such as restarting the partition or process. The Health Monitor also provides a way for partitions to communicate with each other to exchange health-related information. It is an essential component of a real-time operating system, as it ensures the system's reliability and availability. Components of the HM are contained within the following software elements:

#### *Core O/S*

This is the software component that manages the overall system operations, including system initialization, resource allocation, and scheduling of tasks. It also provides services such as memory management, inter-process communication, and input/output operations.

#### *Application Partitions*

These are software components that perform specific tasks or functions within the system. Each application partition is allocated its own memory space and resources, and runs independently of other partitions. The Health Monitor monitors the status and performance of each application partition to ensure they are functioning correctly.

#### *System Partitions*

These are software components that are essential to the overall operation of the system. They are responsible for providing critical system services, such as timekeeping, interrupt handling, and exception handling. The Health Monitor closely monitors the status and performance of these system partitions to ensure they are functioning correctly and to detect any faults or errors.

### *Mutex*

It is a synchronization object used to protect shared resources, such as memory or files, from concurrent access by multiple threads of execution. It acts as a gatekeeper that prevents simultaneous access to a shared resource. A thread that wants to access the shared resource must first acquire the Mutex, which grants the thread exclusive access to the resource. Once the thread has finished accessing the resource, it must release the Mutex so that other threads can acquire it and access the resource.

## VIDEOS

### *RTOS introduction video*

<https://www.youtube.com/watch?v=F321087yYy4>

### *RTOS*

Real Time Operator System that can help you by using multiple tasks in a brief time dividing features into individual tasks.

Diagram

Description automatically generated

Figure ‑ RTOS logic diagram

### *Task*

Any piece of work that we want to get done in code.

### *Thread*

Unit of CPU with its own counter and set of memory.

### *Process*

Instance of a computer program that’s being executed.

### *Operating system*

It is a piece of Software that runs on a computer or microcontroller that accomplishes several essential functions.

### *ISR*

Interrupt Service Routine is a function or routine in a computer's operating system or firmware that handles interrupt requests generated by hardware devices such as input/output controllers, network cards, or timers.

### *RTOS Task Scheduling video*

<https://www.youtube.com/watch?v=95yUbClyf3E>

### *RTOS Schedule*

In the following picture is a scheduler assuming a single-core processor working on RTOS system.

Chart, box and whisker chart

Description automatically generated

Figure ‑ Schedule picture assuming single-core processor

The O/S (Operator System) has the tasks to every time slice to figure out which task to schedule next.

The timer calls the scheduler task at every interval of time, this can call as tick.

In the schedule, there are different tasks which have a priority designed. On figure 1, the task “A” has a low level of priority and task “B” & “C” are high level priority. The O/S gives the instructions to start the tasks and if there are any tasks in queue. The O/S designates again the last working activity to complete the task.

When there are any tasks working, the O/S becomes an Idle state waiting for some recent activity. On the Figure 1, the task “A” worked again. So, the status changed to Idle into task “A.”

When another task with more priority than the already task. The O/S make a standby this activity and focused to do the task that is more important. When they are at same priority, the O/S works them with a few periods of time at only one task per cycle.

If the priority activity is already finished, the last task which was interrupted comes back.

### *Memory Management video*

<https://www.youtube.com/watch?v=Qske3yZRW5I&t=1s>

#### *Static Memory*

This section saves the information which does not change or cannot disappear. That means that it has always been here.

A picture containing shape

Description automatically generated

Figure ‑Memory Block division

#### *Stack Memory*

It is the memory space where the information will be needed to complete the tasks. That information is saved as stacking blocks and it is deleted when the task is complete.

#### *Heap Memory*

It is used to save the Dynamic allocation.

### *Queue video*

<https://www.youtube.com/watch?v=pHJ3lxOoWeI&t=229s>

### *Queue*

It is used to keep the tasks waiting for their turn to do their activity as a queue. When OS start the task which are waiting

A picture containing chart

Description automatically generated

Figure ‑Queue in block scheme

### *Atomic Variable*

It is a variable that can be read or written to atomically, meaning that the operation is performed as a single, indivisible operation. This ensures that the variable is always in a consistent state, even if multiple threads are accessing it at the same time. Atomic variables are often used in multi-threaded programming to implement synchronization primitives such as locks, semaphores, and barriers.

### *Atomic Operation*

It is a single, indivisible operation that occurs without interruption or interference from other concurrent operations. This means that the operation is executed in its entirety, and no other operation can access or modify the data being operated on until the atomic operation is complete. They are often used in multi-threaded programming to ensure that data is correctly synchronized and that no two threads try to access the same data simultaneously.

# References

|  |  |
| --- | --- |
| [1] | E. T. Choi, O. K. Ha and Y. K. Jun, Configuration tool for ARINC 653 operating systems, Vols. vol. 9, no. 4, International Journal of Multimedia and Ubiquitous Engineering, 2014, p. pp. 73–84. |
| [2] | I. Lafoz, M. A. Mozas, O. Charrier, C. F. D. L. Hoz, Eads-Casa and J. Lennon, IMA Systems Development and Configuration: From UML to Binary. A Proposal of UML Profile to be used with the ARINC 653 Standard. |
| [3] | Airlines Electronic Engineering Committee, Avionics Application Software Standard Interface: ARINC Specification 653P1-2, Aeronautical Radio, 2006.  Youtube videos playlist  https://www.youtube.com/watch?v=F321087yYy4&list=PLEBQazB0HUyQ4hAPU1cJED6t3DU0h34bz |