GLOBALPLATFORM®

THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

GlobalPlatform Technology

Virtual Primary Platform – VPP Firmware Format

Version 1.0.1.12

Public Review

February 2021

Document Reference: GPC_SPE_143
(formerly GPC_FST_143)

Contents

Figures

Tables

1 Introduction

1.1 Audience

This document specifies the VPP Firmware Format, which enables the Primary Platform Maker to reach the needed interoperability with other Firmware Makers as defined in [OFL]. The Firmware Maker shall use this format as the basis for an image generation as defined in [OFL] in conjunction with the VPP Concepts and Interfaces specification ([VCI]).

1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://globalplatform.org/specifications/ip-disclaimers/. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

1.3 References

Table 1-1: Normative References

Standard / Specification	Description	Ref
GPC_SPE_134	GlobalPlatform Technology Open Firmware Loader for Tamper Resistant Element v2.0	[OFL]
GPC_SPE_142	GlobalPlatform Technology Virtual Primary Platform – Concepts and Interfaces v2.0	[VCI]
RFC 4122	A Universally Unique IDentifier (UUID) URN Namespace	[RFC 4122]

Table 1-2: Informative References

Standard / Specification	Description	Ref
GSMA iUICC PoC	GSMA iUICC POC Group Primary Platform Requirements approved release Version 1.0, 18 May 2017	[IUICC Req]

1.4 Terminology and Definitions

Selected terms used in this document are included in Table 1-3.

Table 1-3: Terminology and Definitions

Term	Definition
Execution Domain	Definition in [VCI].
Firmware	Definition in [VCI]. See also *VPP Firmware*.

Term	Definition
Firmware Maker	Entity creating the Firmware.
Firmware Segment	Definition in [OFL].
Kernel	Definition in [VCI].
LIB Descriptor	Shared library metadata, as defined in this specification.
Low Level Operating System (LLOS)	Definition in [VCI].
Memory Partition	Concatenated set of Sub-Memory Partitions related to a VPP Firmware.
Part Number Identifier	Universally unique and uniform identifier, as defined in [OFL].
Primary Platform	Definition in [VCI].
Process	Definition in [VCI].
Process Descriptor	Process metadata, as defined in this specification.
Sub-Memory Partition	Ordered set of data mapping the content of the memory regions of a Process Descriptor or a LIB Descriptor.
Tamper Resistant Element (TRE)	Definition in [VCI].
Virtual Primary Platform (VPP)	Definition in [VCI].
Virtual Register (VRE)	Definition in [VCI].
VPP Firmware	Firmware compliant with the VPP Firmware Format as defined in this specification.
VPP Firmware Family Identifier	A UUID identifying that a particular VPP Firmware belongs to a particular class of VPP Firmware. Equivalent to Firmware Family Identifier in [OFL].
VPP Firmware Identifier	A UUID identifying a particular VPP Firmware.
VPP Firmware Loader	Definition in [VCI].

1.5 Abbreviations and Notations

In this document, the following conventions apply:

- All lengths are presented in bytes, unless otherwise stated. Each byte is represented by bits b8 to b1, where b8 is the most significant bit and b1 is the least significant bit. In each representation, the leftmost bit is the most significant bit.
- Hexadecimal values are specified between single quotes, e.g., '1F'.
- All bytes specified as RFU shall be set to '00' and all bits specified as RFU shall be set to 0.

Selected abbreviations and notations used in this document are included in Table 1-4.

Table 1-4: Abbreviations

Abbreviation	Meaning
IPC	Inter Process Communication
LIB	Library
LLOS	Low Level Operating System
NVM	Non-Volatile Memory
TRE	Tamper Resistant Element
UUID	Universally Unique IDentifier version 5 as defined in [RFC 4122]
VM	Volatile Memory
VPP	Virtual Primary Platform
VRE	Virtual REgister

1.6 Revision History

GlobalPlatform technical documents numbered *n*.0 are major releases. Those numbered *n*.1, *n*.2, etc., are minor releases where changes typically introduce supplementary items that do not impact backward compatibility or interoperability of the specifications. Those numbered *n.n*.1, *n.n*.2, etc., are maintenance releases that incorporate errata and precisions; all non-trivial changes are indicated, often with revision marks.

Table 1-5: Revision History

Date	Version	Description
March 2018	1.0	Public Release (with document reference GPC_FST_143)
August 2018	1.0.1	Maintenance Release
October 2019	1.0.1.6	Committee Review
December 2019	1.0.1.7	Member Review
February 2021	1.0.1.12	Public Review
TBD	2.0	Public Release (with document reference GPC_SPE_143)

2 Overview

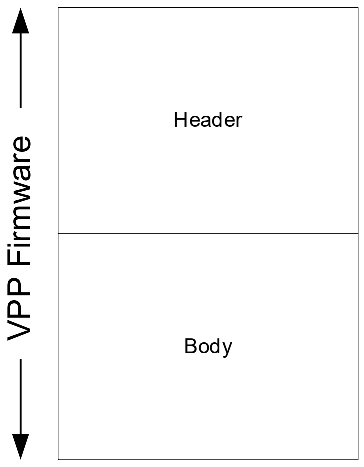
This specification defines the VPP Firmware Format, a data format for loading/updating a VPP Firmware, using a VPP Firmware Loader.

Firmware integrity and confidentiality for data transfer between any non-TRE entity and the TRE is ensured and covered by the Firmware Loader security scheme as defined in [OFL].

VPP Firmware is made of two parts:

- VPP Firmware Header – containing all the metadata of the VPP Firmware.
- VPP Firmware Body – containing the VPP Firmware itself.

Figure 2-1: VPP Firmware Structure



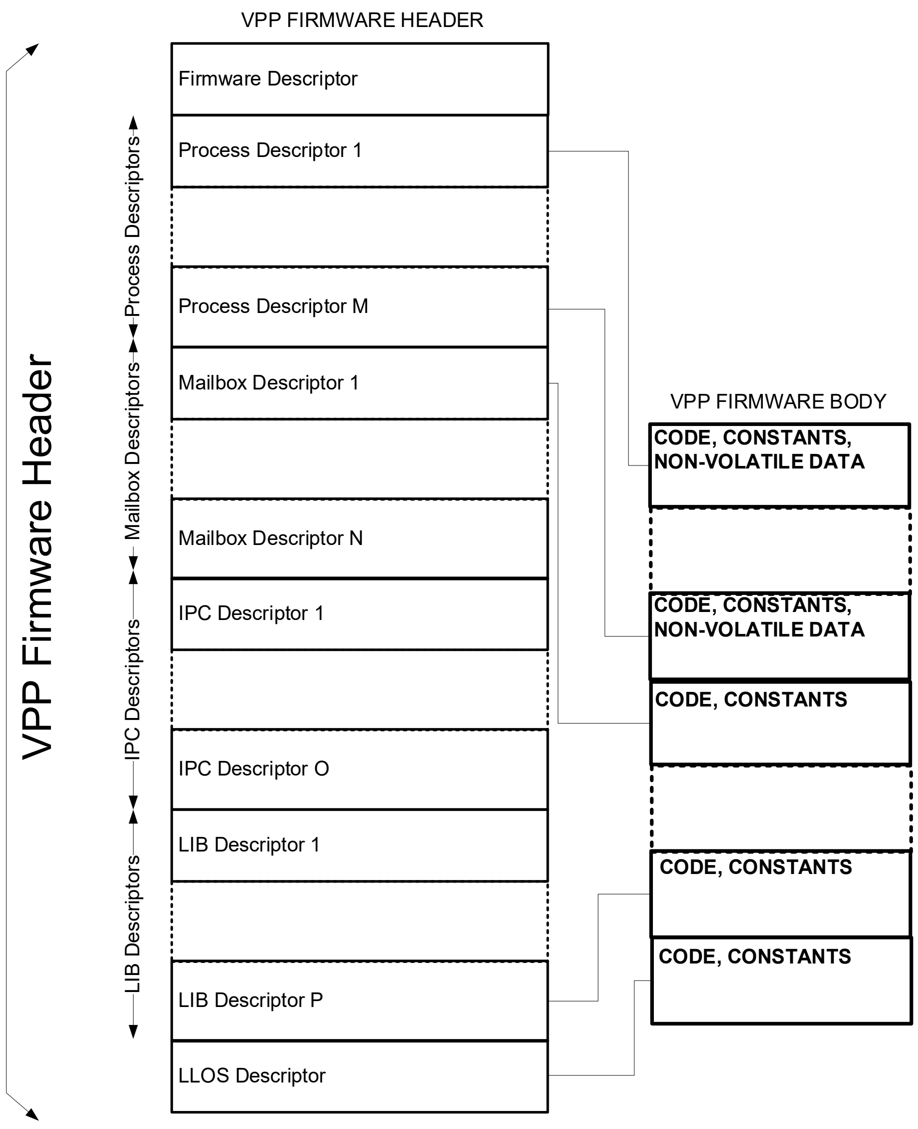
The VPP Firmware Header consists of the following parts:

- VPP Firmware Descriptor
- An array of Process Descriptors
- An array of Mailbox Descriptors
- An array of IPC Descriptors
- An array of LIB Descriptors
- An LLOS Descriptor

All values involving the security rules described in [VCI] are accessible directly from the VPP Firmware Header. Therefore values should be verified without parsing the content of the VPP Firmware Body.

Figure 2-2 illustrates the structure of the VPP Firmware Header Format.

Figure 2-2: VPP Firmware Header Format Structure



3 Basic Data Types and Identifiers

The byte order of any data element in the VPP Firmware Format shall be the native byte order (i.e., endian format) of the CPU within the TRE. The TRE is identified by using the Part Number Identifier as defined in [OFL]; consequently, the nature and the features of the TRE (e.g., endian format) are available for the Firmware Maker.

With the exception of `MK_FORMAT_VERSION`, all basic data types, constants, and identifiers are defined in [VCI].

4 Constants

Table 4-1 defines the constants related to the VPP Firmware Format.

Table 4-1: Constants

Name	Description	Value
`MK_FORMAT_VERSION`	Major and Minor version of the VPP Firmware Format	'0200'

Backward compatibility of all minor releases of the VPP Firmware Format to the associated major release and intervening minor releases shall be ensured.

5 The VPP Firmware Header

The VPP Firmware Header consists of a varying number of descriptors stored in a contiguous memory region. The number and type of descriptors is defined by fields in the VPP Firmware Descriptor as described below. The descriptors are aligned according to their individual alignment constraints. Descriptors are zero-padded to their succeeding descriptor's alignment. The fields inside the descriptors are not aligned, and padding is not needed.

5.1 The VPP Firmware Descriptor

Table 5-1 defines the structure and content of the VPP Firmware Descriptor.

Table 5-1: VPP Firmware Descriptor Structure

Type	Field Name	Description	Size	Notes
`UUID_t`	`m_xFirmwareID`	VPP Firmware Identifier	16	
`UUID_t`	`m_xFamilyID`	VPP Firmware Family Identifier	16	
`uint16_t`	`m_uHeaderLength`	Length of the VPP Firmware Header	2	
`uint16_t`	`m_uVersionFormat`	Major and minor version of the VPP Firmware Format	2	
`uint16_t`	`m_uVersionFirmware`	Major and minor version of the VPP Firmware	2	
`uint8_t`	`m_uProcessCount`	Number of Process Descriptors	1	Maximum is `MK_PROCESS_LIMIT`, as defined in [VCI]
`uint8_t`	`m_uMailboxesCount`	Number of Mailbox Descriptors	1	Maximum is `MK_MAILBOX_LIMIT`, as defined in [VCI]
`uint8_t`	`m_uIPCCount`	Number of IPC Descriptors	1	Maximum is `MK_IPC_LIMIT`, as defined in [VCI]
`uint8_t`	`m_uLIBCount`	Number of shared LIB Descriptors	1	Maximum is `MK_LIB_LIMIT`, as defined in [VCI]
`VPP_FRW_TYPE_e`	`m_eFirmware_Type`	Type of the VPP Firmware	1	Enumerated VPP Firmware Type
`VPP_SCHEDULING_TYPE_e`	`m_eSchedulingType`	Type of scheduling	1	Enumerated Scheduling Type in [VCI]
Total in bytes			44	

The VPP Firmware version (i.e., `m_uVersionFirmware`) is set by a Firmware Maker and may be used by a VPP Firmware Loader.

The VPP Firmware Format version (i.e., `m_uVersionFormat`) according to the present document shall be set to the value of `MK_FORMAT_VERSION`. The portion of `m_uVersionFormat` that specifies the Major release will be incremented for each release of this specification that is not fully backward compatible with the previous release.

The VPP Firmware Descriptor alignment is 32 bits.

5.2 The Process Descriptor

Table 5-2 defines the structure and content of the Process Descriptor.

All offsets are defined from the beginning of the VPP Firmware Body.

Table 5-2: Process Descriptor Structure

Type	Field Name	Description	Size
`PPROCESS_Function_t`	`m_pvProcessCode`	Offset within the CODE segment to the Process entry point	4
`v32_u`	`m_uLength_Process_CODE`	Length of the whole segment for CODE	4
`v32_u`	`m_uLength_Process_CONSTANTS`	Length of the whole segment for CONSTANTS data	4
`v32_u`	`m_uLength_Process_VD`	Length of the whole segment for Volatile Data	4
`v32_u`	`m_uLength_Process_NVD`	Length of the whole segment for Non-Volatile Data	4
`MK_VRE_e`	`m_eVRE`	ORing of Mandatory Access Control for VRE	4
`MK_PROCESS_ID_u`	`m_xID`	Process identifier	2
`MK_MAILBOX_ID_u`	`m_xKernel_Mailbox`	Identifier of the mailbox receiving signals from the kernel	2
`MK_Index_t`	`m_uParent_Process`	Index of the parent Process in the group of processes (self-referenced for `MK_PROCESS_MAIN_APP_ID` process as defined in [VCI])	2
`uint16_t`	`m_uSizeStack`	Size of the stack (in bytes; must be a multiple of 4)	2
Total in bytes			32

The Process array shall be sorted in ascending order of the Process identifiers (`m_xID`).

The first entry shall be considered the MAIN Process.

The Process Descriptor alignment is 32 bits.

5.3 The Mailbox Descriptor

Table 5-3 defines the structure and content of the Mailbox Descriptor.

Table 5-3: Mailbox Descriptor Structure

Type	Field Name	Description	Size
`MK_MAILBOX_ID_u`	`m_xID`	Identifier of the mailbox	2
`MK_Index_t`	`m_uIX_Owner`	Index of the owner/receiver Process in the group of processes	2
`MK_Index_t`	`m_uIX_Sender`	Index of the sender Process in the group of processes	2
Total in bytes			6

The Mailbox array shall be sorted in ascending order of the Mailbox identifiers (`m_xID`).

The Mailbox Descriptor alignment is 16 bits and there is no padding for fields smaller than 16 bits.

5.4 The IPC Descriptor

Table 5-4 defines the structure and content of the IPC Descriptor.

Table 5-4: IPC Descriptor Structure

Type	Field Name	Description	Size
`MK_IPC_ID_u`	`m_xID`	Identifier of the IPC	2
`uint16_t`	`m_uLength_IPC`	Length of the shared memory (in bytes)	2
`MK_Index_t`	`m_uIX_Writer`	Index of the writer Process in the group of processes	2
`MK_Index_t`	`m_uIX_Reader`	Index of the reader Process in the group of processes	2
Total in bytes			8

The IPC array shall be sorted in ascending order of the IPC identifiers (`m_xID`).

The IPC Descriptor alignment is 16 bits.

5.5 The LIB Descriptor

Table 5-5 defines the structure of the LIB Descriptor for a Library shared between multiple processes declared in the VPP Firmware Header. A Library is not shareable between different Execution Domains.

Table 5-5: LIB Descriptor Structure

Type	Field Name	Description	Size
`v32_u`	`m_uLength_LIB_CODE`	Length of the whole segment for CODE	4
`v32_u`	`m_uLength_LIB_CONSTANTS`	Length of the whole segment for CONSTANTS data	4
`MK_BITMAP_t`	`m_uAssigned_Process`	Bitmap indicating which processes have access to the shared library, where LSB is Process index 0 and MSB is Process index 31	4
`MK_LIB_ID_u`	`m_xID`	Identifier of the shared LIBrary	2
Total in bytes			14

The LIB Descriptor array shall be sorted in ascending order of the LIB identifiers (`m_xID`).

The LIB Descriptor alignment is 32 bits.

The VPP Firmware Loader may support the processing of the LIB Descriptor.

5.6 The LLOS Descriptor

Table 5-6 defines the structure and content of the LLOS Descriptor for the LLOS Sub-Memory Partition.

Table 5-6: LLOS Descriptor Structure

Type	Field Name	Description	Size
`PLLOS_Function_t`	`m_pvLLOSCode`	Offset within the CODE segment to the LLOS entry point	4
`v32_u`	`m_uLength_LLOS_CODE`	Length of the whole segment for LLOS CODE	4
`v32_u`	`m_uLength_LLOS_CONSTANTS`	Length of the whole segment for LLOS CONSTANTS data	4
`v32_u`	`m_uLength_LLOS_VD`	Length of the whole segment for LLOS Volatile Data	4
Total in bytes			16

The LLOS Descriptor alignment is 32 bits.

The LLOS has no process.

6 The VPP Firmware Body

The structure of the VPP Firmware Body is left to the implementer of the VPP Firmware Loader. However, this section describes a set of recommendations that may be followed during the VPP Firmware Body implementation.

The VPP Firmware Body may contain the binary executable code, the constants, the initialized data, and the NVM data for each Process and the binary executable code and constant data for each library (LIB) for filling or updating a Memory Partition.

The Memory Partition may be located outside the TRE but it shall be protected by a means inside the TRE as defined in [VCI].

The VPP Firmware Body is segmented in a way that matches the size of the Process Descriptors or the LIB Descriptors.

At the beginning of the loading operations, the Primary Platform allocates a Memory Partition containing `m_uProcessCount` (see Table 5-1) Sub-Memory Partitions and `m_uLIBCount` (see Table 5-1) Sub-Memory Partitions.

The mapping of a Virtual Memory Space related to a Process and using a Sub-Memory Partition is described in [VCI] Figure 5-4.

The VPP Firmware Loader is responsible for fully writing each Firmware Segment in each Sub-Memory Partition related to a given Process, a given shared library, or the LLOS. If data extracted from the Firmware Segment by the VPP Firmware Loader is less than indicated in the VPP Firmware Header, the VPP Firmware Loader shall pad the Firmware Segment with zeros ('00'). All Sub-Memory Partitions are concatenated without gap.

The VPP Firmware Loader shall support the management of Firmware Segments. Each Firmware Segment contains at least the following information:

- The address of the data to be written in the memory space. The offset between the address and the beginning of the region containing the address of the Firmware Segment, is added to the beginning of the address returned by the `_mk_Assign_SubMemoryPartition` defined in [VCI].
- The length of the Firmware Segment.
- The identity of the process (`MK_PROCESS_ID_u`) or the identity of the library (`MK_LIB_ID_u`) as defined in the VPP Firmware Header.

The total number of Firmware Segments depends on the VPP Firmware itself:

- The nature of the operation (initial loading or subsequent updates)
- The fragmentation of data to be written
- The amount of data to be written
- The number of processes
- The number of libraries

7 Consistency Rules

The rules below can be used to evaluate the correctness and validity of the VPP Firmware Format. Correctness may be assumed if all rules evaluate to 'TRUE'.

7.1 The VPP Firmware Descriptor

The following rules apply for the VPP Firmware Descriptor:

- `m_xFirmwareID` and `m_xFamilyID` should be UUIDs version 5 according to [RFC 4122].
- `m_uHeaderLength` shall be equal to:
  - size of VPP Firmware Descriptor + (`m_uProcessCount` x size of Process Descriptor) + (`m_uMailboxesCount` x size of Mailbox Descriptor) + (`m_uIPCCount` x size of IPC Descriptor) + (`m_uLIBCount` x size of LIB Descriptor), or
  - size of VPP Firmware Descriptor + size of LLOS Descriptor
- `m_uVersionFormat = MK_FORMAT_VERSION`
- `m_uProcessCount <= MK_PROCESS_LIMIT` described in [VCI]
- `m_uMailboxesCount <= MK_MAILBOX_LIMIT` described in [VCI]
- `m_uIPCCount <= MK_IPC_LIMIT` described in [VCI]
- `m_uLIBCount <= MK_LIB_LIMIT` described in [VCI]
- `m_eFirmware_Type` shall be in the list of `VPP_FRW_TYPE_e` (Enumerated VPP Firmware Type) described in [VCI].
- `m_xID` of each Process Descriptor, Mailbox Descriptor, and IPC Descriptor shall have the same Execution Domain Type (`MK_EXECUTION_DOMAIN_TYPE_e`) defined in [VCI], which shall be:
  - `MK_EXECUTION_DOMAIN_TYPE_VPP` if `m_eFirmware_Type = FIRMWARE_TYPE_VPP`
  - `MK_EXECUTION_DOMAIN_TYPE_APP` if `m_eFirmware_Type = FIRMWARE_TYPE_APP`
- If `m_eFirmwareType = FIRMWARE_ TYPE_LLOS`, then the following statements shall be true:
  - `m_uProcessCount = 0`
  - `m_uMailboxesCount = 0`
  - `m_uIPCCount = 0`
  - `m_uLIBCount = 0`
  - Only the LLOS Descriptor is present
  - `m_uHeaderLength` is equal to size of VPP Firmware Descriptor + size of LLOS Descriptor
- The VPP Firmware Loader may process a VPP Firmware related to:
  - the `FIRMWARE_TYPE_SYSAPP` type
  - the `FIRMWARE_TYPE_VPP` type
  - the `FIRMWARE_TYPE_LLOS` type
- The sum of the lengths of all non-volatile memory regions (i.e., `m_uLength_Process_CODE`, `m_uLength_Process_CONSTANTS`, `m_uLength_Process_NVD`, `m_uLength_LIB_CODE`, `m_uLength_LIB_CONSTANTS`) of all Process Descriptors and LIB Descriptors shall be less than `MK_MEMORY_PARTITION_SIZE_NVD`.

- The sum of the lengths of all volatile memory regions (i.e., `m_uLength_Process_VD`, `m_uSizeStack`, `m_uLength_IPC`) of all Process Descriptors and IPC Descriptors shall be less than `MK_MEMORY_PARTITION_SIZE_VD`.

7.1.1 Process Hierarchy

- The Process hierarchy for `m_eFirmware_Type = FIRMWARE_TYPE_APP` or `FIRMWARE_TYPE_SYSAPP` shall be represented by a tree.
- The data structure in the VPP Firmware Format, which represents the Process hierarchy tree structure, shall be validated (i.e., a connected acyclic graph whose root is the Process identified by `MK_PROCESS_MAIN_APP_ID`).

7.2 The Process Descriptor

The following rules apply for the Process Descriptor:

- `0 < m_pvProcessCode < m_uLength_Process_CODE`
- `m_eVRE` shall be compliant with the Mandatory Access Control in [VCI]
- `m_xID`
  - If `m_xID` is not part of the Cross-Execution-Domain Identifiers as defined in [VCI] section 7.4, then:
    - `m_xID >= MK_PROCESS_DOMAIN_BASE_ID` and
    - `m_xID <= MK_MAX_PROCESS_ID`
- `m_xKernel_Mailbox`
  - If `m_xKernel_Mailbox` is not part of the Cross-Execution-Domain Identifiers as defined in [VCI] section 7.4, then:
    - `m_xKernel_Mailbox >= MK_MAILBOX_DOMAIN_BASE_ID` and
    - `m_xKernel_Mailbox <= MK_MAILBOX_MAX_ID`
- `m_uParent_Process`
  - If `m_xID = MK_PROCESS_MAIN_APP_ID`, then `m_uParent_Process =` index of the process identified by `m_xID` (the parent process identifier does need to be defined as it is in the VPP Execution Domain, then by default it is assigned to the child process identifier).
  - If `m_xID <> MK_PROCESS_MAIN_APP_ID`, then `m_uParent_Process <>` index of the process identified by `m_xID` (the index of the parent process is different from the index of the process in that case)
- `m_uSizeStack`
  - `m_uSizeStack >= MK_MIN_SUPPORTED_STACK`
  - `m_uSizeStack <= MK_MAX_STACKS_SUM_SUPPORTED`
- The sum of all `m_uSizeStack` of each Process shall be less or equal to `MK_MAX_STACKS_SUM_SUPPORTED`.
- For each Process Descriptor, the overall Sub-Memory Partition size, which is the sum of the lengths of all its memory regions (i.e., `m_uLength_Process_CODE`, `m_uLength_Process_CONSTANTS`, `m_uLength_Process_VD`, `m_uLength_Process_NVD`), shall be less than `MK_MEMORY_PARTITION_SIZE`.

- Each non-volatile memory region size (`m_uLength_Process_CODE`, `m_uLength_Process_CONSTANTS`, `m_uLength_Process_NVD`) shall be less than `MK_VSPACE_REGION_SIZE_NVD`.
- Each volatile memory region size (`m_uLength_Process_VD`, `m_uSizeStack`) shall be less than `MK_VSPACE_REGION_SIZE_VD`.
- The `m_xID` of the Process Descriptor having the index N-1 shall be less than the `m_xID` of the Process Descriptor having the index N.

7.3 The Mailbox Descriptor

The following rules apply for the Mailbox Descriptor:

- `m_xID`:
  - If `m_xID` is not part of the Cross-Execution-Domain Identifiers as defined in section 7.4 of [VCI], then:
    - `m_xID >= MK_MAILBOX_DOMAIN_BASE_ID`
    - `m_xID <= MK_MAILBOX_MAX_ID`
    - `m_xID` shall be unique in the array of Mailbox Descriptors and among all the `Process_Descriptor.m_xKernel_Mailbox`
- `m_uIX_Owner < Firmware_Descriptor.m_uProcessCount`
- `m_uIX_Sender < Firmware_Descriptor.m_uProcessCount`
- `m_uIX_Owner <> m_uIX_Sender`
- The `m_xID` of the Mailbox Descriptor having the index N-1 shall be less than the `m_xID` of the Mailbox Descriptor having the index N.

7.4 The IPC Descriptor

The following rules apply for the IPC Descriptor:

- `m_xID`:
  - If `m_xID` is not part of the Cross-Execution-Domain Identifiers as defined in section 7.4 of [VCI], then:
    - `m_xID >= MK_IPC_DOMAIN_BASE_ID`
    - `m_xID <= MK_IPC_MAX_ID`
    - `m_xID` shall be unique in the array of IPC Descriptors
- `m_uLength_IPC <= MK_IPC_SIZE_LIMIT`
- `m_uIX_Writer < Firmware_Descriptor.m_uProcessCount`
- `m_uIX_Reader < Firmware_Descriptor.m_uProcessCount`
- `m_uIX_Writer <> m_uIX_Reader`
- For each IPC Descriptor, the volatile memory region size (`m_uLength_IPC`) shall be less than `MK_VSPACE_REGION_SIZE_VD`.
- The `m_xID` of the IPC Descriptor having the index N-1 shall be less than the `m_xID` of the IPC Descriptor having the index N.

7.5 The LIB Descriptor

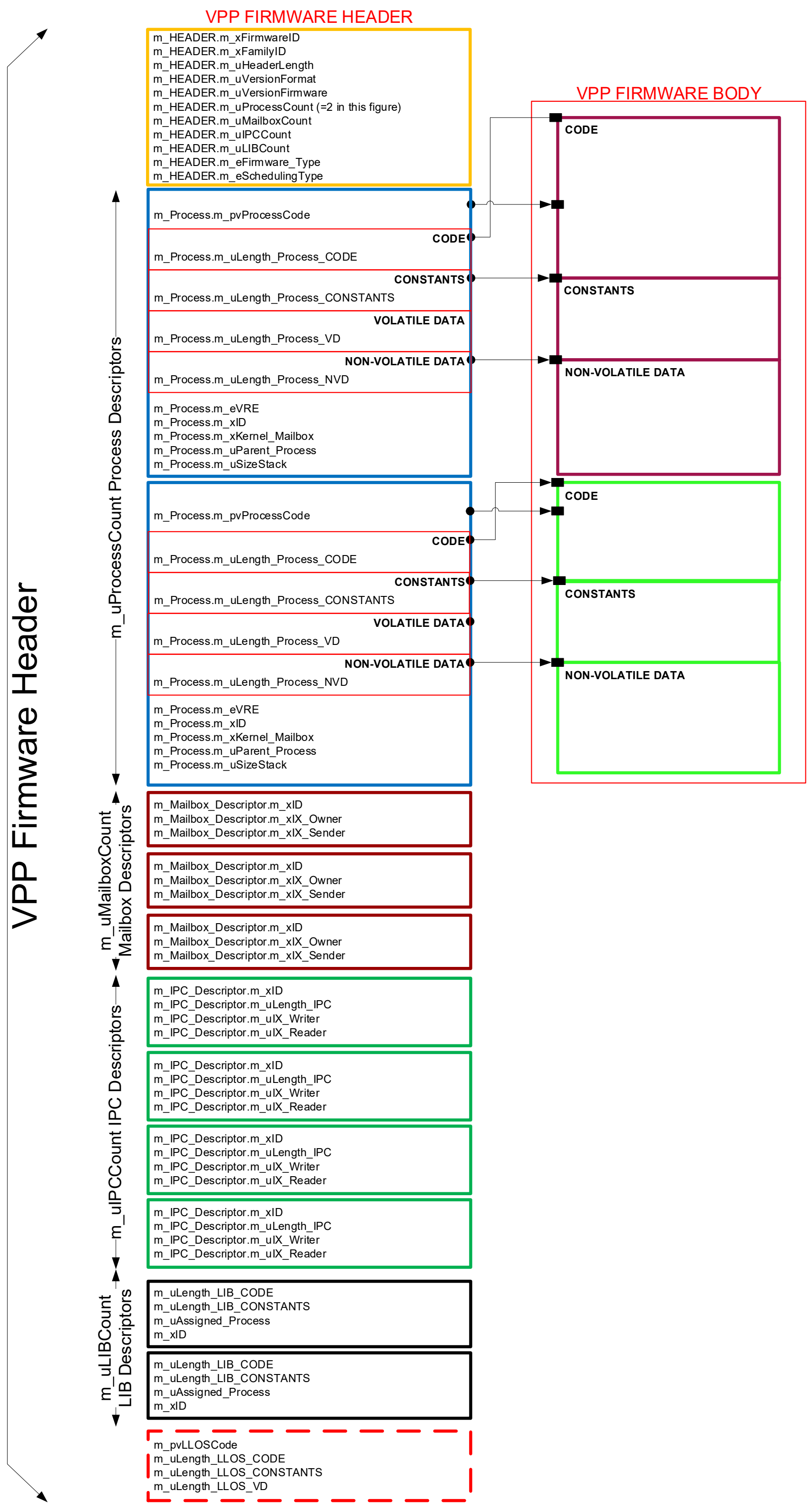
The following rules apply for the LIB Descriptor:

- `m_xID`
  - `m_xID >= MK_LIB_DOMAIN_BASE_ID`
  - `m_xID <= MK_LIB_MAX_ID`
- `m_xID` shall be unique in the array of LIB Descriptors
- `m_uAssigned_Process`
  - The number of bits set to 1 shall be greater than 1 and less than `Firmware_Descriptor.m_uProcessCount`
  - `m_uAssigned_Process <` $2^{\text{Firmware\_Descriptor.m\_uProcessCount}}$
- For each LIB Descriptor, the non-volatile memory region size (i.e., `m_uLength_LIB_CODE`, `m_uLength_LIB_CONSTANTS`) shall be less than `MK_VSPACE_REGION_SIZE_NVD`.
- The `m_xID` of the LIB Descriptor having the index N-1 shall be less than the `m_xID` of the LIB Descriptor having the index N.

Annex A Example Structure of VPP Firmware Format

Figure A-1 illustrates an example of the structure of a VPP Firmware Format with two Processes.

Figure A-1: Example of Structure of VPP Firmware Format


VPP FIRMWARE HEADER
m_HEADER.m_xFirmwareID
m_HEADER.m_xFamilyID
m_HEADER.m_uHeaderLength
m_HEADER.m_uVersionFormat
m_HEADER.m_uVersionFirmware
m_HEADER.m_uProcessCount (=2 in this figure)
m_HEADER.m_uMailboxCount
m_HEADER.m_uIPCCount
m_HEADER.m_uLIBCount
m_HEADER.m_eFirmware_Type
m_HEADER.m_eSchedulingType
VPP Firmware Header
m_uProcessCount Process Descriptors
m_Process.m_pvProcessCode
CODE
m_Process.m_uLength_Process_CODE
CONSTANTS
m_Process.m_uLength_Process_CONSTANTS
VOLATILE DATA
m_Process.m_uLength_Process_VD
NON-VOLATILE DATA
m_Process.m_uLength_Process_NVD
m_Process.m_eVRE
m_Process.m_xID
m_Process.m_xKernel_Mailbox
m_Process.m_uParent_Process
m_Process.m_uSizeStack
m_Process.m_pvProcessCode
CODE
m_Process.m_uLength_Process_CODE
CONSTANTS
m_Process.m_uLength_Process_CONSTANTS
VOLATILE DATA
m_Process.m_uLength_Process_VD
NON-VOLATILE DATA
m_Process.m_uLength_Process_NVD
m_Process.m_eVRE
m_Process.m_xID
m_Process.m_xKernel_Mailbox
m_Process.m_uParent_Process
m_Process.m_uSizeStack
VPP FIRMWARE BODY
CODE
CONSTANTS
NON-VOLATILE DATA
CODE
CONSTANTS
NON-VOLATILE DATA
m_uMailboxCount Mailbox Descriptors
m_Mailbox_Descriptor.m_xID
m_Mailbox_Descriptor.m_xIX_Owner
m_Mailbox_Descriptor.m_xIX_Sender
m_Mailbox_Descriptor.m_xID
m_Mailbox_Descriptor.m_xIX_Owner
m_Mailbox_Descriptor.m_xIX_Sender
m_Mailbox_Descriptor.m_xID
m_Mailbox_Descriptor.m_xIX_Owner
m_Mailbox_Descriptor.m_xIX_Sender
m_uIPCCount IPC Descriptors
m_IPC_Descriptor.m_xID
m_IPC_Descriptor.m_uLength_IPC
m_IPC_Descriptor.m_uIX_Writer
m_IPC_Descriptor.m_uIX_Reader
m_IPC_Descriptor.m_xID
m_IPC_Descriptor.m_uLength_IPC
m_IPC_Descriptor.m_uIX_Writer
m_IPC_Descriptor.m_uIX_Reader
m_IPC_Descriptor.m_xID
m_IPC_Descriptor.m_uLength_IPC
m_IPC_Descriptor.m_uIX_Writer
m_IPC_Descriptor.m_uIX_Reader
m_IPC_Descriptor.m_xID
m_IPC_Descriptor.m_uLength_IPC
m_IPC_Descriptor.m_uIX_Writer
m_IPC_Descriptor.m_uIX_Reader
m_uLIBCount LIB Descriptors
m_uLength_LIB_CODE
m_uLength_LIB_CONSTANTS
m_uAssigned_Process
m_xID
m_uLength_LIB_CODE
m_uLength_LIB_CONSTANTS
m_uAssigned_Process
m_xID
m_pvLLOSCode
m_uLength_LLOS_CODE
m_uLength_LLOS_CONSTANTS
m_uLength_LLOS_VD