GenICam Generic Data Container Specification (GenDC)

Version 1.0.0

Table of Contents

List of Figures

List of Tables

List of Requirements and Objectives

History

This section lists the major milestones of the GenDC specification definition.

Version	Date	Changed by	Change
Version 0.1	2015-03-09	Stephane Maurice, Matrox	Initial draft of GenSP based on the concept of a TL independent, self-described and generic imaging Data Container format.
Version 0.5	2017-04-11	Stephane Maurice, Matrox	GenSP proposal presented for adoption as a GenICam module.
Version 0.7	2017-11-10	Stephane Maurice, Matrox	GenSP renamed to GenDC Some Headers' field names changed. Notion of data Flow added. Added rules and recommendations.
Version 0.82	2018-08-13	Thomas Hopfner, MVTec	General review. Restructuration of the document chapters. Some Headers' field names and position changed. Described the different Descriptor types. Improved Flow notion documentation. Added numbered requirements. Added disclaimer.
Version 1.0.0	2018-12-10	Stéphane Maurice, Matrox	Updated the specification and requirements according to the conclusion of the Austin IVSM meeting and included the corrections received.

1 Introduction

This document describes the Generic Data Container (GenDC) GenICam module. GenDC is Transport Layer neutral, self-described and used to represent, transmit or receive various kinds of data. GenDC targets especially machine vision related image data (such as 2D, 3D, multi-spectral) and metadata (like extra information, histograms and statistics). Besides the GenDC Container layout, this specification also describes the available data types.

A Transport Layer defines how to transport the GenDC Container with the concept of GenDC Flows but it does not know the content of the Container. This allows using and adding data types without touching a particular Transport Layer specification.

1.1 Objectives

The GenDC specification is intended to meet the following objectives:

1. Define a generic and self-described autonomous Data Container usable for representation, transmission and reception of arbitrary data Components.
2. Be Transport Layer agnostic, the Transport Layer is able to transport GenDC as an opaque Data Container without further knowledge.
3. Separate the notion of "What" is the data from the way "How" the data is generated or transported as far as feasible.
4. Be useable in hardware and software implementations.
5. Favor generality, future flexibility and expandability over Transport Layer or media specific definitions.
6. Be able to use the same Container layout anywhere in the data manipulation chain from the sensor data encoding to data delivery.
7. Define a GenDC Container that is also usable for general data storage.
8. Support complex and arbitrary data content (1D, 2D, 3D images, processing results, image sequences/bursts, Multispectral, Metadata, Mixed content …).
9. Support heterogeneous and independent data Component's members.
10. Support multi-plane Components made of individual Parts of various and mixed size data format.
11. Support information metadata (such as GenICam chunk data).
12. Allow separate transfer and storage of Container Descriptor and data sections.
13. Permit reuse of GenDC Container encoder and decoder independent of the Transport Layer protocol.
14. Support simple and efficient implementation of encoding and decoding.
15. Permit the addition of new data types without requiring any Transport Layer protocol specification update.
16. Define a Container structure that supports early processing of data during the reception.

1.2 Terms and Definitions

Name	Description
Component	A single individual element of a Container. A Component has one Header that describes the Component and includes one Part Header for each of its Part(s). A Component can have one or more Parts that belong together.
Container	An object that contains the complete description and data of simple or complex data buffers. A Container has one Descriptor and one or more Components.
Descriptor	A structure describing the Container's organization and its data. The Container's Descriptor groups the Container Header, all the Component Headers and their associated Part Headers and fully describes the content of the Container including the data Offsets. To support various use cases in the Transport Layers, three kinds of Descriptors are defined: Prefetch, Preliminary and Final Descriptor but they all share the same layout.
Flow	A data transport entity that can carry a Descriptor and/or one or more Components and Parts. Allows mapping of a Container's contents to different memory locations and/or parallel transport of it.
FlowOffset	The position of a specific Part in a Flow. It is specified in bytes from the Flow start.
Group	An ensemble of GenDC Components related together.
Header	Structure describing a particular member of the Container's Descriptor.
Offset	The position of a specific element in the Container. The Offset is always in bytes from the Container start except the Part's FlowOffset.
Part	A Part is the basic constituent of a Component and contains the data.
Product	A functional entity using GenDC.
Transport Layer	An entity transporting the data between receivers and transmitters. It typically handles the transfer and representation of data on a particular physical layer like Ethernet, USB3 or coax cables using a well-defined protocol. Examples of Transport Layers protocol in the context of this specification are GigE Vision™, USB3 Vision™, CoaXPress™ and Camera Link HS™.

Table 1-1: Terms and Definitions

1.3 Normative References

GenICam:	EMVA GenICam Specification
GenICam SFNC:	EMVA GenICam Standard Features Naming Convention
GenICam PFNC:	EMVA GenICam Pixel Format Naming Convention
GenICam PFNC Pixel Format Values:	EMVA GenICam Pixel Format Naming Convention Pixel format values list.

1.4 Requirements Terminology

This specification uses the following convention to list requirements.

Term	Description	Representation
Absolute Requirement	The product MUST support this requirement. It is mandatory to support it to ensure interoperability.	[R-<sn><suffix>]
Conditional Requirement	The product MUST support this requirement IF another condition is present. It is mandatory to support it when the other condition is met.	[CR-<sn><suffix>]

Table 1-2: Requirements terminology

A unique number in brackets represents each requirement. Each number is composed of up to three elements:

1. Requirement Type: Absolute **R**equirement, **C**onditional **R**equirement.
2. Sequence number (sn): Unique number identifying the requirement. The sequence numbers are attributed sequentially as new requirements are added to the specification.
3. Suffix: Identifies if the requirement is applicable to application software, transmitters, receivers or some combination of these. Currently not used in this specification.

1.5 Liability Disclaimer

This standard is provided "as is" and without any warranty of any kind, expressed or implied. Without limitation, there is no warranty of non-infringement, no warranty of merchantability, and no warranty of fitness for a particular purpose. All warranties are expressly disclaimed.

The user assumes the full risk of using this standard. In no event shall the EMVA, members of the technical committee, or their companies, be liable for any actual, direct, indirect, punitive, or consequential damages arising from such use, even if advised of the possibility of such damages.

2 GenDC Container

2.1 GenDC Container Layout

2.1.1 GenDC Container General Layout

A GenDC Container is a self-described object that holds simple or complex arbitrary data buffers. The basic elements of a Container are the Descriptor and the data section (see Figure 2-1: GenDC Container Descriptor and Data). The Descriptor groups all the Headers describing the Container and the Components (including its Part(s)). In detail, the Descriptor has a Container Header followed by one or more Component Headers where each Component contains a single or multiple Part Headers. The data section consists of all the Part's data.

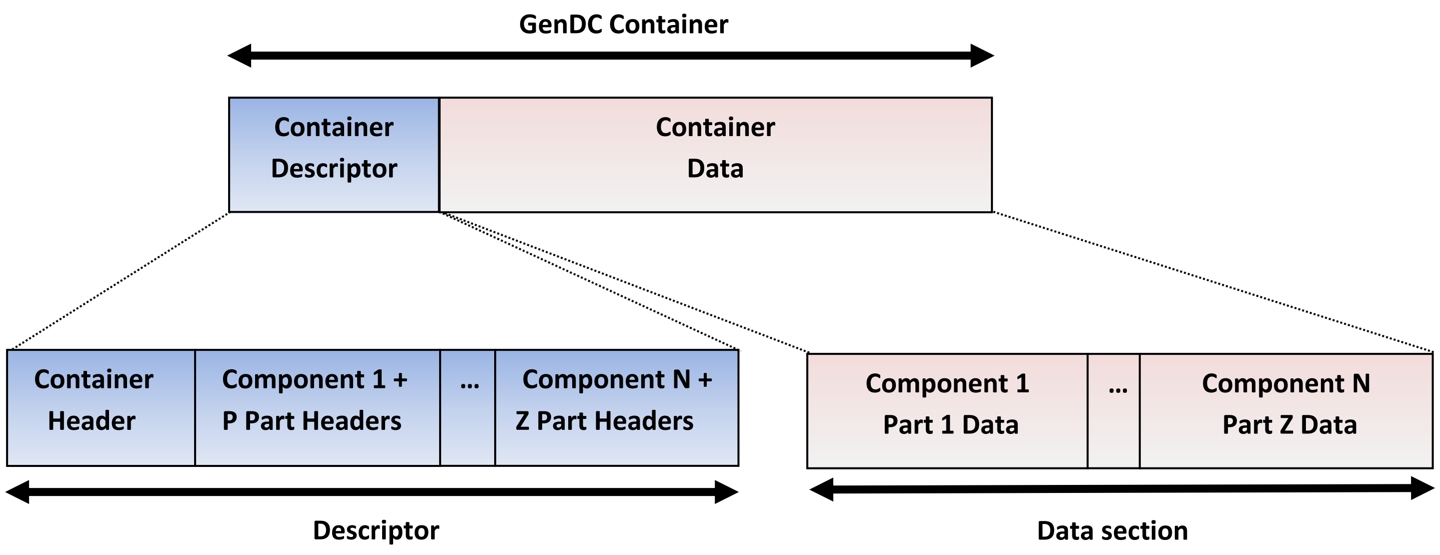

Figure 2-1: GenDC Container Descriptor and Data

* N = Number of Components in the Container, P = Number of Parts in the Component 1, Z = Number of Parts in the Component N.

A Container always starts with a unique, single Descriptor located first in the Container. However, to describe various Container transmission scenarios and allow preprocessing, this specification defines three kinds of Descriptors that the Transport Layers can use. All of them share the same layout but are typically sent at different points in time. The **prefetch** Descriptor is available before the streaming starts either from the XML or from the bootstrap registers of the transmitting device and describes all possible Components and Parts that can be received in the following acquisition phase. A Descriptor has to be sent as early as feasible during the acquisition in order to allow preprocessing of the data during the reception of the data. If the Container cannot vary during transmission, this Descriptor is called the **final** Descriptor giving complete and definitive information on the upcoming data. If the Container can vary during transmission a **preliminary** Descriptor should be sent first at the start of the transmission to allow preprocessing and must indicate which fields of the Component and Part Headers might change during transmission. In that case, a **final** Descriptor is sent at the end of the transmission and contains the final description of the Container's data. It shares the offset with the **preliminary** Descriptor and is therefore suitable to overwrite it.

2.1.2 GenDC Container Headers Hierarchy

The Container Descriptor includes a Container Header that points to one or more Component Headers where each Component Header points to its constituting Part Header(s). Each of those Part Headers then points to their corresponding Part's data section.

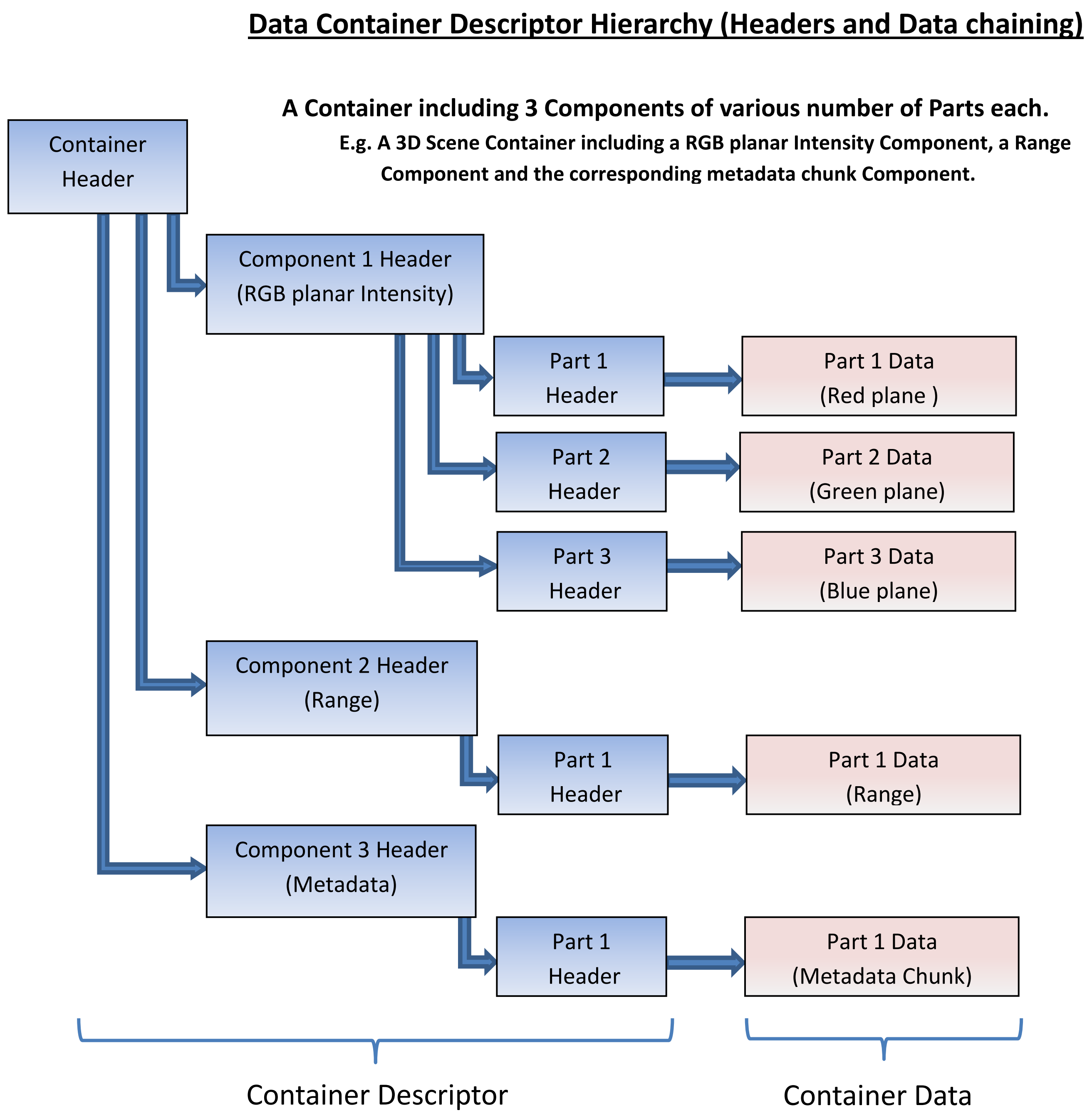
Data Container Descriptor Hierarchy (Headers and Data chaining)
A Container including 3 Components of various number of Parts each.
E.g. A 3D Scene Container including a RGB planar Intensity Component, a Range Component and the corresponding metadata chunk Component.
Container Header
Component 1 Header (RGB planar Intensity)
Part 1 Header
Part 1 Data (Red plane)
Part 2 Header
Part 2 Data (Green plane)
Part 3 Header
Part 3 Data (Blue plane)
Component 2 Header (Range)
Part 1 Header
Part 1 Data (Range)
Component 3 Header (Metadata)
Part 1 Header
Part 1 Data (Metadata Chunk)
Container Descriptor
Container Data

Figure 2-2: GenDC Container Descriptor hierarchy in a multi-Components scenario (headers and data chaining)
(One Container of three Components with different number of data Parts)

2.2 GenDC Headers

The following subsections describe the Headers in detail. In the Header layout representations, there are fields that are common to all Containers or Components (grey background), fields that are common to all Parts (white background) and Part specific fields (light blue background). For each type of Header, a general layout is presented first giving an overview of the fields' names and placement (64-bit per line) followed by a detailed description of each Header's fields.

2.2.1 GenDC Container Header Layout

Signature = "GNDC"				Version Major	Version Minor	Version Sub Minor	Reserved
HeaderType		Flags		HeaderSize			
Id							
VariableFields		Reserved					
DataSize							
DataOffset							
DescriptorSize				ComponentCount			
ComponentOffset[ComponentCount] …							

Container

Figure 2-3: GenDC Container Header Layout

2.2.2 GenDC Container Header Description

Width (Bytes)	Offset (Bytes)	Description
4	0	**Signature** = "GNDC" Unique signature identifying a GenDC Container: a FourCC code encoded as 4 ASCII characters not null terminated (GDC_SIGNATURE = "GNDC" = 0x43444E47).
3	4	**Version** = Container Descriptor version coded as Major.Minor.SubMinor. (e.g. GDC_VERSION => 01.00.00 = 0x01,0x00, 0x00 as three consecutive 8 bit fields). The version corresponds to the GenDC specification that the Container complies to. The following versioning rules apply: • The layout of the first 8 bytes of the Container Header will never change in a way that the binary compatibility is broken. • The major version is incremented if the layout of Headers changes or major rules change, that break the compatibility. • The minor version is incremented if new Components, Parts or flags are added that need to be interpreted. • The sub minor version is incremented for clarifications to the standard document. • If an implementation supporting a major version of the specification (e.g. 1.0.0), receives a Container with the same major version (e.g. 1.x.x) but a higher minor and/or sub minor version (e.g. 1.1.2), it is guaranteed to be able to interpret the known (1.0.0) Components and Parts.
1	7	**Reserved** Reserved for future use, set to 0.
2	8	**HeaderType** = Unique Header format identifier (Container Header) (GDC_CONTAINER_HEADER = 0x1000). A GenDC Container must always start with a Container Header.

2	10	**Flags** Flags specifying the characteristics and format of the Container. See section 4 for more information.

Width (bits)	Bit offset (lsb << x)	Description
1	0	**TimestampPTP** If true, the timestamps of the Components are relative to the epoch January 1, 1970 00:00:00 (TAI) like in the PTP IEEE-1588 format.
1	1	**ComponentInvalid** If True, Components in the Container might be invalid. The **Invalid** flag of each Component must be checked before using it. An example use case is a device with a static ComponentCount but leaving out Components on-the-fly if they cannot be generated.
14	2	**Reserved** (set to 0)

4	12	**HeaderSize** = Size of the Container Header. Size of the Container Header in bytes including the variable sized ComponentOffset array.
8	16	**Id** Container identifier. Strictly monotonically incrementing by 1 with each transmitted Container. Note: Start value and reset condition can be specified by the Transport Layer Protocols.

2	24	**VariableFields** • Flags specifying which type of data information might vary during the Container reception. These flags indicate fields in the Container containing preliminary information, such as images acquired from variable scan or trigger controlled acquisition, where the final size or some other information is not fully known until the end of the acquisition. In general these flags are used to allow processing before the complete Container has been received. • If any of these flags is set, the Container Descriptor is **preliminary** or **prefetched** and might change. An updated **final** Descriptor must be received before the Descriptor can be fully and definitively interpreted. The preliminary Descriptor can be used to start preprocessing of the Container's data. • In the final Descriptor all the VariableFields flags must be set to 0.

Width (bits)	Bit offset (lsb << x)	Description
1	0	**DataSize** If True, the DataSize of the Components' Parts might become smaller. Note that it is not allowed to become larger because the preliminary DataSize gives an upper limit.
1	1	**SizeX** If True, the SizeX and PaddingX of the Components' Parts might change.
1	2	**SizeY** If True, the SizeY and PaddingY of the Components' Parts might change.
1	3	**RegionOffset** If True, the RegionOffsetX and RegionOffsetY of the Components might change.
1	4	**Format** If True, the Components' Format might change. This bit can only be set in a prefetch Descriptor.
1	5	**Timestamp** If True, the Timestamp of the Components might change.

		VariableFields (continued) Width (bits) | Bit offset (lsb << x) | Description 1 | 6 | **ComponentCount** — If True, some Components might not be transmitted. The omitted Components must be the last ones. 1 | 7 | **ComponentInvalid** — If True, the **Invalid** flag of certain Components of the Container might change. 8 | 8 | **Reserved** (set to 0).
6	26	**Reserved** Reserved for future use (set to 0).
8	32	**DataSize** Maximum size of the entire Container's data section in bytes. Note: Does not include the size of the Container Descriptor.
8	40	**DataOffset** Offset in bytes of the start of the entire Container's data section relative to the start of the Container's Descriptor.
4	48	**DescriptorSize** Size of the Container Descriptor in bytes. This represents the cumulative size of the Container Header, all the Component Headers and their Part Headers.
4	52	**ComponentCount** Number of Components in the entire Container.
Component Count x8	56	**ComponentOffset[]** Array of the offsets in bytes of the start of each of the Component Headers relative to the start of the Container's Descriptor. The size of the array is **ComponentCount** x 8 bytes.

Nested VariableFields table:

Width (bits)	Bit offset (lsb << x)	Description
1	6	**ComponentCount** If True, some Components might not be transmitted. The omitted Components must be the last ones.
1	7	**ComponentInvalid** If True, the **Invalid** flag of certain Components of the Container might change.
8	8	**Reserved** (set to 0).

Table 2-1: GenDC Container Header Description

2.2.3 GenDC Component Header Layout

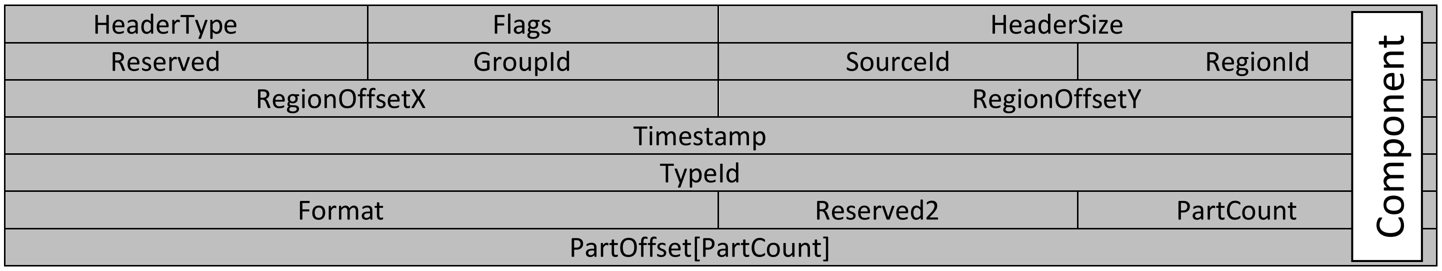

Figure 2-4: GenDC Component Layout

2.2.4 GenDC Component Header Description

2.2.4.1 GenDC Component Header Common Fields Description

Width (Bytes)	Offset (Bytes)	Description
2	0	**HeaderType** = Unique Header format identifier (Component Header) (GDC_COMPONENT_HEADER = 0x2000). A GenDC Container must always contain at least one Component Header.
2	2	**Flags** Flags specifying the characteristics and format of the Component. Width (bits) | Bit offset (lsb << x) | Description 1 | 0 | **Invalid** The Component is invalid and must not be used. If this flag is set the **ComponentInvalid** flag of the Container Header must be also set. 15 | 1 | Reserved (set to 0)
4	4	**HeaderSize** = Size of the Component Header Size of the Component Header in bytes including the variable sized PartOffset array. Note the Component's Part Headers are **not** included.
2	8	**Reserved** = Reserved for future use (set to 0).

2	10	**GroupId** Group identifier for a Component in the Container. GroupId specifies which Components of a Container are related together (e.g. All the Components of a particular 3D scene would have the same GroupId).
2	12	**SourceId** Identifier of the data source that generated the Component. E.g. If a device has two sensors each sensor can be a source with its own identifier.
2	14	**RegionId** Identifier of the data source's region that generated the Component. Note that processing modules can also have a unique region identifier.
4	16	**RegionOffsetX** X Offset of the region (in pixels).
4	20	**RegionOffsetY** Y Offset of the region (in pixels).
8	24	**Timestamp** Timestamp of the start of the Component's data creation in nanoseconds. The Timestamp value is represented as a 64-bit signed integer (two's complement) with only the positive range defined. If the TimestampPTP flag of the Container Header is set, the Component's Timestamp value must be relative to January 1, 1970 00:00:00 (TAI) like the epoch of the PTP IEEE-1588-2008 format. To transfer time information between a GenDC Timestamp and PTP IEEE 1588-2008, the conversions between the time formats can be done using: $t_{\mathrm{GenDC}} = t_{\mathrm{PTP}Nanoseconds} + 10^9 \times \mathrm{t}_{\mathrm{PTP}Seconds}$ Note: If necessary t_{GenDC} must be saturated to 0x7FFF FFFF FFFF FFFF.
8	32	**TypeId** Component type identifier. This number uniquely identifies what type the Component data represents and is equal to the SFNC's ComponentIdValue feature predefined values.

4	40	**Format** Format of the whole Component (including all its Parts). The value is specified using the standard PFNC's Pixel Format Values list. See the GenICam Pixel Format Naming Convention's "Pixel Format Values" document. If the Component has only a single Part the Component's Format is the same as the Part's Format. For planar formats it is the encapsulating format (e.g. RGB8_planar). For the Components that are not pixel based like Metadata, the PFNC_Data8 format must be used.
2	44	**Reserved2** = Reserved for future use (set to 0).
2	46	**PartCount** Number of Parts in the Component. Note that planar Components must include one separate Part per data plane.
PartCount x 8	48	**PartOffset[]** Array of the offsets in bytes of the start of each of the Part Headers relative to the start of the Containers's Header. The size of the array is **PartCount** x 8 bytes.

Table 2-2: GenDC Component Header fields description

2.2.5 GenDC Part Header Layout

The following figure shows the layout of the Part Headers. In white common fields to all Parts (described in the section 2.2.6.1) and in blue the fields which are specific to the Part's Type.

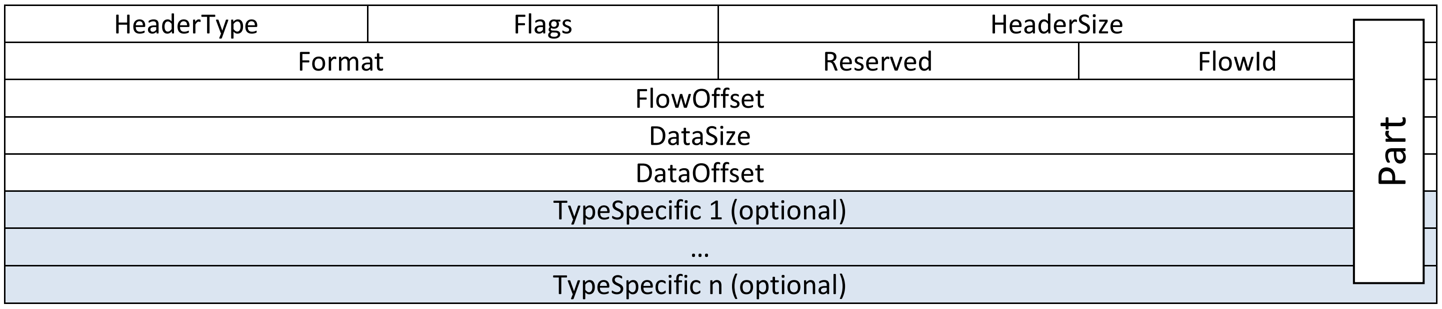

Figure 2-5: GenDC Part Header Layout

Each Part can have multiple 64-bit TypeSpecific fields (shown in blue). For all the Part types defined in this document, these fields are described in the section 2.2.7. In general, if applicable, those TypeSpecific fields should follow a common layout. This common layout illustrated in Figure 2-6, starts with the Parts' dimensions (8 bytes, e.g. for 2D images SizeX and SizeY each 4 bytes), followed by the Parts' padding (4 bytes, e.g. for 2D images PaddingX and PaddingY each 2 bytes) and a reserved field (4 bytes) for future additions. Other optional Part Type specific fields (8 bytes each) can also be present.

Dimension	
Padding	InfoReserved
TypeSpecifc 3 (optional)	
…	
TypeSpecific n (optional)	

Figure 2-6: GenDC Part TypeSpecific fields general layout

2.2.6 GenDC Part Header Description

2.2.6.1 GenDC Part Header Common Fields Description

This table describes the Part's common fields as presented in Figure 2-5: GenDC Part Header Layout.

Width (Bytes)	Offset (Bytes)	Description
2	0	**HeaderType** = Part Type Header format identifier (GDC_PART_HEADER = 0x4xxx). A GenDC Component must contain at least one Part Header. See section 2.2.7 for the table of defined types.
2	2	**Flags** = Part specific flags Width (bits) / Bit offset (lsb << x) / Description: 16 / 0 / Reserved (set to 0).
4	4	**HeaderSize** = Size of the Part type specific Header. Size in bytes of the Part Header including Part Type specific fields.
4	8	**Format** = Data format of the Part. The value is specified using the standard PFNC's Pixel Format Values list. See the GenICam Pixel Format Naming Convention's "Pixel Format Values" document. In general, the Part's Format is identical to its encapsulating Component's Format. For a Component that has a planar format, the Parts of the planar buffer must be ordered according to their encapsulating Component's Format. The Part's Format describes the format of each individual Part/plane of the encapsulating Component (e.g. For a PFNC_RGB8_planar Component, the individual Parts format will be PFNC_R8, PFNC_G8 and PFNC_B8). For Parts that are not pixel based like Metadata, the PFNC_Data8 format must be used.
2	12	**Reserved** = Reserved for future use (set to 0).
2	14	**FlowId** Unique identifier of the data Flow used to transport and store the Part's data. Start at 0 and incrementing. See chapter 3 for more information.

Flags sub-table:

Width (bits)	Bit offset (lsb << x)	Description
16	0	Reserved (set to 0).

8	16	**FlowOffset** Offset in bytes of the Part's data in the data Flow used to transport and store the Part's data relative to the Flows' base address. Note that the Flow base addresses are not part of the Container Descriptor as the Container must be independent of the actual storage location. This is the same as for the Containers' Descriptor base address which is identical to the base address of Flow 0. If FlowId=0 it is therefore the same as DataOffset. For other values of FlowId, DataOffset gives the Flows' start address for linear addressing if FlowOffset=0. This can be used to reconstruct a linear Container e.g. for storage.
8	24	**DataSize** = Size of the Part's data in bytes. Typically the maximum possible data size for a Part. In the final Descriptor the Part's real and valid data size (line scan variable SizeY, compressed image, …).
8	32	**DataOffset** Offset in bytes of the start of the Part's data section relative to the start of the Container's Descriptor for linear addressing.

Table 2-3: GenDC Component Header Part common fields description

2.2.6.2 GenDC Part Header Type Specific Fields Description

This table describes the Part's TypeSpecifc fields as presented in Figure 2-6: GenDC Part TypeSpecific fields general layout. If applicable, those fields should use the following general definition. This permits consistent and easier to interpret Parts headers. In chapter 2.2.8 GenDC Part Header Type Specific Fields you can find Part specific definitions for those generic fields.

Width (Bytes)	Offset (Bytes)	Description
8	40	**Dimension** Typically the size of the Part (e.g. For 2D images SizeX and SizeY. For 1D array SizeX only).
4	48	**Padding** Typically the padding used by the Part (e.g. for 2D images PaddingX and PaddingY. For 1D array PaddingX only).
4	52	**InfoReserved** For future generic Part info use. Set to 0.

8xN	56	**TypeSpecific fields 3 to N (optional)** Optional fields to be used to further describe the Part if necessary. Note that depending on the Type the above 3 fields might not exist. In that case, the TypeSpecific fields will start from Offset 40 instead. This might especially be true for custom formats. It is however highly recommended to use the general layout starting with Dimension and Padding even for custom Parts since generic software can make use of it. For example even if a specific Part Header type is unknown, but it is a 2D image, a generic software can present the data to the user as raw data but with correct SizeX and SizeY.

Table 2-4: GenDC Component Header Part type specific fields description

[R-001] A GenDC compliant product must use the Headers and flags as defined by this specification.

2.2.7 GenDC Part Header Types

This section lists the defined Part Header types.

Value = 0x4000-0x4FFF, 4 bits are used for the general category; the following low 8 bits are used for the sub type of a Part. It is possible to specify a custom category (0x4Fxx), which gives you 256 possible custom sub types. Besides this each standardized category has space for 16 custom sub types (0x4xFx).

Type	Value	Description
GDC_GENERIC_PART_METADATA	0x40xx	Generic Metadata Part type.
GDC_METADATA_GENICAM_CHUNK	0x4000	GenICam Chunk metadata. Binary chunk metadata formatted as specified in the GenICam standard. Note: Only a Component of type GDC_METADATA can have a Part of type GDC_METADATA_....
GDC_METADATA_CUSTOM(x)	0x40Fx	The metadata Part data is custom and does not correspond to a known Type. It is uniquely identified by the lower 4 bits of this field.
GDC_GENERIC_PART_1D	0x41xx	Generic 1D Part type.
GDC_1D	0x4100	1D array (such as 3D Point Cloud).
GDC_1D_CUSTOM(x)	0x41Fx	The 1D Part data is custom and does not correspond to a known Type. It is uniquely identified by the lower 4 bits of this field.
GDC_GENERIC_PART_2D	0x42xx	Generic 2D Part type.
GDC_2D	0x4200	Rectangular uncompressed image (monochrome or none planar color).
GDC_2D_JPEG	0x4201	JPEG compressed Image.
GDC_2D_JPEG2000	0x4202	JPEG 2000 compressed Image.
GDC_2D_H264	0x4203	H.264 compressed Image.
GDC_2D_CUSTOM(x)	0x42Fx	The 2D Part data is custom and does not correspond to a known Type. It is uniquely identified by the lower 4 bits of this field.
…	…	…
GDC_CUSTOM(xx)	0x4Fxx	The Part data is custom and does not correspond to a known category. It is uniquely identified by the lower 8 bits of this field.

Table 2-5: GenDC Part Types

2.2.8 GenDC Part Header Type Specific Fields

This section lists the Part Type specific Header fields.

2.2.8.1 1D Array and Metadata specific Part Header fields

Size		
Padding	PaddingReserved	InfoReserved
InfoTypeSpecific (Reserved)		

Figure 2-7: Part specific Header fields layout for 1D Array or Metadata

Note: For Part Type = 1D array Points Cloud, the Component's Format and Part's Format are defined by a PFNC value.
For Part Type = 1D array Metadata, the Component's Format and Part's Format are set to the Data8 PFNC value.

1D Array (Point Cloud or Metadata) Part Type specific Header fields		
Width (Bytes)	Offset (Bytes)	Description
8	40	**Size** Size of the 1D Part (in number of elements according to Part Format).
2	48	**Padding** Size of the padding at the end of the Part (in bytes).
2	50	**PaddingReserved = 0.** Reserved for alignment and future use.
4	52	**InfoReserved = 0** Reserved for future use.
8	56	**InfoTypeSpecific = 0 or GenICam Chunk Layout Id**. Reserved for future use (0) or set to the Chunk Layout Id for Part of type M_METADATA_GENICAM_CHUNK.

Table 2-6: Part specific Header fields description for 1D Array and Metadata

2.2.8.2 2D uncompressed, JPEG or JPEG2000 compressed image specific Part type fields

SizeX		SizeY
PaddingX	PaddingY	InfoReserved

Figure 2-8: Part specific Header fields layout for 2D uncompressed, JPEG or JPEG2000 compressed image

Width (Bytes)	Offset (Bytes)	Description
4	40	**SizeX** X size of the 2D Part (in pixels).
4	44	**SizeY** Y size of the 2D Part (in pixels).
2	48	**PaddingX** Size of the X padding at the end of each line (in bytes).
2	50	**PaddingY** Size of the Y padding at the end of the Part (in bytes). Padding Y can be used to align the following Part to specific hardware constraints, e.g. processor specific alignment constraints
4	52	**InfoReserved = 0** Reserved for future use.

Table 2-7: Part specific Header fields description for 2D uncompressed, JPEG or JPEG2000 compressed image

2.2.8.3 H.264 compressed image specific Part Header fields

SizeX						SizeY	
PaddingX		PaddingY				InfoReserved	
Reserved	ProfileIDC	CS	PM	RF	LevelIDC	SpropInterleavingDepth	SpropMaxDonDiff
SpropDeintBufReq						SpropInitBufTime	

Figure 2-9: Part specific Header fields layout for H.264 compressed image

2D H.264 Compressed Part Type specific Header fields

Width (Bytes)	Offset (Bytes)	Description
4	40	**SizeX** X size of the 2D Part (in pixels).
4	44	**SizeY** Y size of the 2D Part (in pixels).
2	48	**PaddingX** Size of the X padding at the end of each line (in bytes).
2	50	**PaddingY** Size of the Y padding at the end of the Part (in bytes).
4	52	**InfoReserved = 0** Reserved for future use.
1	56	**Reserved = 0** Reserved for future use.
1	57	**ProfileIDC** Profile IDC sequence parameter set data attribute as defined by H.264.

1	58	**H264Flags** Flags specifying the characteristics and format of the H.264 Part. Width (bits) / Bit offset (lsb << x) / Description: 4 / 0 / **CS** — Constraint set0_flag, set1_flag, set2_flag, set3_flag sequence parameter set data attribute as defined by H.264. 2 / 4 / **PM** (Packetization Mode) — This parameter signals the packetization properties of the H.264 Part. When the value of packetization-mode is equal to zero, the single NAL mode, as defined in section 6.2 of RFC3984, is used. When the value of packetization-mode is equal to one, the non-interleaved mode, as defined in section 6.3 of RFC3984, is used. When the value of packetization-mode is equal to two, the interleaved mode, as defined in section 6.4 of RFC3984, is used. Other values are reserved. 2 / 6 / **RF** (Reserved Flags. Set to 0).
1	59	**LevelIDC** level_idc sequence parameter set data as defined by H.264.
2	60	**SpropInterleavingDepth** This parameter may be used to signal the properties of a NAL unit stream. This parameter is not applicable if the value of the packetization mode is equal to zero or one. Otherwise, it specifies the maximum number of VCL NAL units that precede any VCL NAL unit in the NAL unit stream in transmission order and follow the VCL NAL unit in decoding order as per RFC3984.
2	62	**SpropMaxDonDiff** This parameter may be used to signal the properties of a NAL unit stream. This parameter is not applicable if the value of the packetization mode is equal to zero or one. Otherwise, SpropMaxDonDiff is an integer in the range of 0 to 32767 and it is used in the NAL unit de-interleaving process as per RFC3984.
4	64	**SpropDeintBufReq** This parameter may be used to signal the properties of a NAL unit stream. This parameter is not applicable if the value of the packetization mode is equal to zero or one. Otherwise, SpropDeintBufReq signals the required size of the de-interleaving buffer for the NAL unit stream as per RFC3984.

Nested table in H264Flags:

Width (bits)	Bit offset (lsb << x)	Description
4	0	**CS** Constraint set0_flag, set1_flag, set2_flag, set3_flag sequence parameter set data attribute as defined by H.264.
2	4	**PM** (Packetization Mode) This parameter signals the packetization properties of the H.264 Part. When the value of packetization-mode is equal to zero, the single NAL mode, as defined in section 6.2 of RFC3984, is used. When the value of packetization-mode is equal to one, the non-interleaved mode, as defined in section 6.3 of RFC3984, is used. When the value of packetization-mode is equal to two, the interleaved mode, as defined in section 6.4 of RFC3984, is used. Other values are reserved.
2	6	**RF** (Reserved Flags. Set to 0).

4	68	**SpropInitBufTime** This parameter may be used to signal the properties of a NAL unit stream. This parameter is not applicable if the value of the packetization mode is equal to zero or one. Otherwise, the parameter signals the initial buffering time that a receiver must buffer before starting decoding to recover the NAL unit decoding order from the transmission order as per RFC3984.

Table 2-8: Part specific Header fields description for H.264 compressed image

[R-002] A GenDC compliant product must use the Part types as defined by this specification.

3 GenDC and Transport Layers

GenDC is agnostic to the Transport Layer and independent of it. To accomplish this, the notion of Flows is introduced as intermediate layer to decouple the Container content description from how it is transported and stored in receivers' memory.

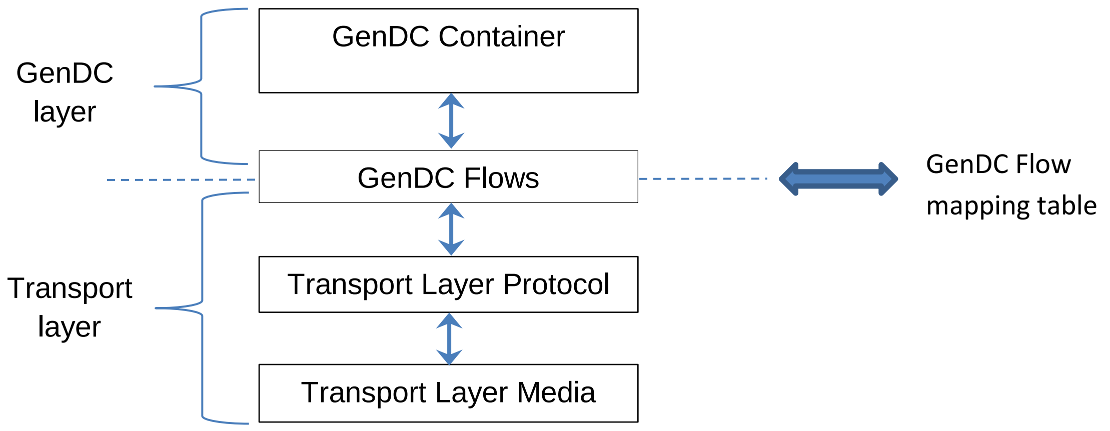

Figure 3-1: GenDC layer and Transport Layer coupled by GenDC Flows

The top layer defines a GenDC Data Container and is the generic representation of a possibly complex data buffer to transmit/receive or that resides in memory. It is made of a group of standardized Headers called a Descriptor that gives the information about the Container itself, the Components and their individual Parts and describes the Container's data. This Container is "what" the device/system needs to generate, transmit, receive, store or manipulate the data. The Container definition and layout is self-described and independent of "how" it can be transported. For transmission the Container must use a linear memory layout. Example: a GenDC Container for 3D multi-Components image data.

GenDC Flows adapt the top and bottom layer. Both layers need to be aware of the Flows. GenDC describes the Flows and the Transport Layer provides means to transfer them. This way, it is possible to have a fully GenDC agnostic Transport Layer which can transport Flows even in parallel without further knowledge of the GenDC Container. The Transport Layer only needs to know how to transport and store Flows. To accomplish this, it is necessary to have a Flow table from the upper layer just giving the FlowID and FlowSize for each Flow in addition to a matching table of Flow base addresses which is given by the user or allocated by the Transport Layer itself. Example: Transfer of the GenDC Container sequential to one base address in one Flow.

The bottom layer is the Transport Layer (*not in the scope of GenDC*). It transports the GenDC Container as defined in the transport layer protocol based on the transport layer media. It typically takes care of data consistency and any other transport media related mechanism necessary for efficient and reliable data transmission. Example: GigE Vision protocol packets that transmit the GenDC Data Container on Ethernet.

3.1 GenDC Typical Transmission and Reception

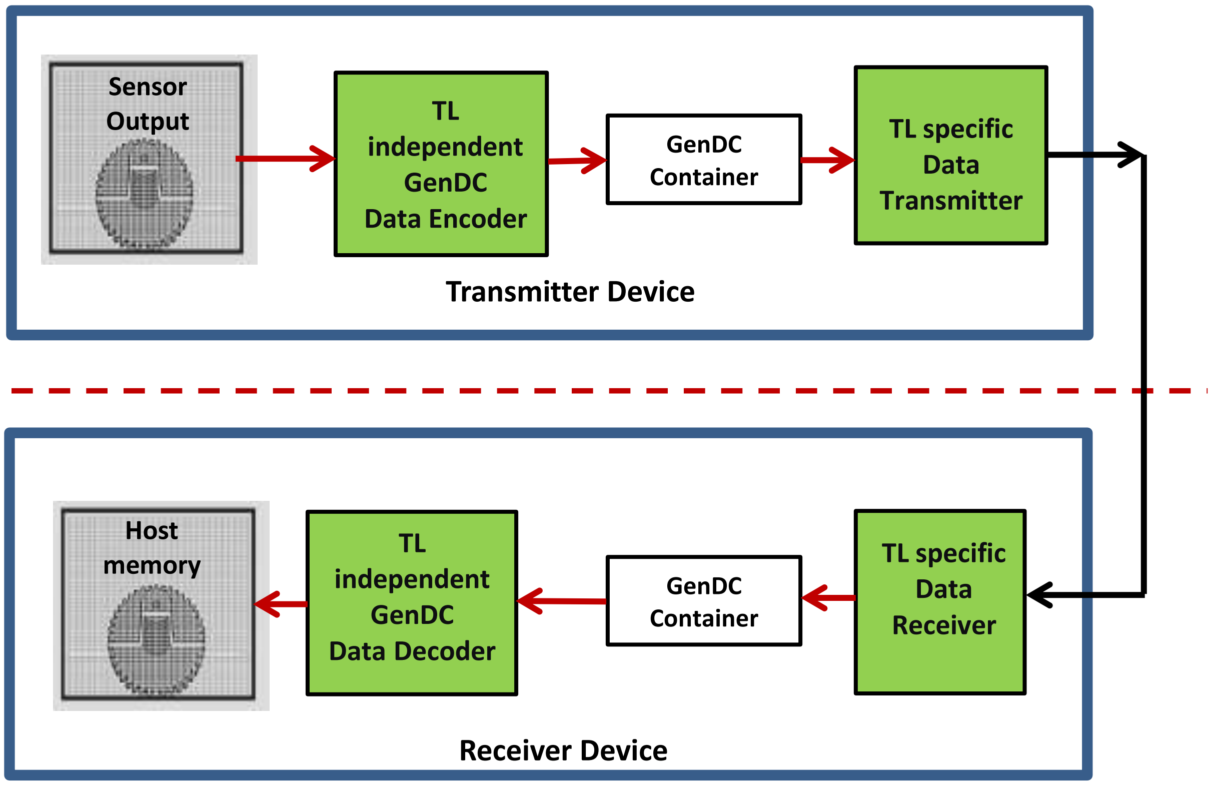
Sensor Output
TL independent GenDC Data Encoder
GenDC Container
TL specific Data Transmitter
Transmitter Device
Host memory
TL independent GenDC Data Decoder
GenDC Container
TL specific Data Receiver
Receiver Device

Figure 3-2: GenDC typical Transmission and Reception data handling

On the transmitter side, the Descriptor is typically encoded first and then passed with the Container's data to a transport layer specific data transmitter to be streamed out (possibly out of order). The receiver typically handles the GenDC Container by a transport layer specific data receiver. The assembled Container is then passed to a GenDC decoder to be interpreted.

3.2 GenDC Flows

Flows allow the TL specific data transmitter and receiver to work without knowledge of the GenDC Container. Each Flow represents an independent memory transfer with a given size and identification number starting from zero. It is possible to transport the GenDC Container in a single Flow or in multiple Flows. In general, a Flow mapping table as shown in section 3.2.1 is sufficient for a Transport Layer to handle a GenDC Container. The receiver simply allocates buffer space for each Flow and does not need to know what is transported inside.

Arbitrary Parts of a GenDC Container can be transported in parallel to separate memory locations by using GenDC Flows. The relation between the GenDC Container, GenDC Flows and an arbitrary Transport Layer is shown in Figure 3-1. GenDC Flows allow the transport layer to reserve the necessary buffer space. In case the transport layer provides multiple Flows this also supports parallelism. Note that certain Transport Layer, Transmitter or Receiver might have limitations regarding parallelism.

The transfer of a GenDC Descriptor is always done in Flow zero. The GenDC Containers' Part(s) Header provides information about the FlowId and FlowOffset for all individual Parts. This optionally allows sending Parts on other Flows than Flow zero. Besides this, the GenDC Container is agnostic to Flows.

3.2.1 GenDC Flow mapping table

The Transport Layer gets the Flow mapping table that provides the Flow information about the number of Flows and the size of each Flow in a Transport Layer specific way, for example using the device XML or bootstrap registers. The Flow information is static after TLParamsLocked has been set.

HeaderType		Flags	HeaderSize
VersionMajor	VersionMinor	Reserved	FlowCount
FlowSize[FlowCount] …			

Figure 3-3: GenDC Flow mapping table layout

Width (Bytes)	Offset (Bytes)	Description
2	0	**HeaderType** = Unique Header format identifier (Flow Mapping Table Header) (GDC_FLOW_MAPPING_HEADER = 0x7000).
2	2	**Flags** = Flags specifying the characteristics and format of the Table. Width (bits) | Bit offset (lsb << x) | Description 16 | 0 | Reserved (set to 0)
4	4	**HeaderSize** = Size of the Flow Mapping Table Size of the Flow Mapping Table Header in bytes including the variable sized FlowSize array (i.e.16+ FlowCount x 8)
1	8	**VersionMajor** = Major Version of this table. Must be set to 1 for this GenDC specification.
1	9	**VersionMinor** = Minor Version of this table. Must be set to 0 for this GenDC specification.
2	10	**Reserved** = Reserved for future use (set to 0).
4	12	**FlowCount** = Number of entries (=Flows) in the table

FlowCount x 8	16	**FlowSize[]** Array of the size in bytes of each Flow. The size of the array is **FlowCount** x 8 bytes.

Table 3-1: GenDC Flow mapping table description

The GenDC Container is transmitted using linear addressing relative to the Container start. The Transport Layer can keep this and assign the Flow base addresses also in a linear way or it can use any other Flow base addresses for example putting the data to the memory of a graphics card and keeping the Descriptor in normal Host memory. For this, the internal offsets of the GenDC Data Container do not need to be altered; all is done by changing the Flow base addresses which are not part of the Container.

Early processing of Parts transmitted in different Flows can be easily done if the Transport Layer provides an end of Flow notification of some kind. It is highly recommended for each Transport Layer to provide such an end of Flow notification.

[R-003] A GenDC compliant transmitter must provide a Flow mapping table.

4 GenDC Container formatting, requirements and recommendations

Since the GenDC format is very flexible, it might be suitable to restrict the number of scenarios currently supported in order to simplify and make its usage uniform without restricting its expandability and future usage. This chapter exposes some rules and recommendations that standardize and simplify GenDC encoding, transmission and decoding.

Note that none of the possible rules and recommendations below is dictated by the GenDC format itself which aims to be general and flexible. The generic GenDC format targets to be usable in many different scenarios and even outside of the Transmitter to Receiver transmission context.

Each individual Transport Layer Protocol (TLP) supporting GenDC is also free to add its own particular rules and restrictions regarding the GenDC Container transport to simplify the GenDC usage in its particular context. Recommendations can be ignored if necessary but the TLP must not modify or override basic formal requirements stated in the GenDC standard.

4.1.1 Requirements and recommendations

This section describes requirements and recommendations to be compliant with the GenDC specification:

The GenDC Containers are always stored in little Endian. This permits to have a unified way to encode and decode a GenDC Container Descriptor and data independently of the CPU, Transport Layer Protocol or context where it is used.

[R-004] The GenDC Container Descriptor must be always stored in little-endian ordering.

[R-005] Container data Part in PFNC format must use little-endian ordering.

GenDC Container format and ordering: In the context of data exchange between a Transmitter and a Receiver, a virtual GenDC Container is always represented as a continuous block starting with a GenDC Descriptor immediately followed by a continuous Container's data section. The DataOffset field represents the offset of the data from the Descriptor start.

[R-006] For Transmission, a GenDC Container is always represented as a continuous block of linear memory starting with the Descriptor.

[R-007] The Part's DataOffset is always the offset of the data in bytes from the start of the Descriptor.

GenDC Container transmission Flow(s): In the context of data exchange between a Transmitter and a Receiver, the data section(s) of a virtual GenDC Container are transmitted using one or more data Flow(s). Flows allow splitting a Container in sections to facilitate parallel transmission. Flows have a FlowId that is numbered sequentially starting with 0. A Flow can contain a Container Descriptor and/or one or more data Parts. The GenDC Descriptor is always sent as early as possible in Flow 0. The Container Descriptor should be sent as soon as possible to provide its information early to the receiver and help decoding and preprocessing of the data. A Flow can carry one or many data Parts but a Part must only be mapped to a single Flow. All the Component's Parts sharing a Flow are mapped in the same order as they are listed in the GenDC Descriptor. Due the support of variable data content and hardware based memory alignment constraints a Container can contain unused memory areas. It is up to the receiver to store the data sections of a Container transported on different Flows separately in a single or multiple target buffers after the transport.

[R-008] The Part's FlowOffset is always the offset of the data in bytes from the start of the Flow specified by the Part's FlowId.

[R-009] The Descriptor is always transferred in Flow 0.

[R-010] A Part must only be mapped to a single Flow.

[R-011] Flow must be numbered sequentially starting from 0.

GenDC variable Container Transmission: In the context of data exchange between a Transmitter and a Receiver, if a GenDC Container has some elements that are variable (e.g. Component's Size Y, Region X,Y Offset, Data Size Timestamps, Sequence length…), the corresponding flag(s) of the "VariableFields" field of the Container Header must be set. If the "VariableFields" field is not null, a preliminary Descriptor should be sent to describe the characteristics of the maximum Container that will be transmitted. This preliminary Descriptor must represent the maximum size, number of Components and number of Parts that can be sent. Also, for variable Container, a final GenDC Descriptor including updated Container information must be sent immediately after transmission of the Container's data section.

[CR-012] If a Container has variable content during the transmission the VariableFields flags of the Container must be set accordingly.

[CR-013] If any of the Container's VariableFields flags is set, a final Descriptor must be sent as soon as possible but at least just after the transmission of the data section.

GenDC Container Storage: A GenDC Container can be stored on external storage (e.g. as a file). If stored in GenDC binary format, it must use the extension ".gendc" for the filename.

[R-014] When storing a GenDC Container using the ".gendc" file extension, the standard GenDC binary format must be used.

[R-015] When storing a GenDC Container, a linear Container must be used.

GenDC Container Metadata: Metadata related to a Container must be added to the Container as an additional and separate Metadata Component (e.g. A Container including the Components of a 3D scene (Range, Confidence, …) with an additional GenICam Chunk Metadata Component to store the 3D scene additional information).

[CR-016] If Metadata is added to a Container, this must be done using a separate Metadata Component.

5 Summary

The definition of a Generic Data Container in the context of a GenDC GenICam module permits to standardize the format of the data buffers transmitted and received by the various present and future Transport Layer Protocol (TLP) standards. Those Transport Layers Protocols are then able to concentrate their effort on implementing efficient, reliable and real-time data transmission over a particular physical media. They are no more affected by the data representation changes and new payload types can be added without TLP specification modifications. The GenDC Data Container is defined and standardized only once by a group of representatives of each TLP. The basic principle of GenDC is to define a layer that decouples “what” to transport from “how” to transport it using a particular media.

Also, since the GenDC Container specification defines a self-described data format, it defines at the same time, a standard way to represent almost any imaging related data information and can be used in many different contexts outside of the data exchange between a Transmitter and a Receiver.

Appendix A GenDC Container Structure overview

In this section, a few examples of Container organization are represented. For a more exhaustive list of the Container format for the typical scenarios, see Appendix B.

A.1 GenDC Container typical layout for monochrome or color packed 2D images.

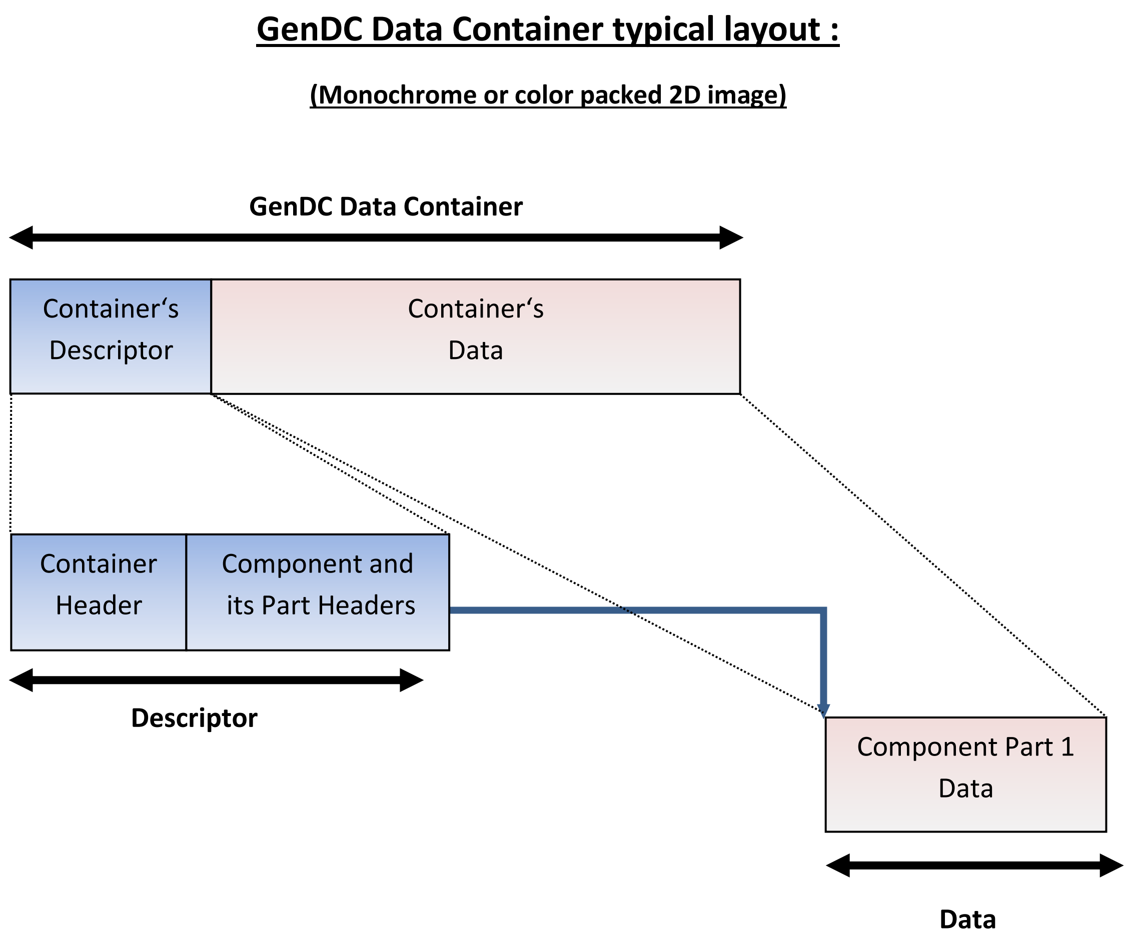

Figure 5-1: GenDC Container's typical layout for a monochrome or color packed 2D image

A.2 GenDC Container typical layout for color planar 2D images.

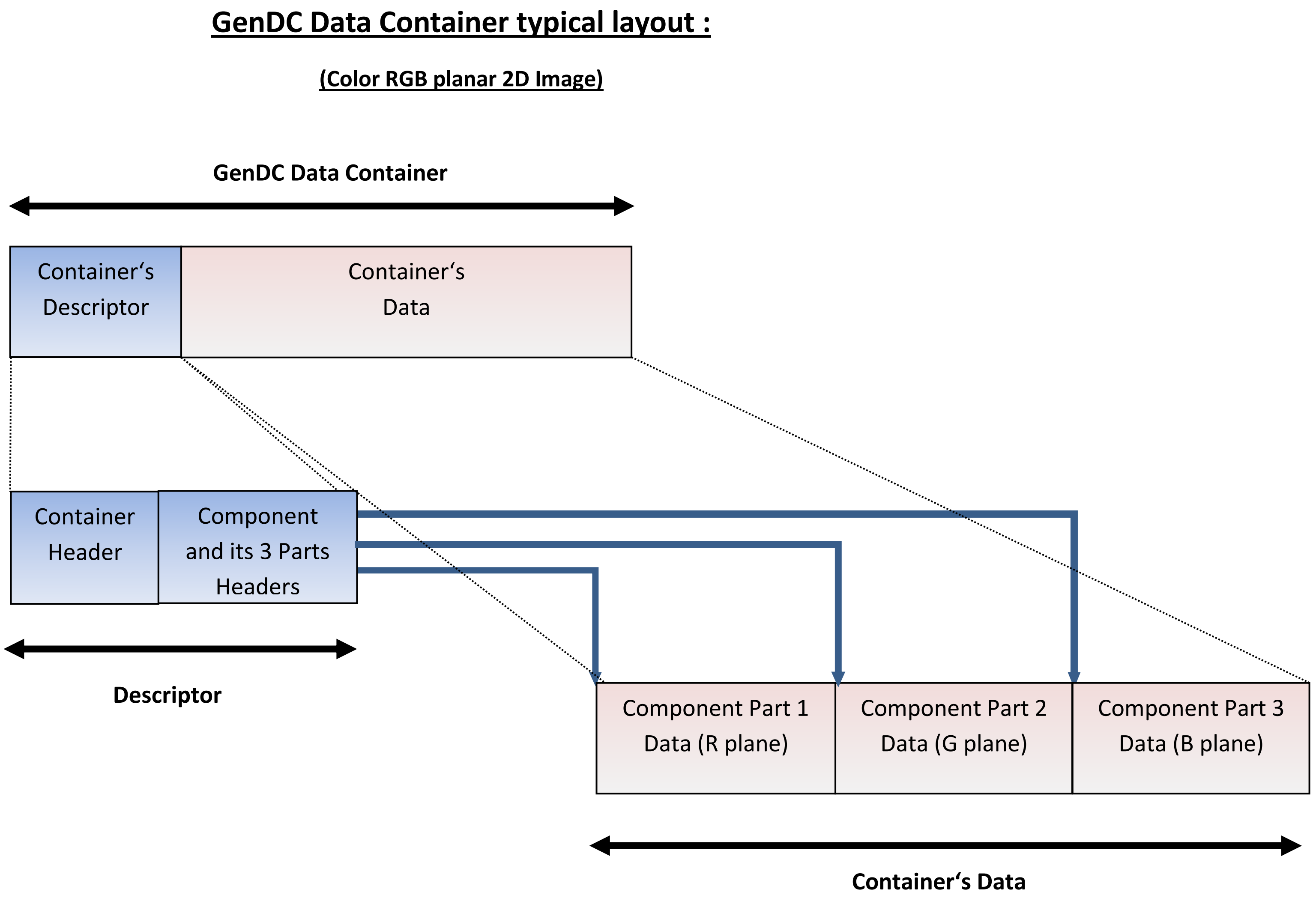
GenDC Data Container typical layout :
(Color RGB planar 2D Image)
GenDC Data Container
Container's Descriptor
Container's Data
Container Header
Component and its 3 Parts Headers
Descriptor
Component Part 1 Data (R plane)
Component Part 2 Data (G plane)
Component Part 3 Data (B plane)
Container's Data

Figure 5-2: GenDC Container's typical layout for a RGB planar 2D image

A.3 GenDC Container typical layout for a multi-Components 3D scene.

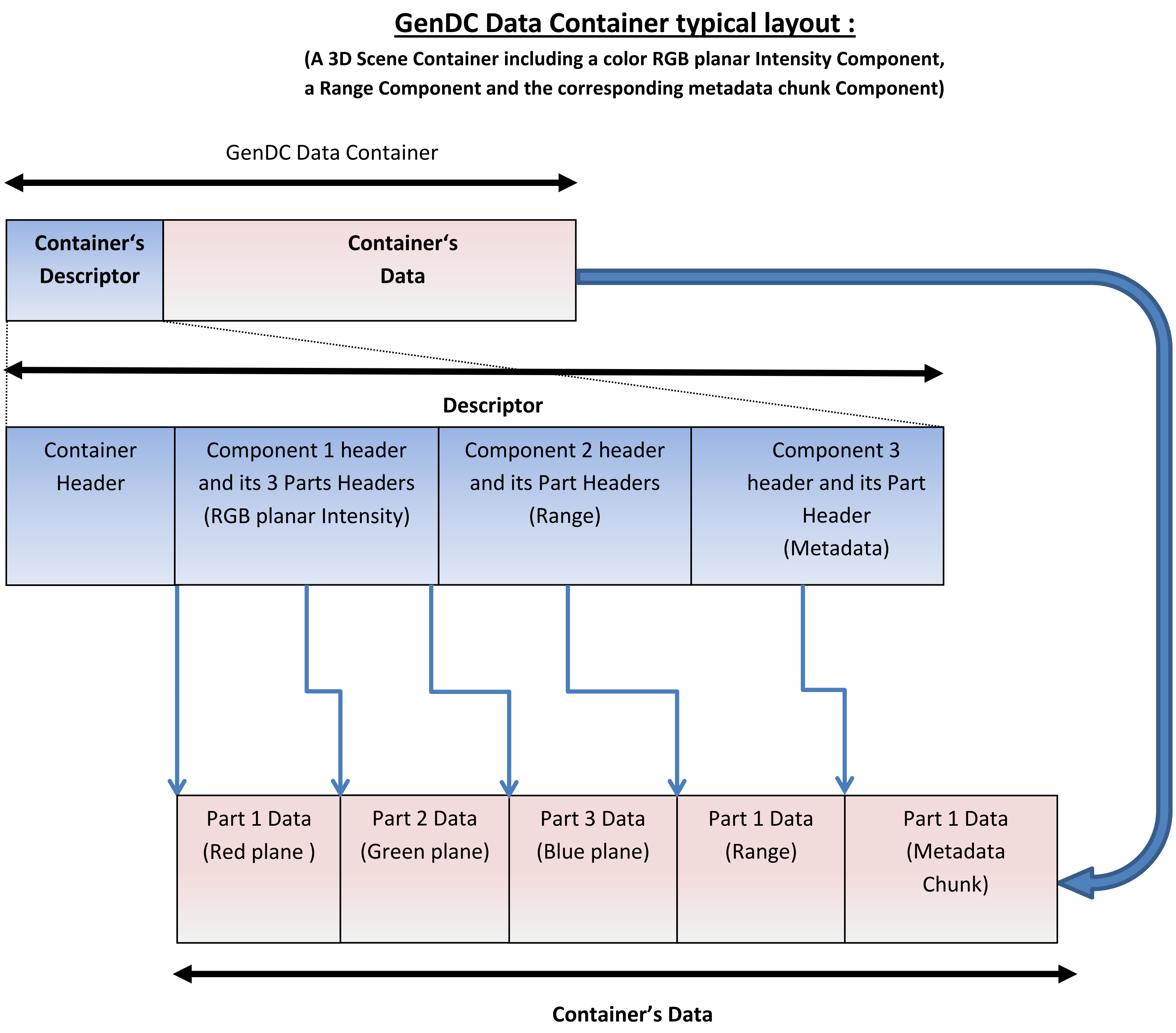
GenDC Data Container typical layout :
(A 3D Scene Container including a color RGB planar Intensity Component,
a Range Component and the corresponding metadata chunk Component)
GenDC Data Container
Container's Descriptor
Container's Data
Descriptor
Container Header
Component 1 header and its 3 Parts Headers (RGB planar Intensity)
Component 2 header and its Part Headers (Range)
Component 3 header and its Part Header (Metadata)
Part 1 Data (Red plane)
Part 2 Data (Green plane)
Part 3 Data (Blue plane)
Part 1 Data (Range)
Part 1 Data (Metadata Chunk)
Container's Data

Figure 5-3: GenDC Container's typical layout for a multi-Components 3D scene with Intensity, Range and Metadata.
(One Container of three Components with different number of data Parts)

Appendix B Typical GenDC Containers layout

This section contains various examples of GenDC data Container layouts that can be used to represent typical Imaging data payloads.

B.1 1D Array

1D Array Container including one Component of one data Part made of 32 bit integers.

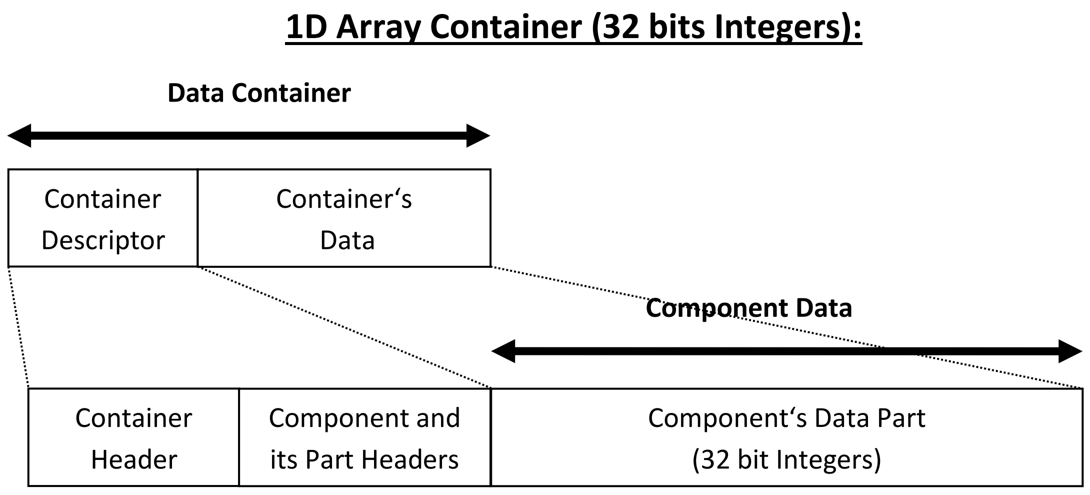
1D Array Container (32 bits Integers):
Data Container
Container Descriptor
Container's Data
Component Data
Container Header
Component and its Part Headers
Component's Data Part (32 bit Integers)

Figure 5-4: 1D Array (32 bit Integers)

B.2 2D Image (monochrome)

2D Image Container including one monochrome Intensity Component of one data Part.

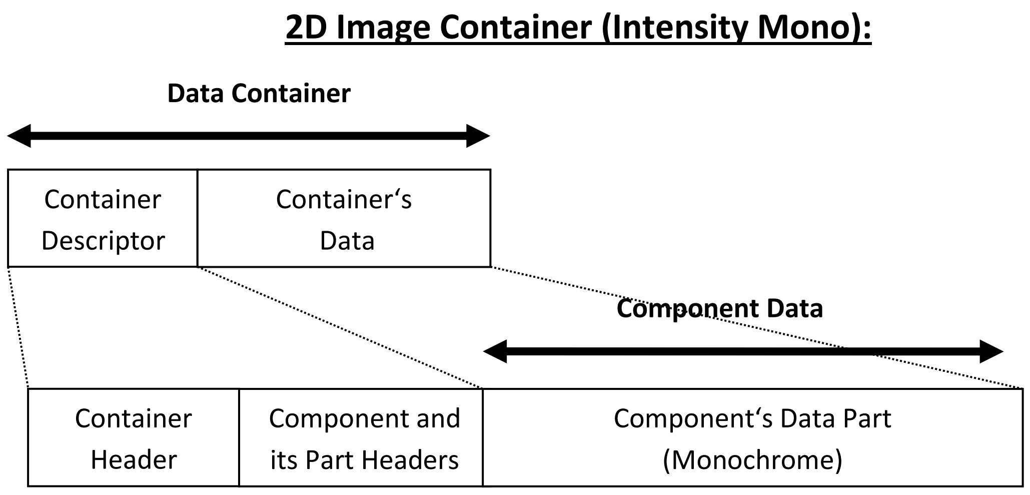
2D Image Container (Intensity Mono):
Data Container
Container Descriptor
Container's Data
Component Data
Container Header
Component and its Part Headers
Component's Data Part (Monochrome)

Figure 5-5: 2D Image (monochrome 8 bit)

B.3 2D Image (color packed)

2D Image Container including one color Component that has one color Intensity data Part.

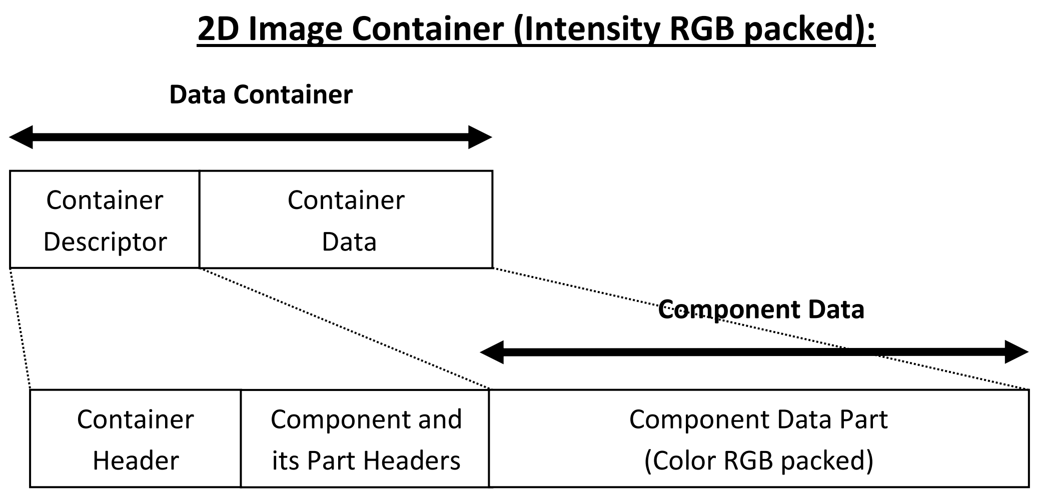
2D Image Container (Intensity RGB packed):
Data Container
Container Descriptor
Container Data
Component Data
Container Header
Component and its Part Headers
Component Data Part (Color RGB packed)

Figure 5-6: 2D Image (color RGB 32 bit packed)

B.4 2D Image (color planar)

2D image Container including one color Component that has 3 monochrome data Parts.

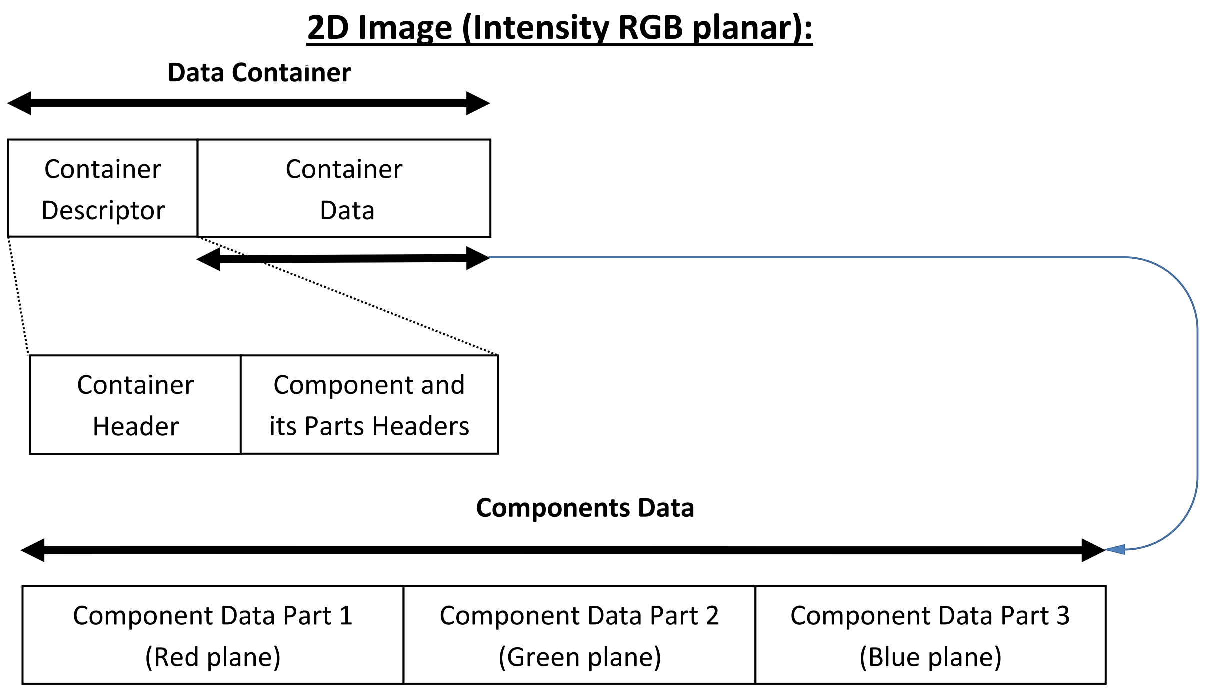
2D Image (Intensity RGB planar):
Data Container
Container Descriptor
Container Data
Container Header
Component and its Parts Headers
Components Data
Component Data Part 1 (Red plane)
Component Data Part 2 (Green plane)
Component Data Part 3 (Blue plane)

Figure 5-7: 2D Image (color RGB planar)

B.5 Multispectral Image (Intensity of multiple wavelength bands)

2D Image Container including one Multispectral Component of 3 wavelenght planes. The Component has 3 data Parts each containing the data of a specific wavelenght. Note that Container format is the same as RGB planar with one Plane/Part per wavelenght band.

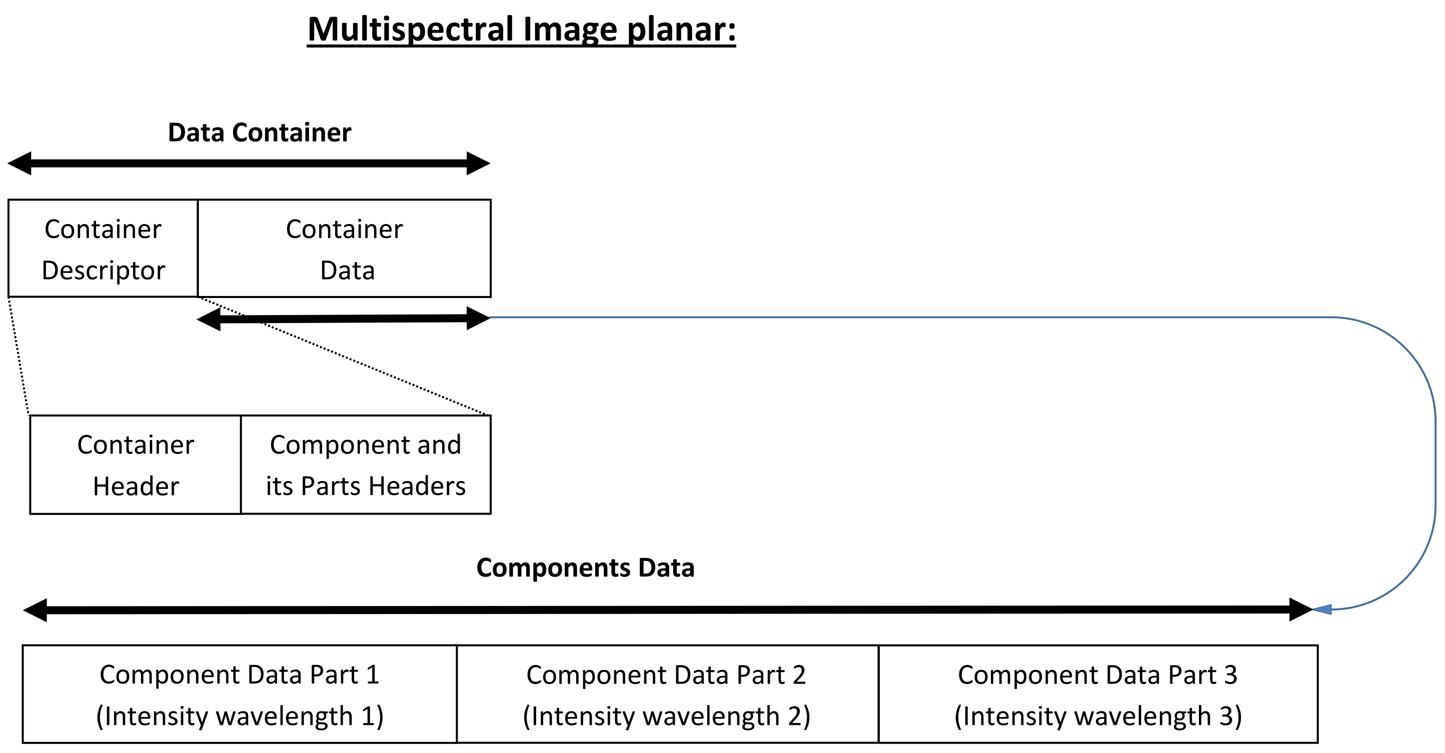
Multispectral Image planar:
Data Container
Container Descriptor
Container Data
Container Header
Component and its Parts Headers
Components Data
Component Data Part 1 (Intensity wavelength 1)
Component Data Part 2 (Intensity wavelength 2)
Component Data Part 3 (Intensity wavelength 3)

Figure 5-8: Multispectral Image (3 wavelength planes)

B.6 2D Image (Compressed)

2D Image Container including one color Component that has one JPEG compressed stream data Part.

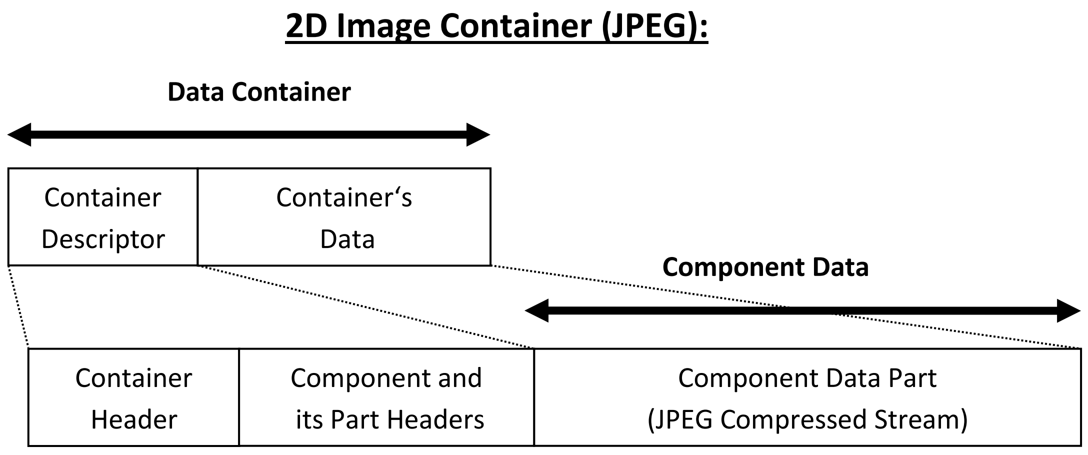
2D Image Container (JPEG):
Data Container
Container Descriptor
Container's Data
Component Data
Container Header
Component and its Part Headers
Component Data Part (JPEG Compressed Stream)

Figure 5-9: 2D Image (color JPEG)

B.7 3D Image (Range, Confidence and Reflectance)

3D scene Container including 3 Components of one data Part each. The 3 individual Components can have different pixel formats.

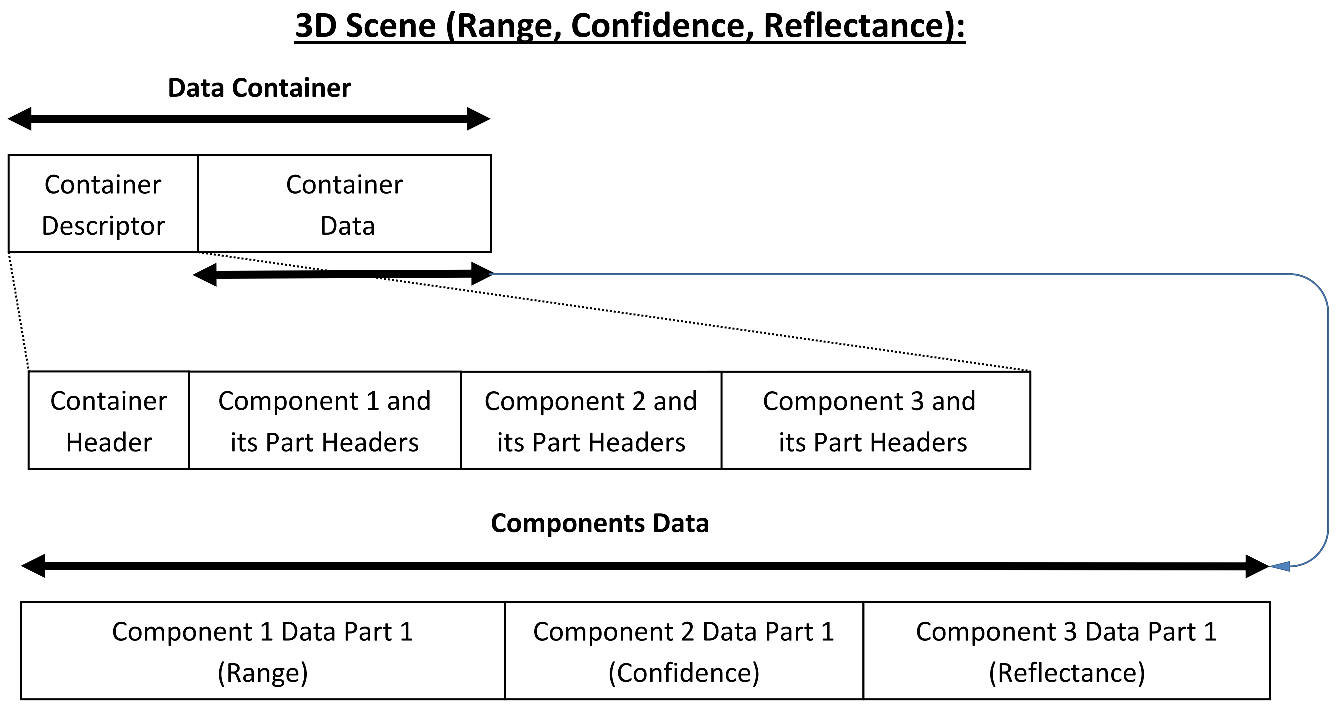
3D Scene (Range, Confidence, Reflectance):
Data Container
Container Descriptor
Container Data
Container Header
Component 1 and its Part Headers
Component 2 and its Part Headers
Component 3 and its Part Headers
Components Data
Component 1 Data Part 1 (Range)
Component 2 Data Part 1 (Confidence)
Component 3 Data Part 1 (Reflectance)

Figure 5-10: 3D Scene (range, confidence and reflectance)

B.8 3D Image (X, Y, Z Planar Point Cloud, Confidence and Reflectance)

3D scene Container including 3 Components point cloud format. The XYZ Component is planar (3 separate data Parts) and the Confidence and Reflectance Components have one data Part each. The 3 Components can have different pixel formats.

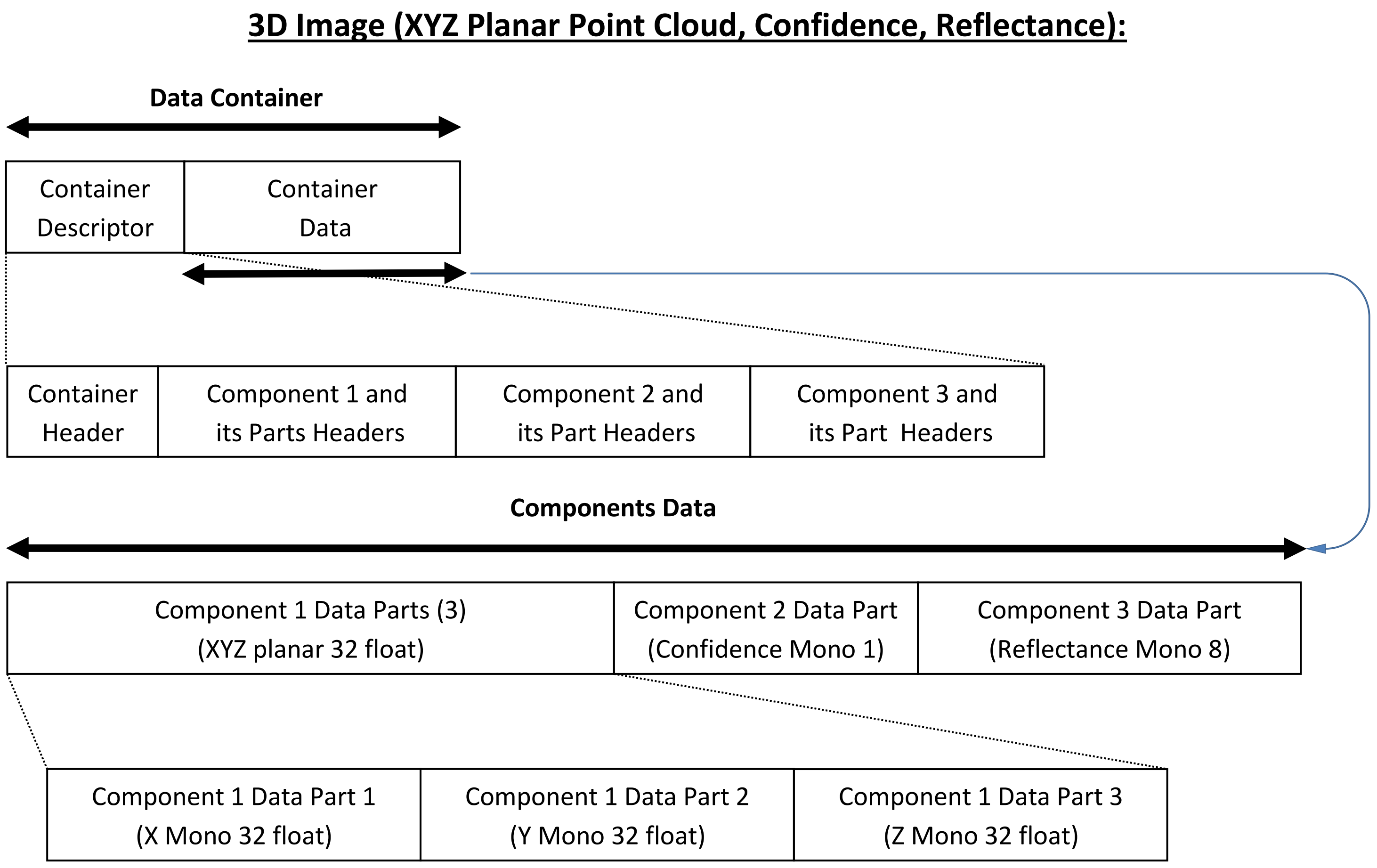
3D Image (XYZ Planar Point Cloud, Confidence, Reflectance):
Data Container
Container Descriptor
Container Data
Container Header
Component 1 and its Parts Headers
Component 2 and its Part Headers
Component 3 and its Part Headers
Components Data
Component 1 Data Parts (3) (XYZ planar 32 float)
Component 2 Data Part (Confidence Mono 1)
Component 3 Data Part (Reflectance Mono 8)
Component 1 Data Part 1 (X Mono 32 float)
Component 1 Data Part 2 (Y Mono 32 float)
Component 1 Data Part 3 (Z Mono 32 float)

Figure 5-11: 3D Image (XYZ planar point cloud, confidence and reflectance)

B.9 2D Image Sequence/Burst (Intensity of consecutive frames taken at different time)

2D Images Sequence Container including 3 Intensity Components. The Components have one monochrome data Part each and a unique Timestamp.

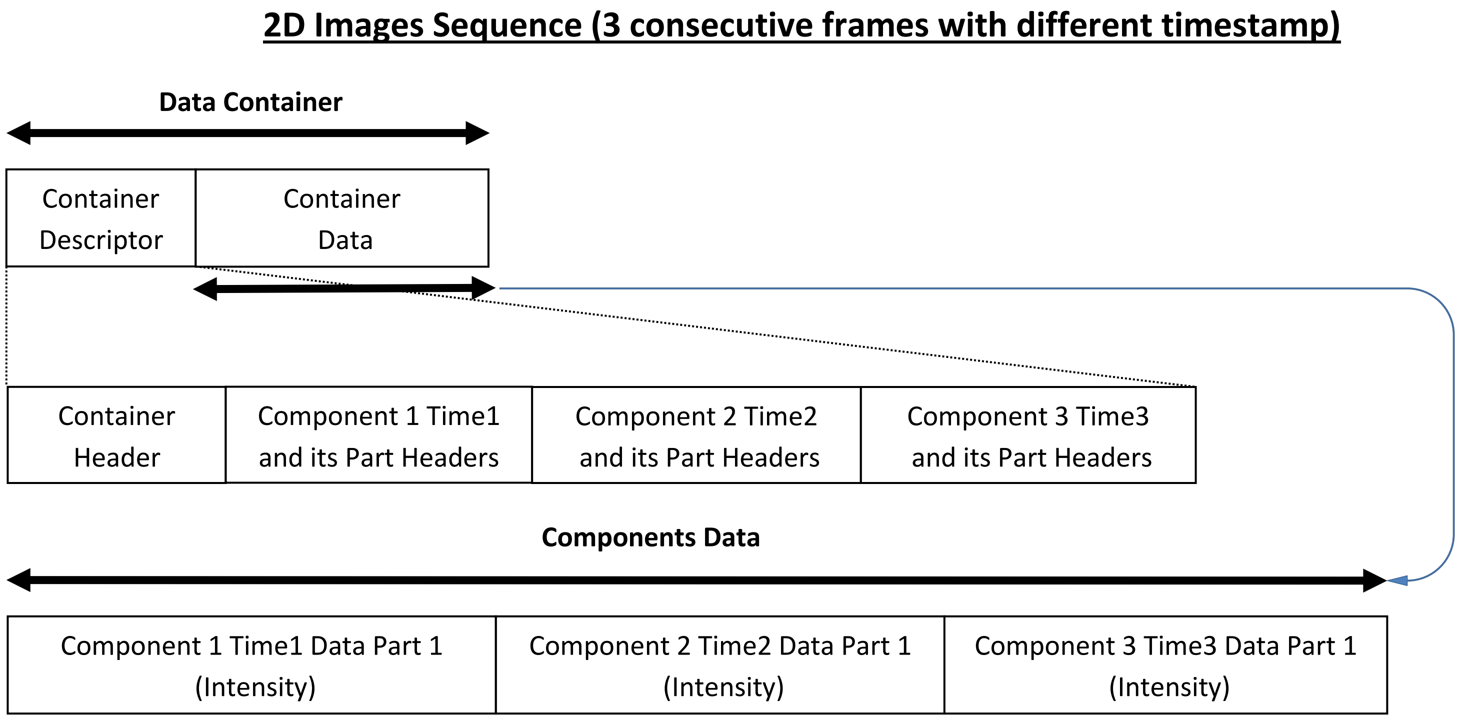
2D Images Sequence (3 consecutive frames with different timestamp)
Data Container
Container Descriptor
Container Data
Container Header
Component 1 Time1 and its Part Headers
Component 2 Time2 and its Part Headers
Component 3 Time3 and its Part Headers
Components Data
Component 1 Time1 Data Part 1 (Intensity)
Component 2 Time2 Data Part 1 (Intensity)
Component 3 Time3 Data Part 1 (Intensity)

Figure 5-12: 2D Image Sequence/burst (Intensity)

B.10 2D Image with multiple regions

2D image Container including 3 regions of Intensity Component. The Components have one region each located at a different X-Y offset and are of different sizes but have all the same source and Timestamp.

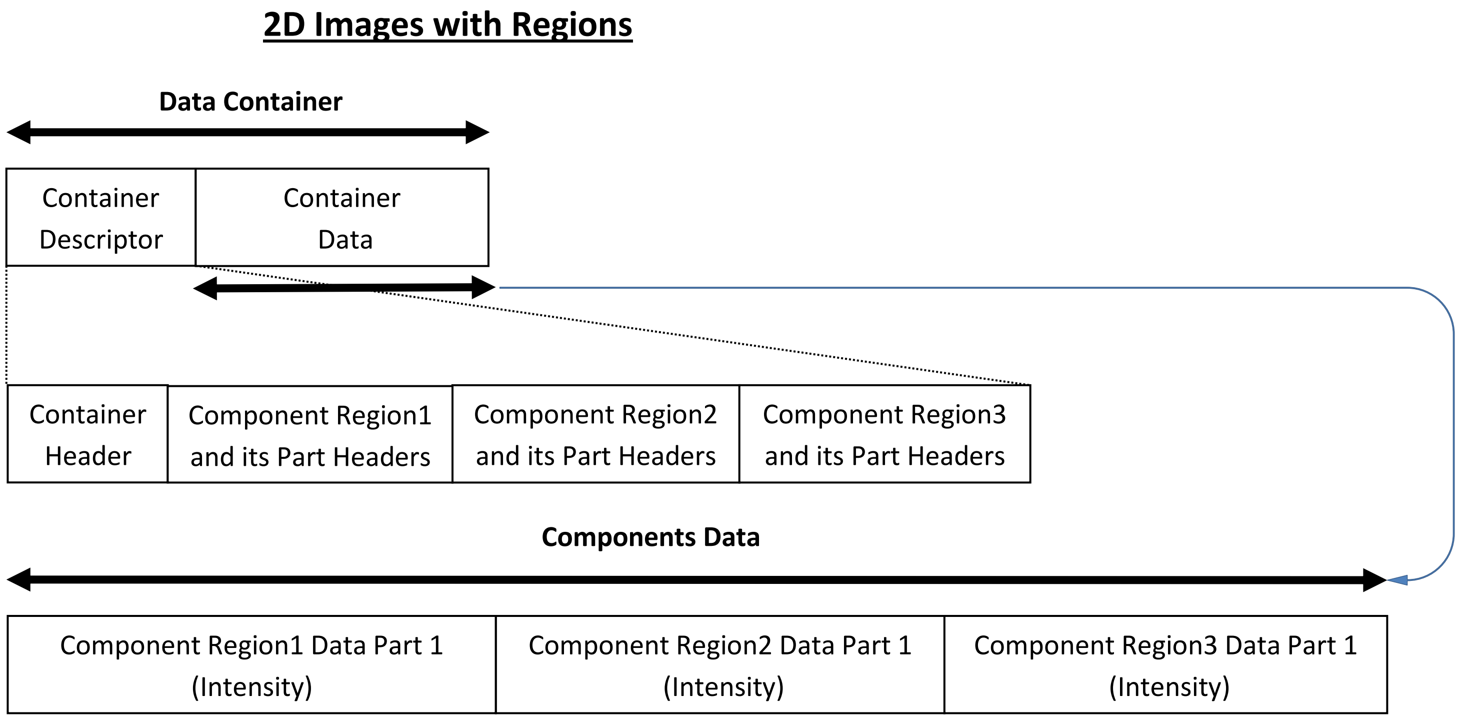
2D Images with Regions
Data Container
Container Descriptor
Container Data
Container Header
Component Region1 and its Part Headers
Component Region2 and its Part Headers
Component Region3 and its Part Headers
Components Data
Component Region1 Data Part 1 (Intensity)
Component Region2 Data Part 1 (Intensity)
Component Region3 Data Part 1 (Intensity)

Figure 5-13: 2D Image with Regions (Intensity)

B.11 2D Image (monochrome Intensity Component with metadata)

A 2D image Container including one Intensity Component of one data Part and one Metadata Component of one data Part (e.g. GenICam Chunk data).

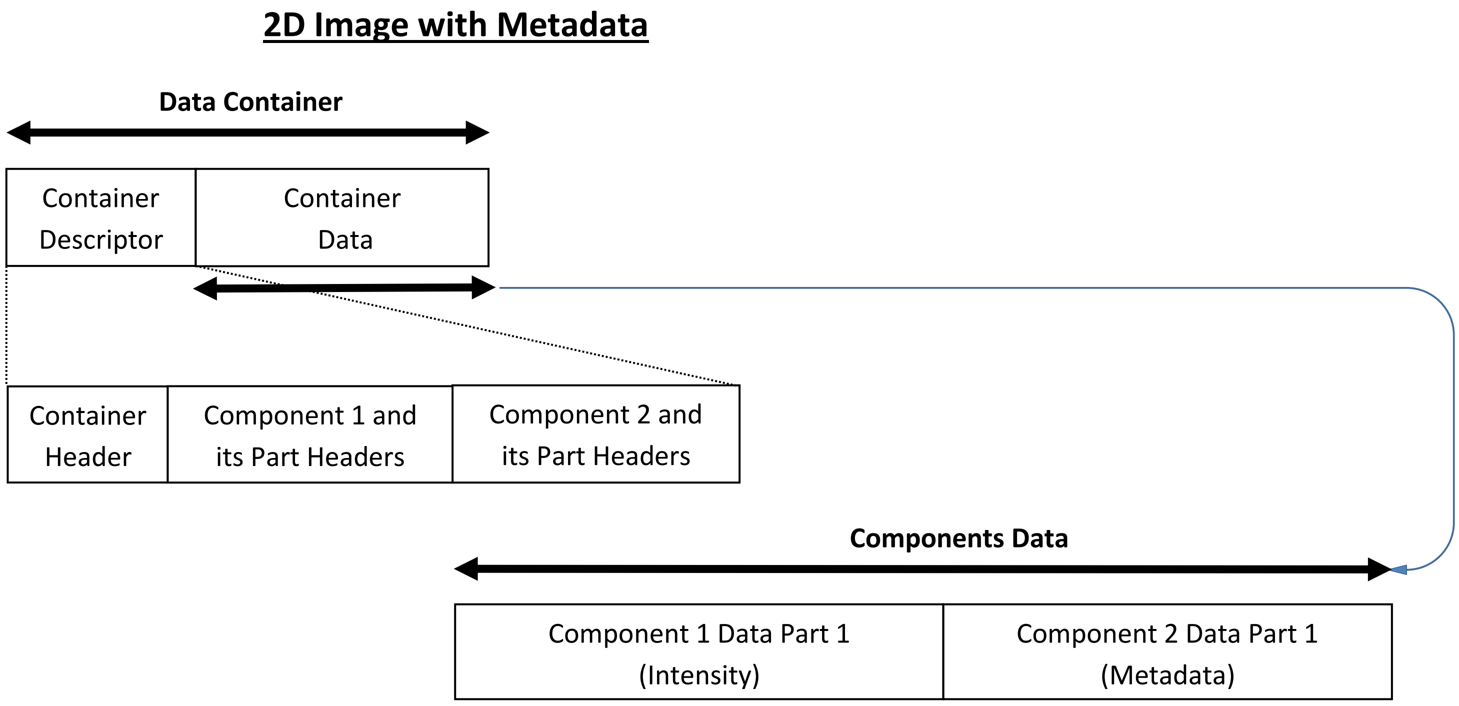
2D Image with Metadata
Data Container
Container Descriptor
Container Data
Container Header
Component 1 and its Part Headers
Component 2 and its Part Headers
Components Data
Component 1 Data Part 1 (Intensity)
Component 2 Data Part 1 (Metadata)

Figure 5-14: 2D Image (intensity with metadata)

B.12 3D Scene with Metadata (Range, Confidence, Metadata)

3D scene Container including 2 image Components of one data Part each and the Metadata associated to that scene. The 3 Components have different pixel formats.

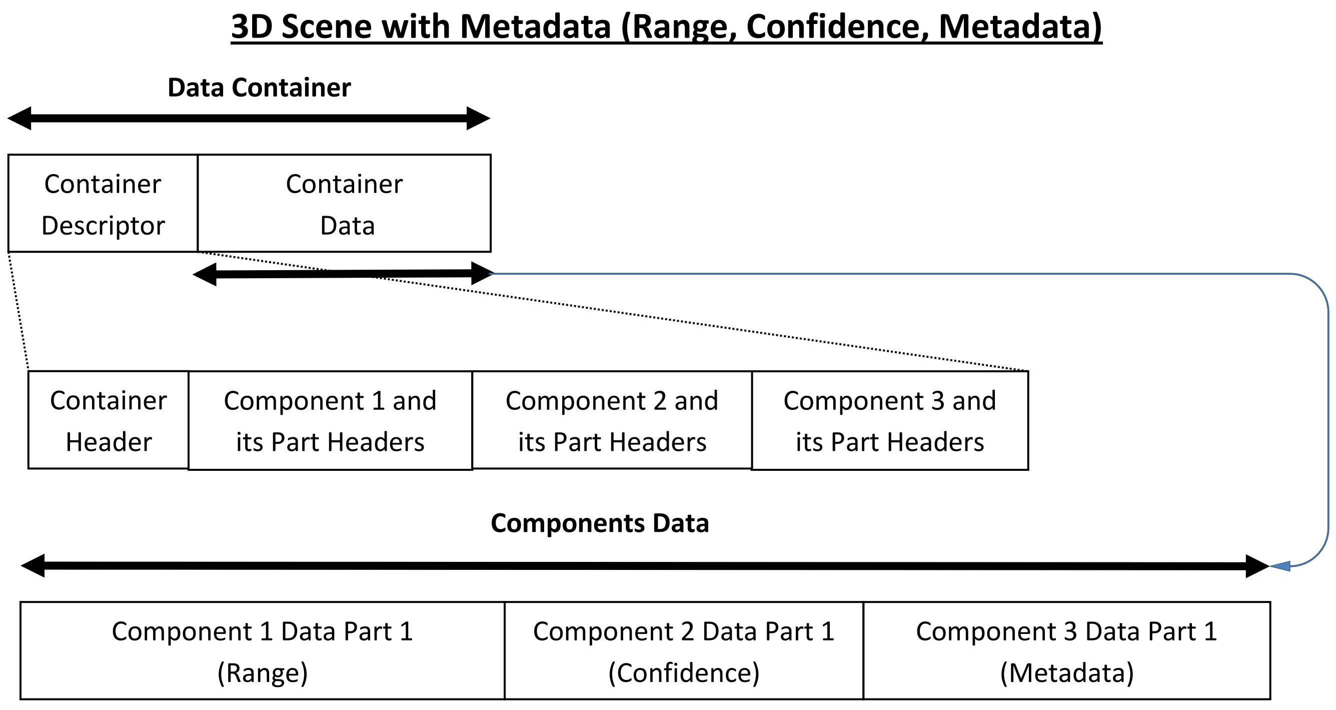
3D Scene with Metadata (Range, Confidence, Metadata)
Data Container
Container Descriptor
Container Data
Container Header
Component 1 and its Part Headers
Component 2 and its Part Headers
Component 3 and its Part Headers
Components Data
Component 1 Data Part 1 (Range)
Component 2 Data Part 1 (Confidence)
Component 3 Data Part 1 (Metadata)

Figure 5-15: 3D Scene (range, confidence and metadata)

Appendix C GenDC Container typical transmission scenarios

This appendix presents some possible scenarios on how GenDC Container could be transmitted on various TLs (for illustration purpose only). Those scenarios are not part of the GenDC Container format definition itself since this specification only target to describe what the Container content is and to be Transport Layer agnostic on the way how it is transported.

C.1 GenDC Container typical streaming using single and multiple data Flows

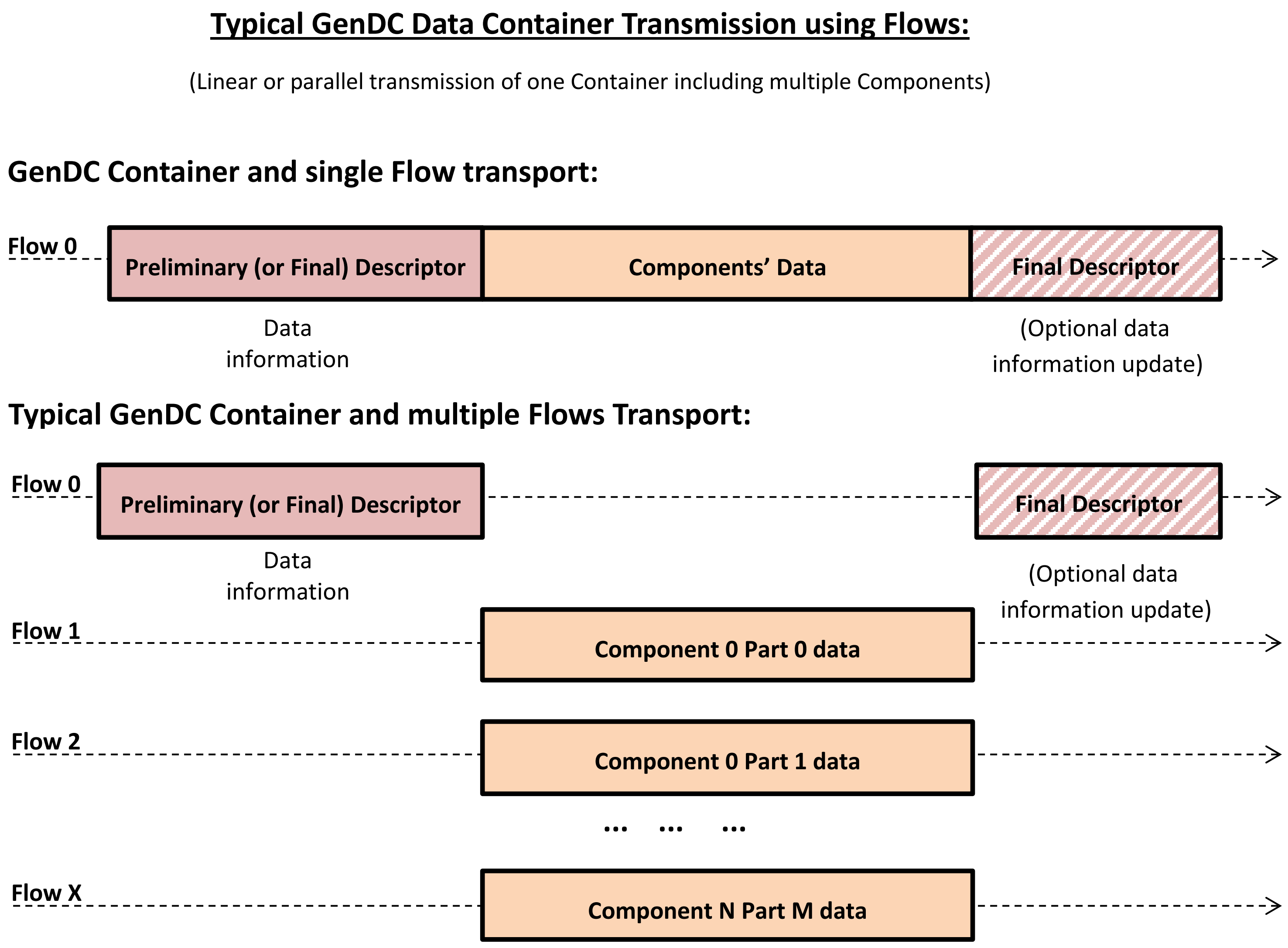
Typical GenDC Data Container Transmission using Flows:
(Linear or parallel transmission of one Container including multiple Components)
GenDC Container and single Flow transport:
Flow 0
Preliminary (or Final) Descriptor
Components' Data
Final Descriptor
Data information
(Optional data information update)
Typical GenDC Container and multiple Flows Transport:
Flow 0
Preliminary (or Final) Descriptor
Final Descriptor
Data information
(Optional data information update)
Flow 1
Component 0 Part 0 data
Flow 2
Component 0 Part 1 data
... ...
Flow X
Component N Part M data

Figure 5-16: Typical GenDC Container's transmission using TL data Flows
(One Container including multiple Components)

C.2 GenDC Container typical transmission using in order transfers

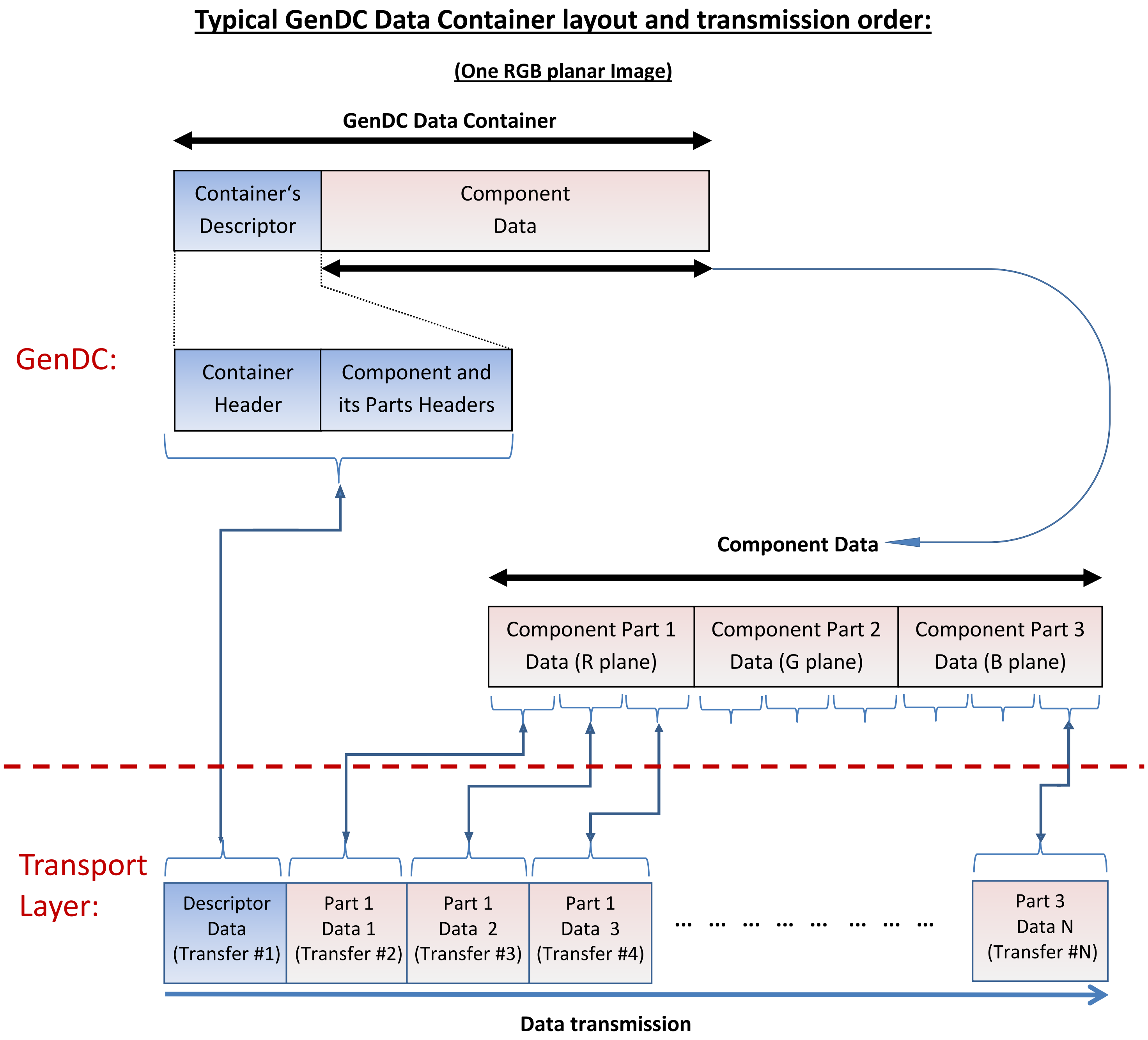
Typical GenDC Data Container layout and transmission order:
(One RGB planar Image)
GenDC Data Container
Container's Descriptor
Component Data
GenDC:
Container Header
Component and its Parts Headers
Component Data
Component Part 1 Data (R plane)
Component Part 2 Data (G plane)
Component Part 3 Data (B plane)
Transport Layer:
Descriptor Data (Transfer #1)
Part 1 Data 1 (Transfer #2)
Part 1 Data 2 (Transfer #3)
Part 1 Data 3 (Transfer #4)
… … … … … … … …
Part 3 Data N (Transfer #N)
Data transmission

Figure 5-17: Typical GenDC Container's layout and transmission order
(One RGB planar image)

C.3 GenDC Container typical transmission using out of order transfers

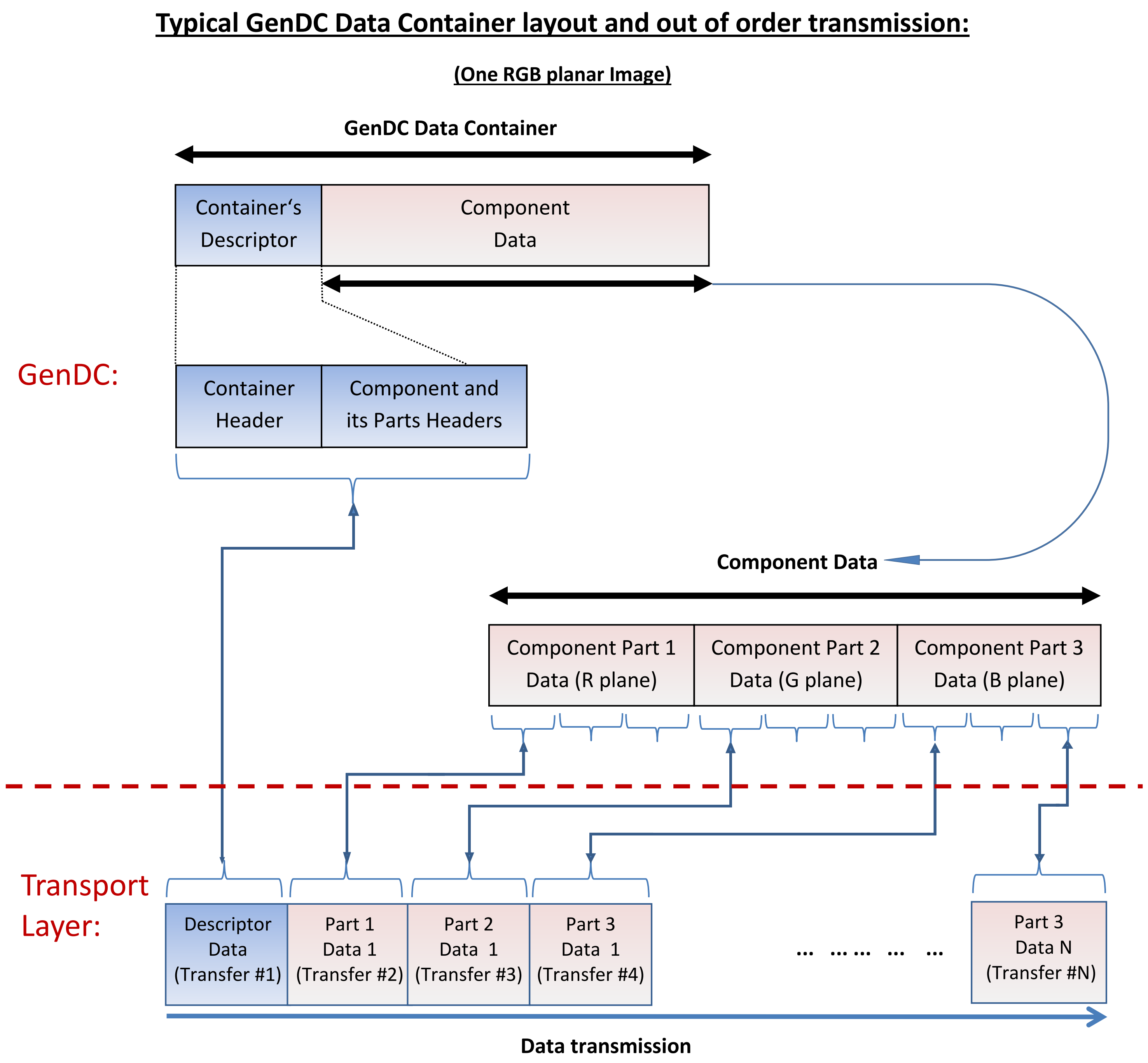
Typical GenDC Data Container layout and out of order transmission:
(One RGB planar Image)
GenDC Data Container
Container's Descriptor
Component Data
GenDC:
Container Header
Component and its Parts Headers
Component Data
Component Part 1 Data (R plane)
Component Part 2 Data (G plane)
Component Part 3 Data (B plane)
Transport Layer:
Descriptor Data (Transfer #1)
Part 1 Data 1 (Transfer #2)
Part 2 Data 1 (Transfer #3)
Part 3 Data 1 (Transfer #4)
… … … … …
Part 3 Data N (Transfer #N)
Data transmission

Figure 5-18: Typical GenDC Container's layout and out of order transmission (One RGB planar image)