IEC 62842

Edition 1.0 2015-09

INTERNATIONAL STANDARD

Multimedia home server systems – File allocation system with minimized reallocation

IEC 62842:2015-09(en)

®

IEC 62842

Edition 1.0 2015-09

INTERNATIONAL STANDARD

Multimedia home server systems – File allocation system with minimized reallocation

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ICS 33.160.60

ISBN 978-2-8322-2843-2

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

MULTIMEDIA HOME SERVER SYSTEMS – FILE ALLOCATION SYSTEM WITH MINIMIZED REALLOCATION

FOREWORD

International Standard IEC 62842 has been prepared by technical aerea 8: Multimedia home systems and applications for end-user network of IEC technical committee 100: Audio, video and multimedia systems and equipment.

The text of this technical report is based on the following documents:

CDV	Report on voting
100/2367/CDV	100/2459/RVC

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC website under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

A bilingual version of this publication may be issued at a later date.

INTRODUCTION

Recently, hard disk and Blu-ray Disc[1] recorders have become popular in the home to record television programmes. Normally a Hard Disk Recorder (HDR) is used for time shift and a Blu-ray Disc (BD) is used for library. When an HDR is used for time shift, television programmes are recorded and played, then many of them are deleted to reuse the spaces for other programmes to be recorded. These programmes are stored as files in a hard disk drive (HDD) using a file system. Continuous recording and deletion of programmes involves the continuous storing and deletion of files in the file system. Television programme streams include at least videos and an electronic programme guide (EPG). The HDR stores videos in a long, variable length file depending on the quality and recording hours. Compared with videos, EPG related information is stored in a shorter file or files but is often updated. This continuous creation, deletion and updating of files of different lengths finally causes the files to be stored in fragments, and the system performance becomes very low.

In a computer, defragmentation tools are provided to solve the problem of a fragmented file system. Normally defragmentation with reallocation of files in sequence takes a long time and the end user cannot but wait for the completion of the defragmentation, with no other activity. In the home server environment, a smarter solution to resolve this problem needs to be provided.

The recent newly developed HDD features will be reflected in the next version of the standard.

1 Blu-ray Disc™ is a trademark of the Blu-ray Disc Association. This information is given for the convenience of users of this document and does not constitute and endorsement by IEC of the product named.

MULTIMEDIA HOME SERVER SYSTEMS – FILE ALLOCATION SYSTEM WITH MINIMIZED REALLOCATION

1 Scope

This International Standard specifies the method for allocating requested file space with no fragmentation, to minimize the need for reallocation of fragmented files in the Universal Disc Format (UDF) file system applied to hard disk drives used in hard disk recorders.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 13346 (all parts), *Information technology – Volume and file structure of write-once and rewritable media using non-sequential recording for information*

ISO/IEC 13346-1:1995, *Information technology – Volume and file structure of write-once and rewritable media using non-sequential recording for information interchange – Part 1: General*

ISO/IEC 13346-3:1999, *Information technology – Volume and file structure of write-once and rewritable media using non-sequential recording for information interchange – Part 3: Volume structure*

ISO/IEC 13346-4:1999, *Information technology – Volume and file structure of write-once and rewritable media using non-sequential recording for information interchange – Part 4: File structure*

OSTA UDF2.01:200, *Information technology – OSTA Universal Disk Format Specification, Revision 2.01*

Secure Universal Disk Format Specification Revision 1.00, *Optical Storage Technology Association (OSTA),* http://www.osta.org/

3 Terms, definitions, abbreviations and notation

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
partition
region allocated to a file system by a disk volume space management system

3.1.2
virtual container partition
virtual partition containing a partition which has a minimum size of power-of-2 of allocation unit size of the disk

3.1.3
buddy
region allocation method where a given region having a power-of-2 unit size is recursively divided into two equal size regions (as 'buddies') until it reaches to one unit in size and, if a region of a given size is requested, the smallest free power-of-2 unit size region that can contain the requested size region is allocated

3.1.4
Concatenation of power of 2
CoPo2
basic allocation method of this standard

3.1.5
divide
process of obtaining divided-partitions through first identifying master-divided partitions, then applying the buddy method to them to get divided-partitions and finally allocating divided-partition numbers

3.1.6
master-dividing
process of identifying the master-divided-partition in the first process of dividing

Note 1 to entry: When the size of a partition is a power-of-2 unit size, the partition as a whole constitutes a master-divided partition.

Note 2 to entry: When the size of a partition is expressed as the sum of mutually different power-of-2 sizes with the power-of-2 size that constitutes the sum, constitute the partition as a concatenation of the areas in the sequence of those sizes as master-divided partitions in decreasing order of size.

3.1.7
master-divided-partition
divided partition identified by master-dividing

3.1.8
one unit in size
predetermined minimum unit of memory size that can be allocated

3.1.9
divided-partition
partition identified by dividing the master-divided partitions in a given partition recursively in half until it reaches one unit in size

3.1.10
divided-partition level
level value of a divided-partition expressed by the power-of-2 value of the divided partition compared to unit size

Note 1 to entry: This is often abbreviated as level n (where n specifies level number).

3.1.11
partition level
level n
abbreviation of divided-partition level, normally used in a simple form, 'level n'

3.1.12
divided-partition pair
pair of divided-partition formed by dividing a divided-partition into two halves as buddies

3.1.13
divided-partition number
number assigned to divided-partitions in the process of dividing a given partition from top level to the lowest level incrementally

Note 1 to entry: Numbers are assigned to each divided-partition from top to bottom and from left to right in the same level with the starting number 1, counting up by one.

3.1.14
master-divided-partition number
divided-partition number assigned to each master-divided-partition

3.1.15
master-divided partitions table
table for managing the master-divided partitions

3.1.16
master-divided-partition number management table
table for managing the master-divided-partition number in contrast with each divided-partition level

3.1.17
end position management table
table for managing the maximum divided-partition number assigned in each divided-partition level

3.1.18
highest divided-partition level management table
table for managing the highest divided-partition level for the partition

3.1.19
multilevel-divided partition management tables
three tables for managing the divided-partitions

Note 1 to entry: The three tables are the master divided-partition number management table, the end position management table and the highest divided-partition level management table.

3.1.20
multilevel-divided-partition allocation table
table for managing the multilevel-divided partitions state in each level with 2 bits

3.1.21
segment
region to be taken or allocated from a partition

Note 1 to entry: If the size of a region is power-of-2 unit size, the region consists of one divided partition.

Note 2 to entry: If the size of a region is a sum of polynomials of power-of-2 of a unit size, the region consists of a concatenation of multilevel divided-partitions, each corresponding to one polynomial component, and it is taken from the container divided-partition for a segment.

3.1.22
multilevel-dividing
procedure for obtaining a region as a segment, consisting of the concatenation of multilevel divided-partition decreasing order in size, of which each corresponds to one power-of-2 polynomial component

3.1.23
container divided partition for a segment
simple power-of-2 region determined by rounding up the size of a polynomial-based requested segment to the size of the nearest simple power-of-2 region and that simple region will consist of the segment to be allocated and adjacent-multilevel-segment

3.1.24
adjacent-multilevel-segment
segment adjacent to the allocated multilevel-segment of which each component divided-partition is allocated in increasing order of size, and which constitutes the rest of the container divided-partition for a segment

3.1.25
first-pass-allocation
first step to allocate a segment, to obtain a minimum divided partition including the requested segment

Note 1 to entry: If a segment is a single level divided-partition, the allocation is completed, but if a segment is multilevel, the process continues to a second-pass allocation.

3.1.26
second-pass-allocation
second step to allocate a multilevel segment, in a segment allocation, where the multilevel-dividing is applied to the divided-partition obtained from the first-pass-allocation

3.1.27
provisional-allocation
procedure to get a segment, where, if a divided-partition including the target segment is not found, a search for the available upper level of a divided-partition is requested and the divided-partition found is allocated as provisional allocation

Note 1 to entry: After getting a provisional divided-partition, first-pass-allocation continues and in the case of allocating a multilevel-segment, second-pass-allocation continues.

3.1.28
allocation state
divided-partition state in view of allocation, classified into available, occupied and reserved

3.1.29
available
allocation state showing the divided-partition is free to use, meaning available

Note 1 to entry: Divided-partitions are classified into first-pass-available and second-pass-available. When searching for a free divided-partition, start by searching for first-pass-available and then search for second-pass-available.

3.1.30
first-pass-available
initial available state assigned to free divided-partitions

Note 1 to entry: When an occupied divided-partition becomes free, its allocation state is set as first-pass-available.

3.1.31
second-pass-available
available allocation state assigned to each component divided-partition of adjacent-multilevel-segments, after allocation of the multilevel-segment

3.1.32
occupied
allocation state showing the divided-partition is in use

3.1.33
reserved
divided-partition allocation state that is not appropriate to classify into available or occupied

Note 1 to entry: Divided-partitions other than available or occupied are set as reserved. When getting a segment through provisional-allocation, the allocated top divided-partition and the divided-partitions in lower levels shall be set as reserved.

3.1.34
master boot record
conventional partitioning record in logical block address 0 of HDD

3.1.35
logical block address
address scheme used for HDD volume space

3.1.36
logical sector number
address scheme used for UDF volume in HDD volume space

Note 1 to entry: UDF volume might be restricted in the first partition of HDD. LSN = LBA − 63.

3.1.37
logical block number
address scheme used for UDF partition in UDF volume space

Note 1 to entry: LBN = LSN − 257.

3.2 Abbreviations

HDD	Hard Disk Drive
MBR	Master Boot Record
LBA	Logical Block Address
LSN	Logical Sector Number
LBN	Logical Block Number
OS	Operating System
UDF	Universal Disk Format

3.3 Notation

Universal disk format (UDF) is an application of the ISO/IEC 13346 series and the specification of the data structure always has a reference to the corresponding part of ISO/IEC 13346 in the form of “ISO/IEC-x.a.b.c” where x is a part number and a.b.c is the paragraph number. In Clause 9, data structure is described in C-like form. The fields in the form that need explanation are written in bold, and the explanation is provided just after the structure.

These standard-specific compound words are expressed by words concatenation with hyphen.

4 Precondition and the policy

4.1 Preconditions

When applying UDF on HDD, the preconditions considered are as follows:

a) a disk volume is configured with partitions managed by the partition manager for the HDD system. Each partition is a contiguous region;

b) a partition is a contiguous region which is an integer multiple of a cluster, which is an integer multiple of a sector.

4.2 Policy

This standard applies the allocation method which tries to allocate the larger free space as far as possible. Using this method of allocation, the possibilities of fragmentation and reallocation are minimized.

The buddy system can allocate the minimum contiguous power-of-2 cluster region which contains the requested size. The region of the clusters left over the requested size, which are waste, are called slack. The slack size becomes larger and larger, when the requested size become larger. This is the reason why the buddy system only applies to memory management systems in the small system and does not apply to the file system.

The CoPo2 (concatenation of power-of-2) resolves the problem of buddy, which helps to use the slack, and can be applied to the file system which minimizes the possibilities of fragmentation and reallocation.

5 Method to be applied-CoPo2

In buddy, a region expressed in binary 1000 in cluster size, a possible divided partition that could be allocated can be expressed as a tree of divided partitions from size 1000 (level 3) to 0001 (level 0):

								Divided-partition level
			(#0,#1)					level 3 (size=1000)
	(#0,#2)					(#1,#3)		level 2 (size=0100)
(#0,#4)		(#1,#5)			(#2,#6)		(#3,#7)	level 1 (size=0010)
(#0,#8)	(#1,#9)	(#2,#10)	(#3,#11)	(#4,#12)	(#5,#13)	(#6,#14)	(#7,#15)	level 0 (size=0001)

In the above tree, each divided partition is expressed as "(#level-internal-divided-partition-number, #divided-partition-number-in-tree)". In other words, in two parenthesis, the hash sign expresses a sequential number, followed by the divided partition number followed by a comma to separate the following expression followed by a hash sign to express a sequential number followed by the divided-partition number in the total tree.

To obtain a 0101 size segment in the tree, a solution is getting divided partition number in tree #2 and #12. (in the following, #n expresses divided-partition-number-in-tree). In buddy this solution is not available, instead the minimum size of divided-partition that contains #2 and #12 is obtained, which is #1. In reality #2 and #12 are used in #1, and the #13 and #7 are left and become slack and could not be used.

In buddy, a cluster that is the multiple of a sector is used as the basic allocation unit. To manage the cluster allocation state in buddy, one bit is used to express 'occupied' or 'available'. With this method, the slack cannot be managed. To manage slack, each divided-partition state requires two bits for state management.

The key states of a divided-partition can be 'occupied' or 'available'. Initially, only #1 is available where a space can be allocated. The other divided-partitions belong to the #1 and cannot be allocated. But when a size 0101 segment is allocated, #1 gives control for space management to the lower layer divided-partitions and #2 and #12 become occupied and concatenated together to form a segment of size 0101. As a result, the states of the upper and lower layer divided-partitions change. In this process #13 and #7, that is a slack in buddy, form an adjacent-multilevel-segment and become available space in the adjacent area of the allocated segment.

Considering the practicality of optimum usage of 2 bits, divided-partitions are classified as follows:

a) available(0x), where space can be allocated, is classified in two states as available1(00) and available2(01) that is normally a slack area in buddy but available in this system.
b) occupied(1x), where space cannot be allocated, is classified in two states as in-use(11) and reserved(10). In-use is used for allocated divided-partitions. Reserved is used for the divided-partitions which depend on the upper or lower layer of a divided-partition in allocations and are reserved for allocation decision.

In summary, the state states shrink into four, as follows:

– available(0x)
  • available1(00)-available in normal;
  • avaiable2(01)-become available in adjacent-multilevel-segment;
– occupied(1x)
  • reserved(10);
  • lin-use(11).

Using those states explains the state change using a simple partition:

The initial state of a partition in tree form is as follows:

```
                                        (#0,#1;00)                                                level 3
                                                                                             (size=1000)
                    (#0,#2;10)                                  (#1,#3;10)                        level 2
                                                                                             (size=0100)
          (#0,#4;10)            (#1,#5;10)            (#2,#6;10)            (#3,#7;10)            level 1
                                                                                             (size=0010)
(#0,#8;10) (#1,#9;10) (#2,#10;10) (#3,#11;10) (#4,#12;10) (#5,#13;10) (#6,#14;10) (#7,#15;10)     level 0
                                                                                             (size=0001)
```

only #1 is available1 (00) and the others are all reserved(10). If a size 0101 segment is allocated, the states change as follows:

– #1 changes to reserved(10);
– #2 changes to in-use(11) and no change in the lower level divided-partitions in the group;
– #3 has no change, but
– #12 changes to in-use(11);
– #7 and #13 change to available2;
– #14 and #15 have no change.

The following are the changes made expressed in tree form:

```
                                   (#0,#1;10)                                             level 3
                                                                                      (size=1000)
               (#0,#2;11)                                    (#1,#3;10)                   level 2
                                                                                      (size=0100)
      (#0,#4;10)            (#1,#5;10)            (#2,#6;10)            (#3,#7;01)        level 1
                                                                                      (size=0010)
(#0,#8;10) (#1,#9;10) (#2,#10;10) (#3,#11;10) (#4,#12;11) (#5,#13;01) (#6,#14;10) (#7,#15;10)   level 0
                                                                                      (size=0001)
```

6 Explanation of basic method CoPo2

6.1 Basics

In the history of file allocation, space management was based on the occupation state states of basic allocation unit. Each allocation unit has a one bit state showing occupied or free. In order to allocate a long file, free allocation units are requested of sufficient total size. This bottom up approach is time-consuming and the algorithm itself has the basic problem that it causes fragmentation.

The better way is taking a top-down approach using buddy and improving the buddy to make slack available. Using this, any size expressed by binary notation can be allocated contiguously with concatenation of the divided-partitions each mapped to the binary ones of the binary size.

Based on these considerations, the method is named CoPo2 (concatenation of power-of-2).

6.2 Two choices to apply CoPo2 to an existing partition scheme

6.2.1 General

Unfortunately existing partitions do not use a power-of-two size of basic allocation unit. To apply buddy to such existing partitions, two approaches can be considered. These are applied to existing partitions and to virtual container partitions.

6.2.2 Applying to an existing partition

Buddy could not be applied to size 1011 of basic allocation unit of a partition. If this partition is considered as a concatenation of buddy divided-partitions of sizes 1000(8), 0010(2) and 0001(1) of basic allocation unit, buddy can be applied for each divided-partition as master. Let this method of partitioning be called master partitioning and name the divided-partitions as master divided-partitions. This configuration of partition is defined by a master divided-partition table and expressed as 1011.

In the following, #1, #8 and #19 are the master divided-partitions and quoted with double parentheses. In this divided-partition numbering, put the most upper level divided-partition as #1 and put continuous numbers to downward and left to right in the same level.

size(11=8+2+1) ((8)) ((2)) ((1))

level 3(size8) ((#1))

level 2(size4) (#2) (#3)

level 1(size2) (#4) (#5) (#6) (#7) ((#8))

level 0(size1) (#9) (#10) (#11) (#12) (#13) (#14) (#15) (#16) (#17) (#18) ((#19))

This master divided-partition number management table is used for managing the master divided-partition at each level. If a master divided-partition does not exist in the level, it is set -1. If the total levels are 6, in the above case, the table is set as (-1, -1, 1, -1, 8, 19).

The start position management table is used for managing the starting divided-partition number of each level and in above case it is set as (-1, -1, 1, 2, 4, 9).

Total number of divided-partitions is managed by the total-number-of-divided-partitions table and it is set as 19, in above case.

The multilevel-divided-partition allocation table is used for managing the allocation state of each divided-partition of each level. The initial states of master divided-partitions are set as available1(00) and others are set reserved(10).

In the above case, if we use "/" as delimiter of each level, the allocation state of the above four levels is expressed as (00/10,10/10,10,10,10, 00/10,10,10,10,10,10,10,10,10,10,00).

The divided-partition number of adjacent divided-partitions in lower levels can be identified based on the target divided-partition with calculations using the tables.

6.2.3 Applying to a virtual container partition

A virtual container partition is a minimum power-of-2 unit size virtual partition that contains the given partition. The buddy divided-partition numbering can be applied to the virtual container partition.

From a size (11=8+2+1) partition expressed as concatenation of the following three divided-partitions "((8))((2))((1))", we can form a virtual container partition with buddy divided-partition numbering.

level4(size16) (#1 virtual divided-partition forming virtual container partition)

level3(size8) ((#2)) #3is a virtual divided-partition

level2(size4) (#4) (#5) #6 and #7 are virtual divided-partitions

level1(size2) (#8) (#9) (#10) (#11) ((#12)) #13,#14 and #15 are virtual divided-partitions

level0(size1) (#16)(#17) (#18)(#19) (#20)(#21) (#22)(#23) (#24)(#25) ((#26)) #27 to #31 are virtual divided-partitions

Figure 1 shows the graphical illustration of the above virtual container partition.

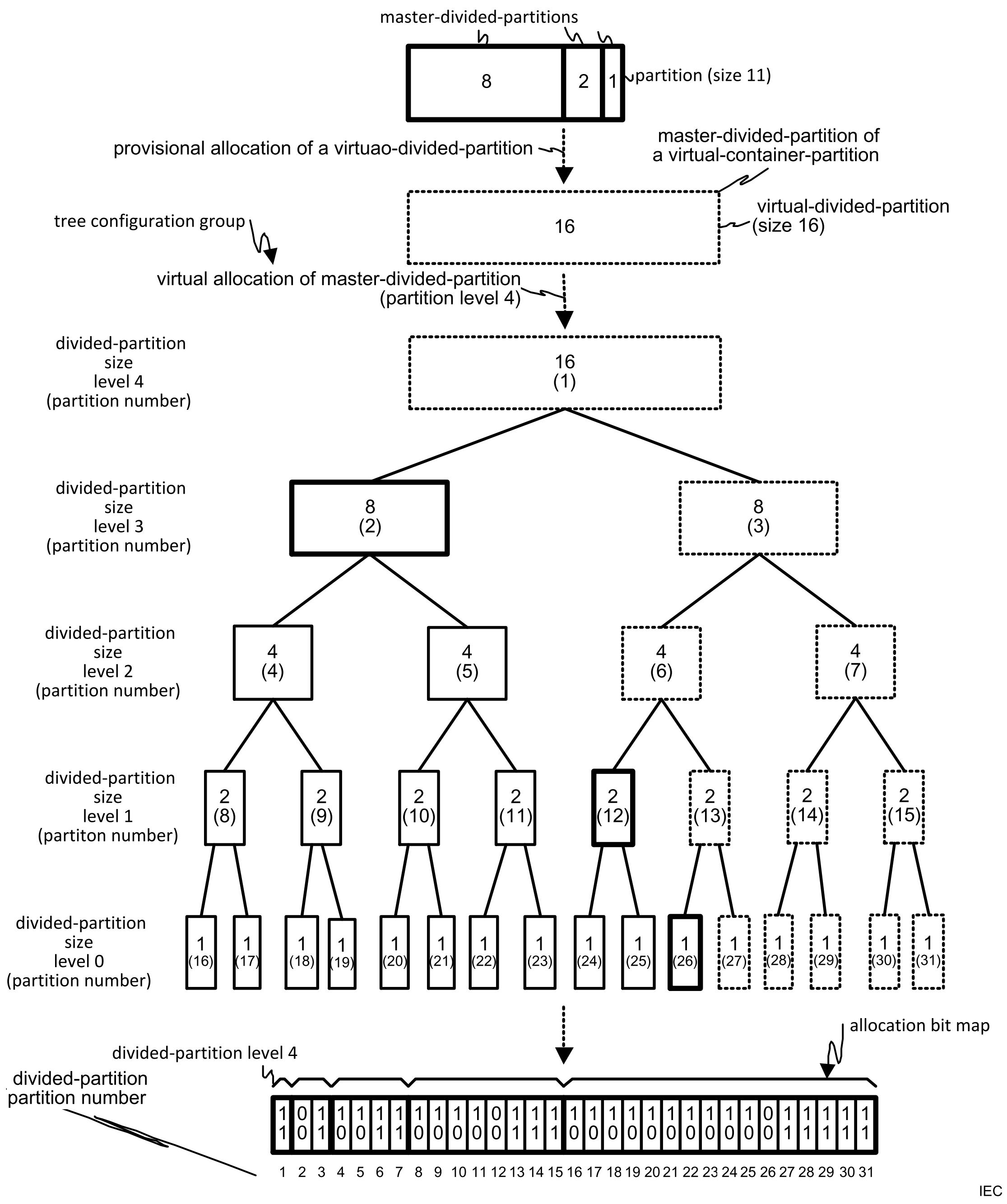
master-divided-partitions
8
2
1
partition (size 11)
provisional allocation of a virtuao-divided-partition
master-divided-partition of a virtual-container-partition
tree configuration group
16
virtual-divided-partition (size 16)
virtual allocation of master-divided-partition (partition level 4)
divided-partition size level 4 (partition number)
16 (1)
divided-partition size level 3 (partition number)
8 (2)
8 (3)
divided-partition size level 2 (partition number)
4 (4)
4 (5)
4 (6)
4 (7)
divided-partition size level 1 (partiton number)
2 (8)
2 (9)
2 (10)
2 (11)
2 (12)
2 (13)
2 (14)
2 (15)
divided-partition size level 0 (partition number)
1 (16)
1 (17)
1 (18)
1 (19)
1 (20)
1 (21)
1 (22)
1 (23)
1 (24)
1 (25)
1 (26)
1 (27)
1 (28)
1 (29)
1 (30)
1 (31)
allocation bit map
divided-partition level 4
divided-partition partition number
1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
1 0 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
IEC

Figure 1 – Virtual container partition

6.2.4 Choice conclusion

With two choices, this standard chose the application of virtual container partition evaluating the benefit of the simplicity of natural extension of buddy.

6.3 Management tables for CoPo2

6.3.1 General

Figure 2 shows the management tables for the standard. In the figure, the divided-partition level is abbreviated as 'partition level'.

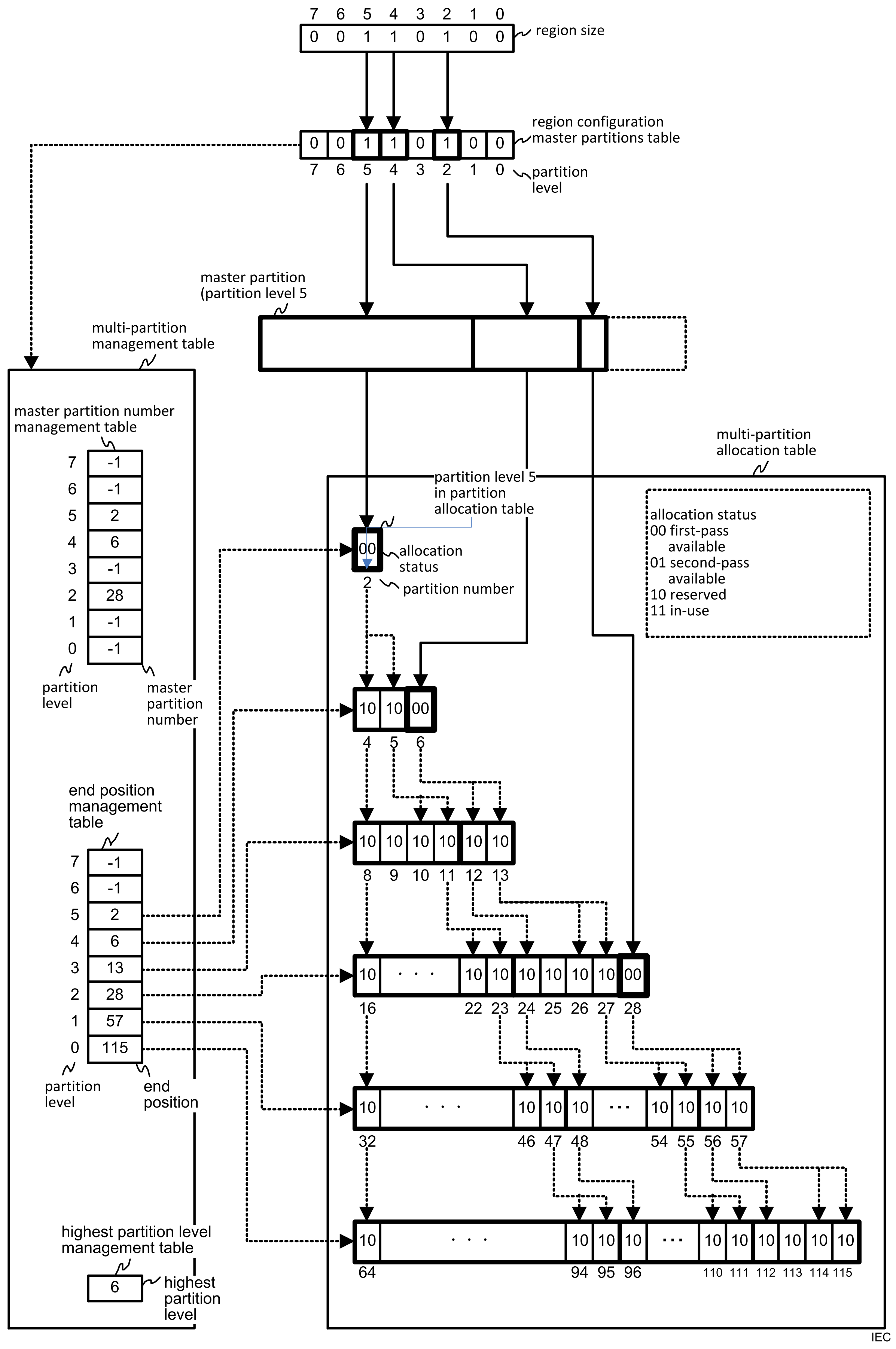

Figure 2 – Management tables for CoPo2

6.3.2 Region configuration master partition table

This table manages the master-divided partitions which configure a region that constitutes a partition.

6.3.3 Multilevel-divided-partition management tables

6.3.3.1 General

The tables are master-divided-partition number management table, endpoint management table and upper most level number management table.

6.3.3.2 Master-divided-partition number management table

This table manages the numbers that contain the master-divided-partitions. The value “-1” means no assignment in the level.

6.3.3.3 Endpoint management table

This table manages the endpoint divided-partition number of the level. The value “-1” means no assignment in the level.

6.3.3.4 Upper most level number management table

This table manages the uppermost level number possible in the partition.

6.3.3.5 Multilevel-divided-partition state management table

This table manages the allocation state of the divided-partitions in multilevel. The states are first-pass-available(00), second-pass-available(01), reserved(10) and in-use(11).

6.4 Functions required to implement CoPo2

6.4.1 General

This subclause explains the functions required to implement CoPo2.

6.4.2 Initialize

6.4.2.1 Get-partition-size

This function gets the size of the partition from the Operating System (OS) and creates the region configuration master partition table.

6.4.2.2 Create multilevel-divided-partition state management table

In the multilevel-divided-partition state management table, set all the divided-partitions in use, then set reserved until the endpoint of each level and set first-pass available in master-divided-partition numbers pointed by endpoint management table.

6.4.3 Manage-multilevel-divided-partitions

6.4.3.1 Search-for-available-divided-partition

First search for first-available divided-partition and secondly search for second-available divided-partition.

6.4.3.2 Allocate-segment

Search for an available minimum divided-partition that contains the requested segment size as follows:

a) If the divided-partition is found, execute first-pass allocation with the following:
   - the single divided-partition segment case: return the segment;
   - multilevel-divided-partition segment case: execute a second-pass allocation to get a multilevel-divided-partition and leave the rest as a adjacent-multilevel-segment;
b) if the divided-partition is not found, search for the available divided-partition in an upper level:
   - if it is found, execute the provisional-allocation, then execute first-pass allocation and set the adjacent-multilevel-segment as second-pass available.

6.4.3.3 Freeing a segment

To free the requested segment, check the adjacent-multilevel-segment together with the segment and execute the following:

a) If two same level contiguous divided-partitions, each of which belongs to the requested segment and the adjacent-multilevel-segment are available, set the two divided-partitions as reserve and set the upper layer divided-partition containing the two as first-pass-available. While the upper layer of the two same level contiguous divided-partitions can be set first-pass-available, keep the same procedure recursively.
b) In other cases, set the divided-partitions belonging to the requested segment first-pass-available.

7 Considerations on the size of management tables

7.1 General

The biggest management table is a multilevel-divided-partition allocation table and the size depends on the sector size and size of the partition.

7.2 Multilevel-divided-partition allocation table size

7.2.1 Blu-ray

Sector size:	2 KB
Total size:	128 GB
Total sectors:	64 M
Total number of divided-partitions:	128 M
Total table size:	256 Mbit = 32 MB = 16 K sectors = 1/4 000 of total sectors

7.2.2 HDD

Sector size:	512 B
Total size:	2 TB
Total sectors:	4 G
Total number of divided-partitions:	8 G
Total table size:	16 Gbit = 2 GB = 4 MB sectors = 1/1 000 of total sectors

8 Applying CoPo2 to UDF

8.1 Storage media to be applied

The HDD recorder has an issue of fragmentation in the market with no tool to resolve the problem. To resolve the problem, apply the standard to HDD.

8.2 Basics when UDF volume format is applied to HDD

To conform to the existing HDD space management, apply the UDF in a partition of the HDD as an UDF volume following the existing HDD space management.

Existing HDD has a space addressed by logical block address (LBA). LBA 0 has the master boot record (MBR) for managing partitions. LBA 1 to 62 is reserved. Partitions managed by MBR are allocated after LBA 63.

LBA 63 is the starting point of UDF volume. The UDF volume address is managed by logical sector number (LSN) and starting from LSN 0. LSN 0 to 63 are reserved and UDF volume recognition descriptors are allocated from LSN 64 to 256. LSN 257 is a starting point of UDF partition and managed by logical block number (LBN) starting with LBN 0.

The relationship with LBA, LSN and LBN is as follows:

LBN = LBA − 63, LBN = LSN − 257.

8.3 Basics to apply management tables to UDF

8.3.1 Master divided-partition table

Set this information to the partition length of the partition descriptor.

8.3.2 Using the implementation use field of the partition descriptor

8.3.2.1 General

Set the following into the implementation use field of the partition descriptor:

Total number of divided partitions;

Highest divided-partition level management table;

Master divided-partition level management table;

LevelEndPositions.

8.3.2.2 Total number of divided-partitions

Reserve 8 byte (Unit64) field.

8.3.2.3 Highest divided-partition level management table

Maximum level number is 31. Reserve one byte field as MaximumLevel.

8.3.2.4 Master divided-partitions table

The maximum volume size that the UDF file system can cover as a partition is 2TB, 4G (512 byte) sectors.

The maximum sector number can be expressed by 4 bytes. The maximum number of divided-partition is double the maximum sector number. Reserve 8 bytes (Unit64) for each level. Maximum level is 31.

8.3.2.5 End position management table

Reserve 8 bytes (Unit64) for each level. Maximum level is 31.

8.3.3 Multilevel-divided-partition allocation table

UDF can cover 2TB as maximum. Based on the sector size 512 B, total sectors are 4 G, total divided-partitions are 8 G and the required number of bits to manage the UDF file system is 16 G (2 GB, 4 M sectors).

The UDF file system region is managed by space bitmap descriptor pointed from unallocated space descriptor.

9 Data structures applied to UDF

9.1 General

9.1.1 Entity identifier

```
struct EntityID {        /* ISO/IEC 13346-1/7.4  */

        Uint8   Flags;
        char    Identifier[23];
        char    IdentifierSuffix[8]; }
```

9.1.2 IdentifierSuffix

The format of the IdentifierSuffix field is dependent on the type of the Identifier. IdentifierSuffix in DomainIdentifier in logical volume descriptor specified in OSTA UDF2.01 is extended.

9.2 Volume structure

9.2.1 Logical volume descriptor

9.2.1.1 General

```
struct LogicalVolumeDescriptor{          /* ISO/IEC 13346-3/10.6 */

    struct tag          DescriptorTag;
    Uint32              VolumeDescriptorSequenceNumber;
    struct charspec     DescriptorCharacterSet;
    dstring             LogicalVolumeIdentifier[128];
    Uint32              LogicalBlockSize;
    struct EntityID     DomainIdentifier;
    byte                LogicalVolumeContentsUse[16];
    Uint32              MapTableLength;
    Uint32              NumberofPartitionMaps;
    struct EntityID     ImplementationIdentifier;
    byte                ImplementationUse[128];
    extent_ad           IntegritySequenceExtent;
    byte                PartitionMaps[];}
```

9.2.1.2 Domain identifier

This field shall indicate that the content of this logical volume conforms to the domain defined in this standard, therefore the DomainIdentifier shall be set to:

"*OSTA UDF Compliant"

9.2.1.3 Identifier suffix

9.2.1.3.1 General

Table 1 states the domain identifier suffix field format.

Table 1 – Domain identifier suffix field format

RBP	Length	Name	Contents
0	2	**UDF revision**	Unit16 （= #0500）
2	1	**Domain flags**	Unit8
3	5	**CoPo2 version**	Unit8
8	4	(Reserved)	bytes （= #00）

9.2.1.3.2 UDF revision

Set #0A00 (which means UDFA.00) specifying the starting number of CoPo2 applied UDF.

9.2.1.3.3 CoPo2 version

Set CoPo2 version.

9.2.1.3.4 Domain flags

Set the flag to apply CoPo2 bit map management for space, as shown in Table 2.

Table 2 – Domain flags

Bit	Description
0	Hard write-protect
1	Soft write-protect
2	Secure UDF given in secure universal disk format specification
3	CoPo2 bit map space management applied
4-7	Reserved

9.2.2 Logical volume integrity descriptor

9.2.2.1 General

```
struct LogicalVolumeIntegrityDesc{           /* ISO/IEC 13346-3/10.10 */
    struct tag          DescriptorTag;
    Timestamp           RecordingDateAndTime;
    Uint32              IntegrityType;
    struct extend_ad    NextIntegrityExtent;
    byte                LogicalVolumeContentsUse[32];
    Uint32              NumberOfPartitions;
    Uint32              LengthOfImplementationUse;
    Uint32              FreeSpaceTable[];
    Uint32              SizeTable[];
    byte                ImplementationUse[];}
```

9.2.2.2 Implementation use

9.2.2.2.1 General

Table 3 shows the implementation use format.

Table 3 – ImplementationUse format

RBP	Length	Name	Contents
0	32	ImplementationID	EntityID
32	4	Number of files	Uint32
36	4	Number of directories	Uint32
40	2	**Minimum UDF read revision**	Uint16 (= #0201)
42	2	**Minimum UDF write revision**	Uint16 (= #0500)
44	2	**Maximum UDF write revision**	Uint16 (= #0500)
46	??[a]	Implementation use	bytes
[a] ?? means variable length.			

9.2.2.2.2 Minimum UDF read revision

Shall indicate the minimum recommended revision of the UDF specification that an implementation is required to support to successfully be able to read all potential structures on the media. This number shall be stored in binary coded decimal format, for example #0150 would indicate revision 1,50 of the UDF specification.

9.2.2.2.3 Minimum UDF write revision

Shall indicate the minimum revision of the UDF specification that an implementation is required to support to successfully be able to modify all structures on the media. This number shall be stored in binary coded decimal format, for example #0150 would indicate revision 1,50 of the UDF specification.

9.2.2.2.4 Maximum UDF write revision

Shall indicate the maximum revision of the UDF specification that an implementation that has modified the media has supported. An implementation shall update this field only if it has modified the media and the level of the UDF specification it supports is higher than the current value of this field. This number shall be stored in binary coded decimal format, for example #0150 would indicate revision 1,50 of the UDF specification.

9.2.3 Partition descriptor

9.2.3.1 General

```
struct PartitionDescriptor {            /* ISO/IEC 13346-3/10.5 */

    struct tag          DescriptorTag;
    Uint32              VolumeDescriptorSequenceNumber;
    Uint16              PartitionFlags;
    Uint16              PartitionNumber;
    struct EntityID     PartitionContents;
    byte                PartitionContentsUse[128];
    Uint32              AccessType;
    Uint32              PartitionStartingLocation;
    Uint32              PartitionLength;
    struct EntityID     ImplementationIdentifier;
    byte                ImplementationUse[128]
    byte                Reserved[156];  }
```

9.2.3.2 PartitionContentsUse

Set the contents of the partition header descriptor in this field.

9.2.3.3 PartitionStartingLocation

The value of this field shall be an integral multiple of the HDD sector size.

9.2.3.4 PartitionLength

The value of this field shall be an integral multiple of the HDD sector size.

9.2.3.5 ImplementationIdentifier

Set the value “CoPo2 compliant”.

9.2.3.6 ImplementationUse

Set the contents of CoPo2 header descriptor.

9.3 File data structures

9.3.1 Partition header descriptor

9.3.1.1 General

```
struct PartitionHeaderDescriptor{          /* ISO/IEC 13346-4/14.3 */

    struct short_ad     UnallocatedSpaceTable;
    struct short_ad     UnallocatedSpaceBitmap;
    struct short_ad     PartitionIntegrityTable;
    struct short_ad     FreedSpaceTable;
    struct short_ad     FreedSpaceBitmap;
    byte                Reserved[88]; }
```

9.3.1.2 UnallocatedSpaceBitmap

Reserve the field and when implementing backward compatibility, create the latest values.

9.3.1.3 PartitionIntegrityTable

Shall be set to all zeros since PartitionIntegrityEntrys are not used.

9.3.2 CoPo2 partition header descriptor

9.3.2.1 General

```
struct PartitionHeaderDescriptor{

    struct short_ad     UnallocatedSpaceTable;
    struct short_ad     Multi level divided PartitionsStateBitmap;
    struct short_ad     PartitionIntegrityTable;
    struct short_ad        FreedSpaceTable;
    struct short_ad        FreedSpaceBitmap;
    struct short_ad        CoPo2ManageTableAd
    byte                   Reserved[80]; }
```

9.3.2.2 MultilevelDividedPartitionsStateBitmap

Manage each divided-partition usage state with two bits.

9.3.2.3 PartitionIntegrityTable

Shall be set to all zeros since PartitionIntegrityEntrys are not used.

9.3.2.4 CoPo2ManageTableAd

Set short_ad of CoPo2ManagementTable.

9.3.3 Space bitmap descriptor

9.3.3.1 General

```
SpaceBitmap{ /* ISO/IEC 13346-4/14.12 */
    struct Tag             DescriptorTag;
    Uint32                 NumberOfBits;
    struct      Uint32     NumberOfBytes;
    byte                   Bitmap[];}
```

9.3.3.2 DescriptorTag

9.3.3.3 CoPo2ManageTable

Table 4 shows the CoPo2 manage table.

Table 4 – CoPo2ManageTable

RBP	Length	Name	Contents
0	16	DescriptorTag use	DescriptorTag
16	32	TotalNumberOfDividedPartitions	EntityID
48	4	MuximumPartitonLevel	Uint32
52	256	MasterPartitionManagementTable	Uint64[32]
308	256	EndPositionNumberManagementTable	Uint64[32]
564	460	Reserve	

____