OTFormat

Specification

April 2021

Ver. 2.0.0

USAGE

Revision History

Revision	Date	Originator
1.0.0	September 20, 2020	FUJIFILM

Changes between v1.0.0 and v2.0.0

- Incremented specification version number to 2.0.0.
- Added definition of Pool Group name in Section 4.6.
- Added description of Pool Group ID and Pool ID for clustered system in Section 5.
- Added definition of object format in Section 8.
- Clarified that the first RCM does not contain PR Directory and may not contain System Info in Section 12.
- Added definition of Pool Group name for System Info in Section 12.

1. Introduction

This document is a specification of OTFormat. OTFormat is a standard for writing objects on tapes. Object storage provides scalable storage with parallel machines in the cloud environment. The purpose of OTFormat is to apply the design approach of object storage to tapes to realize scalable object storage where multiple tapes can be written on successively, and thus provide a storage solution suitable for cold data archiving.

2. Scope

This document defines requirements regarding OTFormat. Those requirements for OTFormat specify the size and the sequence of data blocks and file marks written on tape and define the content and the format of OTFormat labels and OTFormat references that are data structures specific to OTFormat. This document also defines various information stored in Medium Auxiliary Memory (MAM). It shall be ensured that OTFormat tapes can be interchangeable among all data storage systems that support this format.

3. Definition of Terms

3.1 Terms

- Big Endian: A way of storing the most significant byte of data at the smallest memory address.
- Block: A unit of data written or read on or from tapes by a tape drive.
- Bucket: A container for stored objects.
- Bucket ID: A UUID used to identify a bucket.
- Data Partition: A partition for storing object data.
- File Mark: A marker used to seek data written on a tape.
- Full Tape: A collective term of a series of data written on Data Partition.
- Object: A set of object data and metadata.
- Object Commit Marker: A marker that gathers the metadata and indicates the end of an Object Series.
- Object ID: A UUID used to identify an object.
- Object Series: Consists of at least one Packed Object and object commit marker.
- Pack ID: A UUID used to identify a Packed Object.
- Packed Object:　One or more objects packed.
- Partial Reference: A marker that gathers the metadata and indicates the end of a partial tape.
- Partial Tape: Consists of one or more Object Series and partial reference. A unit used when updating metadata on Reference Partition.
- Partition: Dividing a tape into one or more storage areas. Alternately, each of the divided areas.
- Pool: A group of one or more tapes. Used to write objects belonging to a specific bucket(s) onto multiple tapes dispersedly. Represents a single copy of a data set (objects).
- Pool Group: A group of one or more pools. Used to write a specific object to multiple pools dispersedly. Represents a set of copies of a particular data set (objects).
- Pool ID: A UUID used to identify a pool.
- Pool Group ID: A UUID used to identify a pool group.
- Reference: Metadata such as the positional information of an object written on a tape.
- Reference Commit Marker: A marker that　indicates the end of an entire tape.
- Reference Partition: A partition for storing object metadata.
- System ID: A UUID used to identify a system.
- Volume ID: A UUID used to identify a tape.

3.2 Abbreviation

・ ANSI	American National Standards Institute
・ ASCII	American Standard Code for Information Interchange

・	DNS	Domain Name System
・	FM	File Mark
・	ID	Identification
・	IP	Internet Protocol
・	JSON	JavaScript Object Notation
・	MAM	Medium Auxiliary Memory
・	NFC	Normalization Form Canonical Compression
・	NFD	Normalization Form Canonical Decompression
・	OCM	Object Commit Marker
・	OS	Operating System
・	PO	Packed Object
・	PR	Partial Reference
・	RCM	Reference Commit Marker
・	RFC	Request for Comments
・	RM	Reference Marker
・	SCSI	Small Computer System Interface
・	SPC	SCSI Primary Commands
・	SSC	SCSI Stream Commands
・	SSL	Secure Sockets Layer
・	UTC	Coordinated Universal Time
・	UTF-8	8-bit UCS/Unicode Transformation Format
・	UUID	Universally Unique Identifier
・	VCI	Volume Coherency Information
・	VCR	Volume Change Reference

4. Data Format

Data shall be stored in “Big Endian”. However, data stored in MAM complies with the SCSI standards.

4.1 JSON Format

JSON is an open standard file format defined in the RFC8259 standard. There are additional restrictions on OTFormat as below.

- ✧ Use UTF-8 NFC to encode characters.

- ✧ Format of vendor specific keys

  This specification shows standard keys in each JSON. Vendor specific keys’ names start with the prefix “Vendor” followed by the “T10 Vendor ID String” for uniquely identifying the key. The “T10 Vendor ID String” used to uniquely identify the key shall be converted into lower-case characters except for the first character before use.

  Example:

```
{
        "DataExample": {
                "VendorFujifilmSpecialNumber": 100,
                "VendorFujifilmSpecialString": "for example"
        }
}
```

4.2 UUID Format

In OTFormat, a UUID is used frequently as a unique key. UUID version 4 standardized in RFC4122 shall be used as a unique key.

Example:

```
{
        "DataExample": {
                "VolumeUuid": "1c863534-f611-4404-8d4f-a414b8cadf4a"
        }
}
```

4.3 Time Format

In OTFormat, the extended time format specified in ISO 8601 is used. The time shall be in UTC and “Z” shall be added directly after the time without a space. Decimal fraction of second shall be six numbers.

Example:

```
{
    "DataExample": {
        "FormatTime": "2018-03-01T18:35:47.866846Z"
    }
}
```

4.4 Creator Format

In OTFormat, creator value shall be in ASCII with up to 1024 characters. The creator value shall include a product name, an OS and the name of an executable file that wrote the OTFormat tape. A recommended creator value is a character string that includes a product name, an OS and the name of an execution file delimited by “ - “ (space shall exist on both sides of the hyphen).

Example:

```
{
    "DataExample": {
        "Creator": "FUJIFILM SDT System 2.0.0 – Red Hat Linux release 7.7 – sdt-taped"
    }
}
```

4.5 Bucket Name Limitations

A bucket name shall be formed by one or more labels that comply with the DNS naming convention. The specific restrictions are as below.

- A bucket name consists of 3 or more characters but not exceeding 63 characters.
- The types of permitted characters are lower-case alphabets, numbers, dots and hyphens.
- Upper-case characters and underscores cannot be used.
- The beginning and end of a bucket name are limited to alphabets and numbers (periods and hyphens cannot be used).
- Continuous periods and a period and a hyphen side by side cannot be used.
- A bucket name must not be formatted as an IP address.

A bucket name containing a period makes it impossible to use Secure Sockets Layer (SSL) because an SSL wildcard certificate matches the bucket name not containing a period. It is recommended not to use a period in a bucket name.

4.6 Pool Group Name Limitations

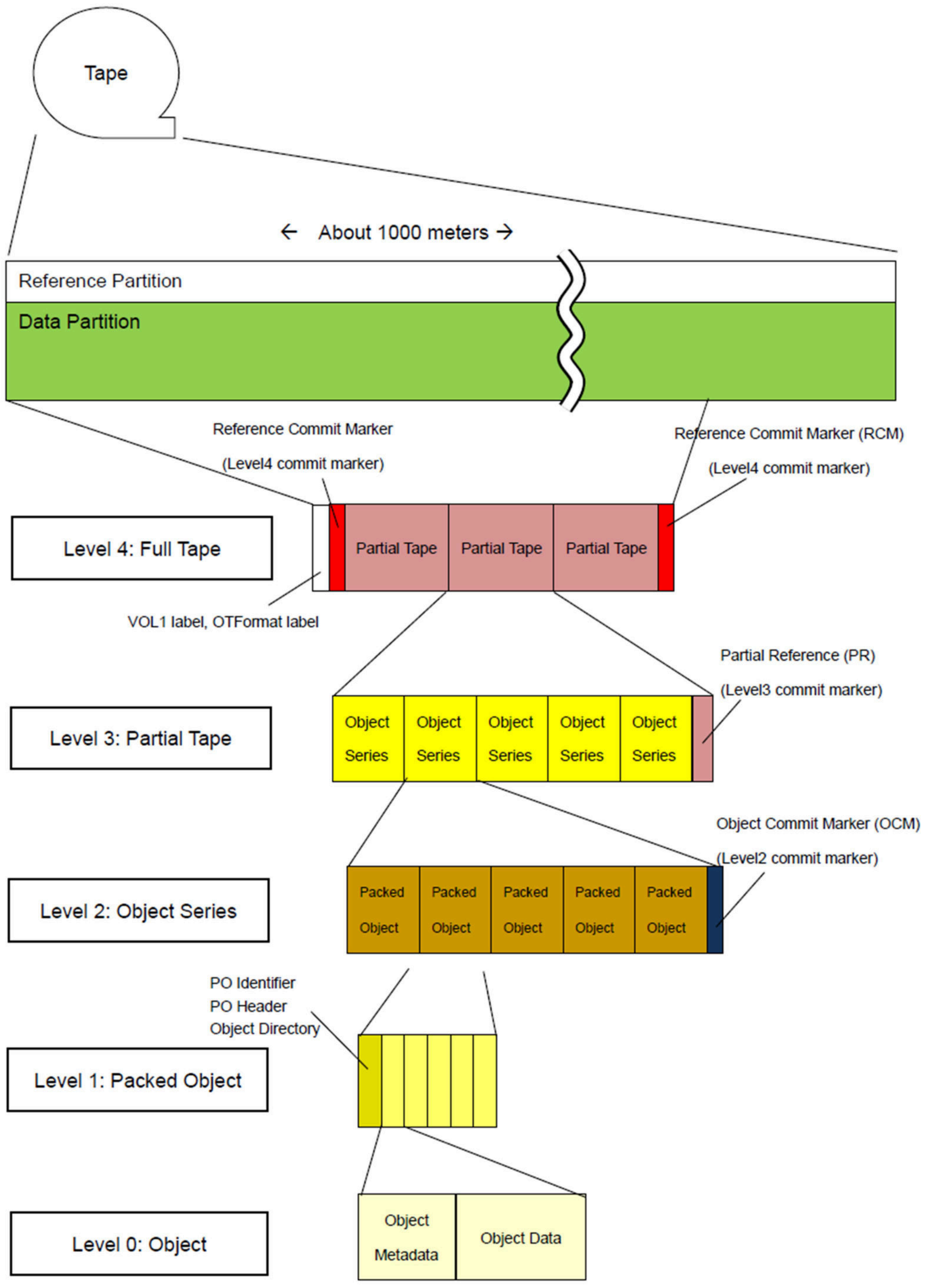
A pool group name shall be a label that complies with the DNS naming convention. The specific restrictions are as below.

- A pool group name consists of 1 or more characters but not exceeding 63 characters.
- The types of permitted characters are lower-case alphabets, upper-case alphabets, numbers and hyphens.
- The beginning of a pool group name is limited to alphabets (numbers and hyphens cannot be used).
- The end of a pool group name is limited to alphabets and numbers (hyphens cannot be used).
- A pool group name is not case-sensitive.

5. Overview of OTFormat

OTFormat tape can contain one billion or more objects in a hierarchical structure so that the objects can be written and read within reasonable time. Each level contains block offset to access data contained in lower levels, which makes it possible to easily move data to another tape.

OTFormat uses a tape by logically dividing it into Reference Partition and Data Partition. The following figure is a conceptual diagram of Data Partition. File marks and other details are omitted. In addition, the end of Full Tape does not match the end of Data Partition because Data Partition is not always full.



In OTFormat, a Packed Object that contains one or more objects is the smallest unit for accessing the tape.

To ensure that one or more Packed Objects has been written on a tape, an Object Commit Marker is written to commit the data. A group of Packed Objects written between Object Commit Markers is called an Object Series.

To increase the speed of searching objects, a Partial Reference is stored in Reference Partition and Data Partition. The Partial Reference includes OCM info that is the core of the Object Commit Marker written on the partial tape. Dividing reference data into Partial References and writing to tape helps avoid the volume of reference data that is written at a time from becoming excessively large.

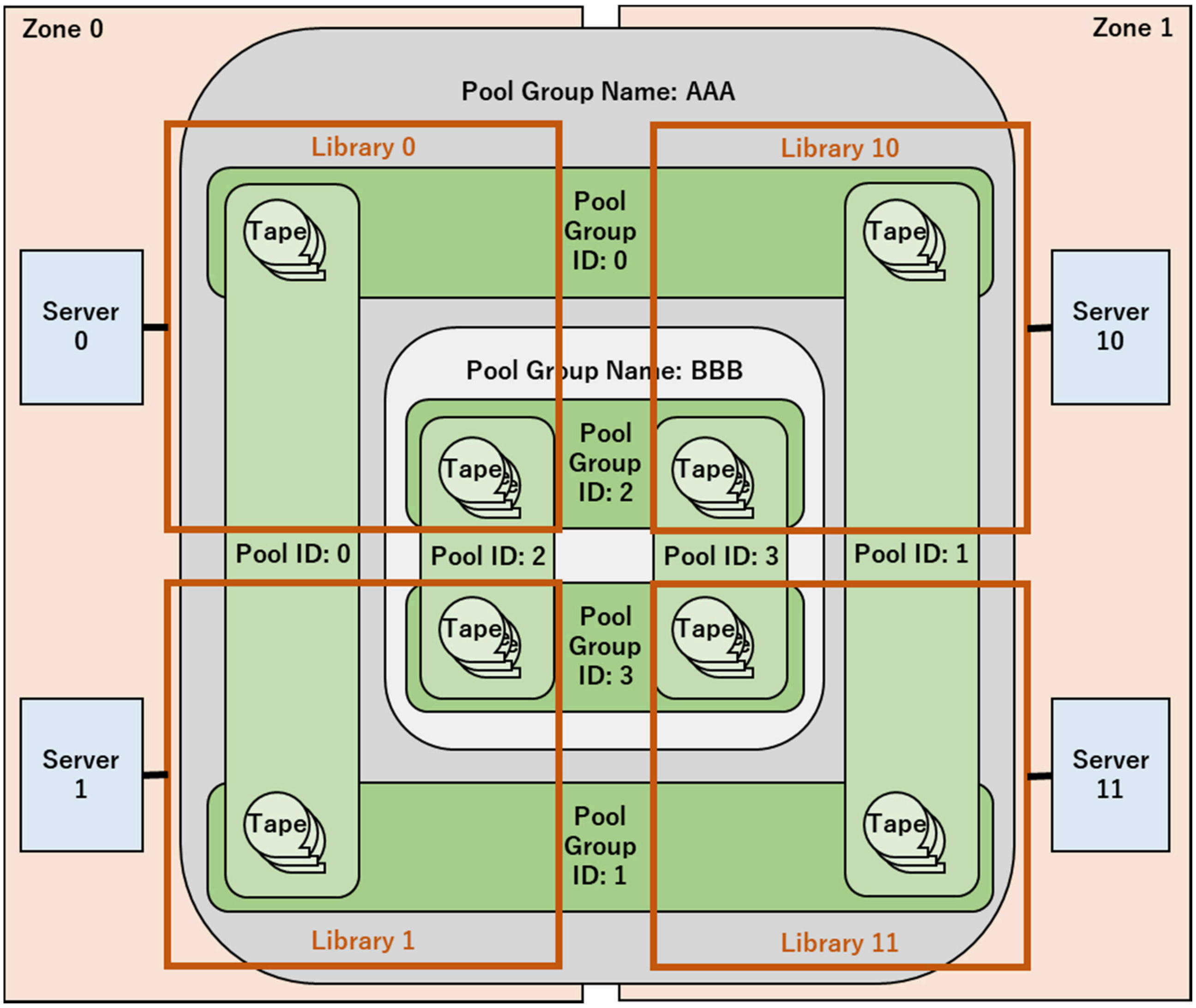
An area separated by Partial References is repeatedly written, then a Reference Commit Marker is written at the end of the partition.

Commit markers exist at each of levels 2 or above. A commit marker is designed to have a file mark at the beginning and the end. These file marks are introduced to find a commit marker as a clue.

In OTFormat, one or more tapes are grouped. Each of the tape groups is called a pool. A OTFormat tape contains a Pool ID, which indicates the pool the tape belongs to. If an OTFormat tape contains unknown Pool ID, writing objects to the tape shall not be permitted though reading objects from that tape is permitted.

In OTFormat, one or more pools are grouped. Each group of the pools is called a pool group. An OTFormat tape contains a Pool Group ID, which indicates the pool group the pool belongs to. A Packed Object shall be written to all pools belonging to a pool group in order to ensure data redundancy.

If a system is formed of multiple servers, each server shall use the same System ID. In this case, each server shall use a unique Pool Group ID in a zone. However, each server in a zone can use the same Pool IDs to identify that a tape belongs to the same pool or not. If a system contains multiple zones, one of servers in each zone can use the same Pool Group ID. By the combination of Pool Group ID and Pool ID, a system can determine a zone, server, pool group and pool to which the tape belongs. Tapes which Pool ID or Pool Group ID is the same shall have the same Pool Group Name.

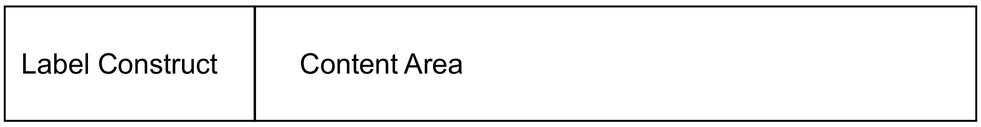


6. Tape Layout

An OTFormat tape shall consist of Reference Partition and Data Partition. The first physical partition (partition 0) shall be Reference Partition and the second physical partition (partition 1) shall be Data Partition.

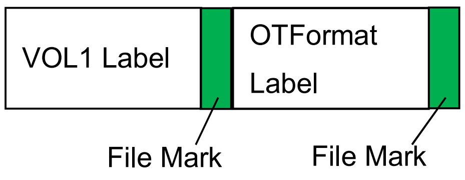
6.1 OTFormat Partitions

Each OTFormat partition shall consist of a Label Construct, followed by a Content Area. This logical structure is shown in the following figure.



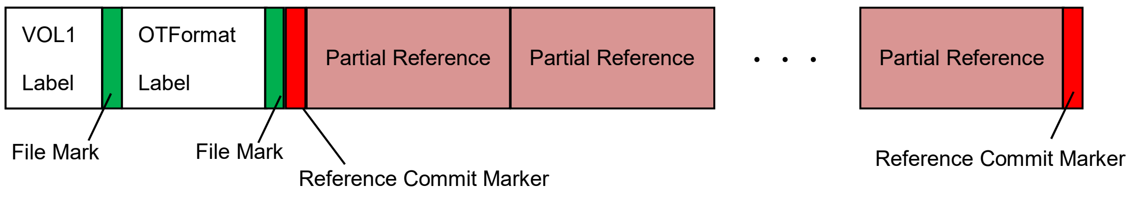
6.2 Label Construct

Label Construct shall consist of a VOL1 label, followed by a file mark, followed by an OTFormat label, followed by a file mark. The Label Construct is shown in the following figure. Refer to Section 7. Label Format for VOL1 label format and OTFormat label format.



6.3 Reference Partition Construct

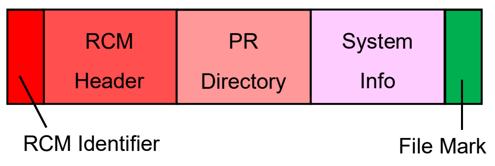
Reference Partition shall consist of a Label Construct, followed by the first Reference Commit Marker, followed by one or more Partial Reference, followed by the last Reference Commit Marker.



6.4 Reference Commit Marker

A Reference Commit Marker (RCM) is shown in the following figure. An RCM stores information on where the tape belongs. The marker is written immediately after the Label Construct and at the end of a partition. The RCM at the end of a partition is overwritten every time data is added to the partition.

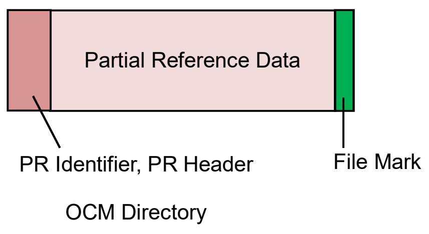
Refer to Section 12. Reference Commit Marker Format (Level 4 Commit Marker) for details.



6.5 Partial Reference Layout

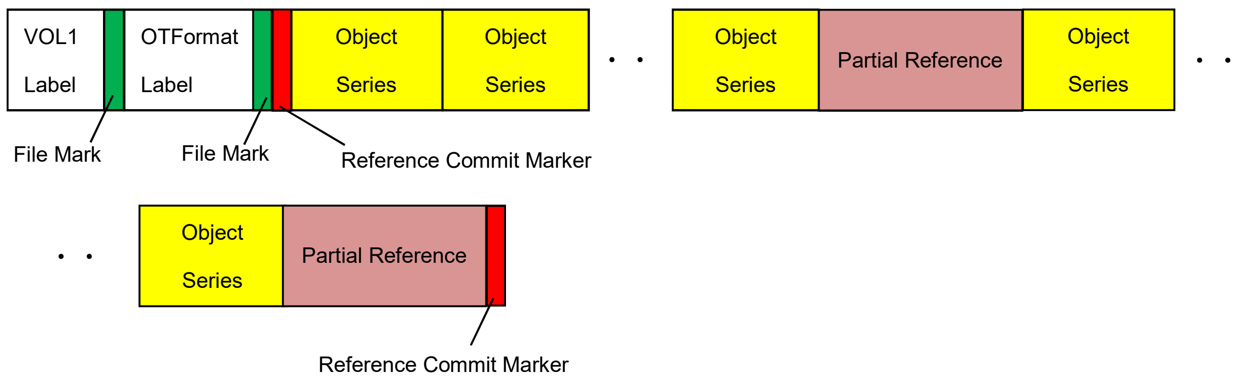
A Partial Reference (PR) is shown in the following figure. A reference is a pair of object metadata and block offset on the tape. A PR is a set of references of objects written in the Partial Tape.

Refer to Section 11. Partial Reference Format (Level 3 Commit Marker) for details.



6.6 Data Partition Construct

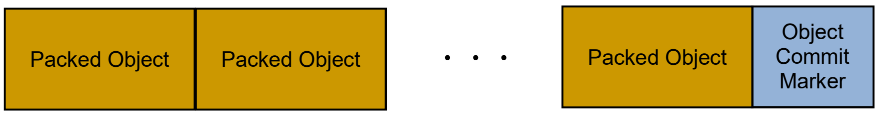
A Data Partition is as shown in the following figure.



A PR written on Data Partition is the same as the PR on Reference Partition. When an access error occurs with a PR on the Reference Partition, a PR on Data Partition is helpful to reduce the time to get objects' metadata and logical positions on the tape.

6.7 Object Series Layout

An Object Series is as shown below.



An Object Series should contain several hundred to several thousands of Packed Objects. An Object Commit Marker shall be written at the end of an Object Series. An Object Commit Marker shall contain object metadata of all objects in the Packed Objects included in the Object Series. Refer to Section 10 Object Commit Marker Format (Level 2 Commit Marker) for details.

6.8 Block Offset

The last RCM contains all PRs' block offsets from the beginning of the last RCM to the beginning of each PR in the Full Tape in the Data Partition. A PR contains block offsets from the beginning of the PR to the beginning of each OCM for all Object Series in the Partial Tape. An OCM contains block offsets from the beginning of the OCM to the beginning of each PO for all POs in the Object Series.

An example of a block offset is illustrated in Appendix A.

By using a logical block number of the last RCM in the Data Partition and block offsets in PRs and RCM written in Reference Partition, a logical block number for each Packed Object on the Data Partition can be calculated.

Using block offsets instead of logical block numbers in RCM, PR and OCM makes it possible to move data by copying byte sequence when the data is to be moved to another tape.

Logical block numbers of the last RCM on both partitions are stored in the MAM . In OTFormat, MAM is the only space to store logical block numbers on tapes.

6.9 Back Pointer

A back pointer is a way of quickly recovering data in case of a tape error. Since the tape is written serially, if an error occurs, data has to be recovered and read by moving back from the point where the data was written last. Tracing metadata and reference data at high speed with the back pointer can reconstruct reference information at high speed.

In OTFormat, a file mark can be used as a back pointer. A typical flow of error recovery and reading process is shown below.

Firstly, the logical block numbers of RCMs at the ends of both Reference Partition and Data Partition are stored in the MAM. In case of a tape error, data in the MAM cannot be trusted. It is therefore necessary to move to the EOD of the tape, find a file mark while returning and to determine which data exists before the file mark.

In the case of Reference Partition, only PRs are placed. Once the last PR is found, data can be read by always going back one file mark if a back pointer is to be traced. (The SCSI commands of “SPACE FM -2”, “SPACE FM +1” and “READ” may be performed.)

In the case of Data Partition, data can be read by going back file marks by twice the number of Object Commit Markers in between (“number of OCMs” in the header) plus one file mark if PRs are to be traced. (If there are 1,000 OCMs, for example, the SCSI commands “SPACE FM 2002”, “SPACE FM +1” and “READ” may be performed.)

If an object commit marker is to be traced, data can be read by always going back two file marks. (The SCSI commands of “SPACE FM -3”, “SPACE FM +1” and “READ” may be performed.)

6.10 Typical Tape Layout

Immediately after formatting.
Reference Partition:
[VOL1][FM][OTF][FM]
Data Partition:
[VOL1][FM][OTF][FM]

Immediately after assigning to pool.
Reference Partition:
[VOL1][FM][OTF][FM][RCM][FM][RCM][FM]
Data Partition:
[VOL1][FM][OTF][FM][RCM][FM][RCM][FM]

Immediately after writing objects.
Reference Partition:
[VOL1][FM][OTF][FM][RCM][FM][RCM][FM]
Data Partition:
[VOL1][FM][OTF][FM][RCM][FM][PO]

Immediately after committing.

Reference Partition:

[VOL1][FM][OTF][FM][RCM][FM][RCM][FM]

Data Partition:

[VOL1][FM][OTF][FM][RCM][FM][PO][FM][OCM][FM]

At completion.

Reference Partition:

[VOL1][FM][OTF][FM][RCM][FM][PR][FM][RCM][FM]

Data Partition:

[VOL1][FM][OTF][FM][RCM][FM][PO][FM][OCM][FM][PR][FM][RCM][FM]

Legend

[VOL1]: VOL1 Label

[FM]: File Mark

[OTF]: OTFormat Label

[RCM]: Reference Commit Marker

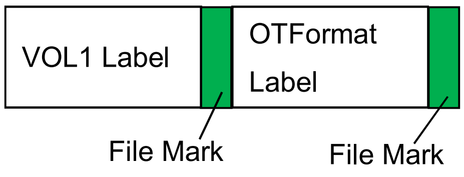
[PO]: Packed Object

[OCM]: Object Commit Marker

[PR]: Partial Reference

7. Label Format

Each partition in an OTFormat tape shall contain a Label Construct. A Label Construct shall consist of a VOL1 label, followed by a file mark, followed by an OTFormat label, followed by a file mark as shown in the following figure. Other data shall not be contained. The Label Construct shall be written at the beginning of each partition. The two partitions in OTFormat each include a Label Construct that contains the same information.



7.1 VOL1 Label

The first block in each partition shall contain an 80-byte VOL1 label that is compliant with ANSI Standard X3.27. All bytes in the VOL1 label are stored as ASCII encoded characters and their content is shown in Table 1. The implementation identifier “OTFormat” is an identifier that indicates an OTFormat format.

Table 1 — VOL1 Label Construct

Offset	Length	Name	Value	Notes
0	3	label identifier	‘VOL’	
3	1	label number	‘1’	
4	6	volume identifier	<volume serial number>	Typically matches the physical cartridge label.
10	1	volume accessibility	Space	
11	13	Reserved	All spaces	
24	13	implementation identifier	‘OTFormat’	Value is left-aligned and padded with spaces to length.
37	14	owner identifier	right pad with spaces	Any printable characters. Typically reflects some user specified content oriented identification.
51	28	Reserved	All spaces	
79	1	label standard version	‘4’	

Note 1: Should not record the single quotation marks written in the Value column on VOL1 label.

Note 2: All values excluding the volume identifier and the owner identifier must contain the constant values shown in the table above.

7.2 OTFormat Label

An OTFormat label is described in JSON format defined in section 4.1. The OTFormat label contains information about the OTFormat tapes. An example of the OTFormat label is shown here.

```
{
    " OTFormatLabel": {
        "Version": "2.0.0",
        "FormatTime": "2018-03-01T18:35:47.866846Z",
        "VolumeUuid": "1c863534-f611-4404-8d4f-a414b8cadf4a",
        "Creator": "FUJIFILM SDT System 2.0.0 - Red Hat Linux release 7.7 -
sdt-taped",
        “BlockSize”: “1048576”,
        "Compression": true
    }
}
```

✧ **OTFormatLabel (Mandatory):**

This object defines the structure of an OTFormat label.

✧ **Version (Mandatory):**

This string shall contain a version of OTFormat in use.

✧ **FormatTime (Mandatory)**:

This shall contain the time when this tape was formatted in OTFormat. It shall conform to the “time format” defined in section 4.3.

✧ **VolumeUuid (Mandatory):**

This string shall contain a UUID value for uniquely identifying the OTFormat tape. It shall conform to the “UUID format” defined in section 4.2.

✧ **Creator (Mandatory):**

This string shall contain information to uniquely identify the writer of the OTFormat tape. It shall conform to the “creator format” defined in section 4.4.

✧ **BlockSize (Optional):**

This string specifies the block size used when writing Packed Objects in the Content Area in Data Partition. The data type of BlockSize is not number but string. This string value shall be an integer value that is formed of 8-bit bytes and the default value is 1048576 bytes (1 MiB) unless specified.

The minimum BlockSize that may be used in an OTFormat tape is 4096 8-bit bytes. All blocks of Packed Objects shall have this fixed block size. If a Packed Object is divided by the block size and the last fragment is smaller than the block size or if a Packed Object is smaller than the block size, the end of the block shall be padded with 0.

✧ **Compression (Optional):**

This boolean shall be true or false to specify whether data compression on a tape drive is enabled or not. If true is specified, compression is enabled when writing an OTFormat tape. If false is specified, compression is disabled. If not specified, the default value is true.

8. Object Format (Level 0)

Typically, an object in object storage consists of object data in binary format and object metadata in JSON format. In some cases, size of object data is zero. It means that an object may contain object metadata only.

An example of object metadata is shown here.

```
{
        "MetadataVersion": 1,
        "Key": "key",
        "Size": 104857600,
        "LastModifiedTime": "2020-03-31T01:44:56.308724Z",
        "IsDeleted": false,
        "Version": "98414380903688999999RG001    87241.0",
        "UserMetadata": {
                "key1": "value1",
                "key2": "value2"
        }
}
```

✧ **MetadataVersion (Mandatory):**

This number shall contain a version of object metadata format. The value shall be 1.

✧ **Key (Mandatory):**

This string shall contain an object key.

✧ **Size (Mandatory):**

This number shall contain an object size in byte. When the object is deleted, the value shall be 0.

✧ **LastModifiedTime (Mandatory):**

This string shall contain last modified date of the object. It shall conform to the “time format” defined in section 4.3.

✧ **IsDeleted (Optional):**

This Boolean shall be true for a delete request. Otherwise, the value shall be false. The default value is false.

✧ **ContentEncoding (Optional):**

This string shall contain a content encoding applied to the object. For more information, see https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11

✧ **ContentType (Optional):**

This string shall contain a content type of the object. For more information, see https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17

✧ **ContentMd5 (Optional):**

This string shall contain a base64-encoded 128-bit MD5 digest.

✧ **ContentLanguage (Optional):**

This string shall contain the language(s) the object is in. For more information, see https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.12

✧ **CreationTime (Optional):**

This string shall contain a creation date of the object. It shall conform to the "time format" defined in section 4.3.

✧ **ObjectRecipe (Optional):**

This array shall contain JSON objects to show what has been done for the object data. When the first modification is applied to the object data, it shall be added to the beginning of the array. Then the next modification is applied to the object data, it shall be appended at the end of the array like as follows: ObjectRecipe: [{"ServerSideCompression":"GZIP"},{"ServerSideEncryption":"AES256"}]

✧ **ServerSideCompression (Optional):**

This string shall contain the server side compression algorithm.

✧ **ServerSideEncryption (Optional):**

This string shall contain the server side encryption algorithm. An encryption key shall be managed by the server.

✧ **ServerSideEncryptionKeyId (Optional):**

This string shall contain an encryption key ID to specify an encryption key managed by the server.

✧ **ServerSideEncryptionCustomer (Optional):**

This string shall contain the server side encryption algorithm. An encryption key shall be provided by the customer.

✧ **UserMetadata (Optional):**

This object shall contain tags. Each tag is a key-value pair.

- ✧ **Version (Optional):**

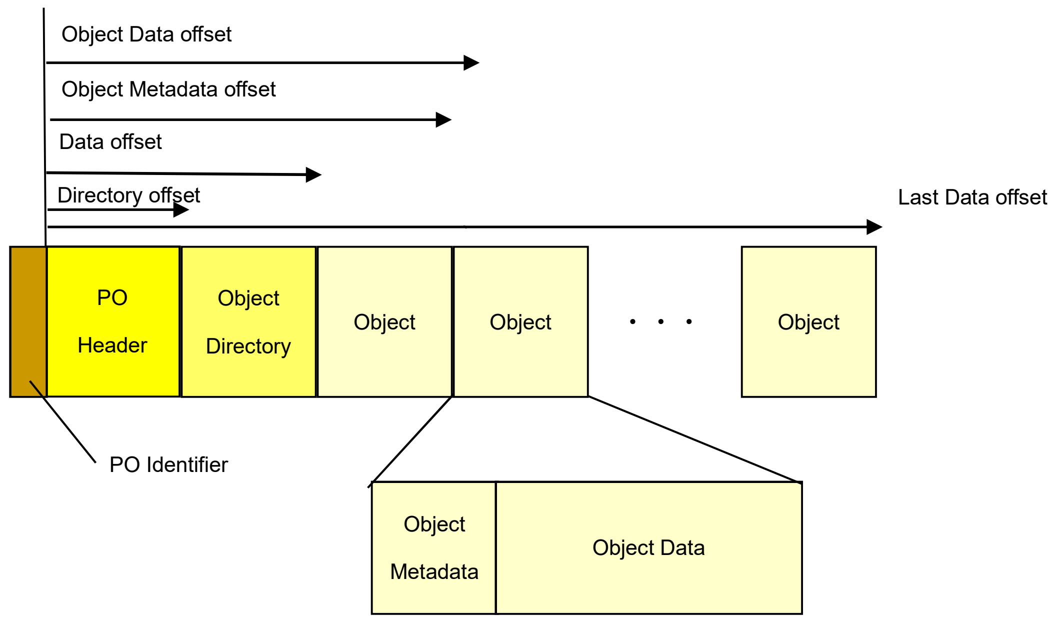
This string shall contain an object version used to reference a specific version of the object.

9. Packed Object Format (Level 1)

A Packed Object (PO) shall contain one or more objects that belong to the same bucket. It is recommended to pack multiple objects if the object size is 10 GiB or less, provided the total object data size in a PO shall be 10 GiB or less. The number of objects stored in a PO shall be 100,000 or less.

As shown in the following figure, a PO shall consist of a PO identifier, followed by a PO header, followed by an object directory, followed by one or more objects. If all these data are containable in one block, they shall be contained in one block. If a block is partially empty, the space from the end of the last object to the end of the block shall be padded with 0. PO identifier of each PO shall be at the beginning of a block.

Each PO shall contain a unique Pack ID, a Bucket ID to which the objects contained in the PO belong, and a System ID in which the PO was created.



A PO identifier shall be a 32-bytes length static string and used as a landmark of the beginning of a PO. The value of the identifier shall be

OTFormat 1.0 Level1

in ACSII codes. The value shall be left-aligned and shall be padded with space (20h).

A PO header shall consist of the following contents.

Offset	Length	Name	Notes	Sample Value
0	8	Directory offset	Number of bytes from the end of the identifier to the directory.	72
8	8	Data offset	Number of bytes from the end of the identifier to the first object.	296
16	8	Number of Objects	Total number of contained objects.	7
24	16	Pack ID	UUID of the PO.	ea03d783-248e-40e0-8101-a27870d47c91
40	16	Bucket ID	UUID of a bucket to which objects contained in the PO belong.	0339a0c1-0dde-4f87-8dcd-6879e0fec42c
56	16	System ID	UUID of a system that generate the PO.	811c7178-7859-4675-895b-38ff181ae28a

- The directory offset and the data offset shall be the number of bytes from the beginning of the PO header. They shall not include the size of the identifier.

The number of Object Directories is that of POs plus one. The Nth directory has the following contents. (N is 1-based.)

The last Object Directory (the “number of objects + 1”th directory) shall contain special data, which indicates the end of the last object. Object ID shall be filled by 0. The Object Metadata offset and the Object Data offset shall be the Last Data offset which is the number of bytes from the beginning of the PO header to the end of the PO.

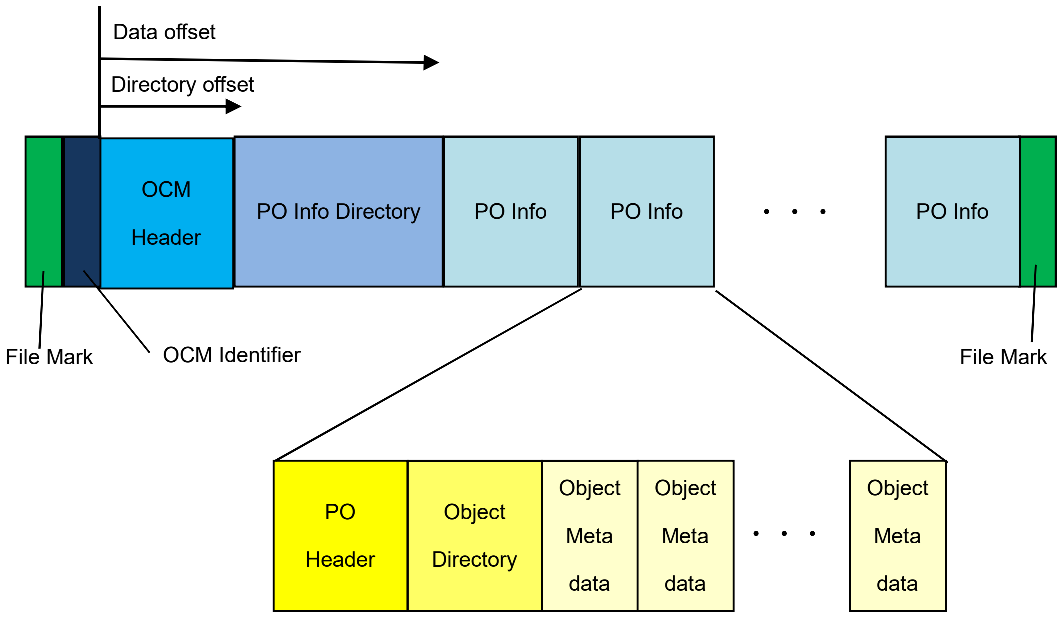
Offset	Length	Name	Notes	Sample Value
(N-1)*32	16	Object ID	The object’s UUID.	d08f5d8c-cf30-4cc9-a478-0dfeefd1 be28
(N-1)*32 +16	8	Object Metadata offset	Number of bytes from the beginning of the PO header the beginning of the object metadata.	450
(N-1)*32 +24	8	Object Data offset	Number of bytes from the beginning of the PO header to the beginning of the object data..	102400

10. Object Commit Marker Format (Level 2 Commit Marker)

An Object Commit Marker (OCM) is a marker of an Object Series.
Committing is to declare that data which was written before writing this marker on the tape is reliable. If an error occurs while data is being written, data that was requested to be written may not be completely written on the tape due to the existence of a buffer in the drive. When writing a file mark at the end of the commit marker, a process called flush for writing data accumulated in the drive buffer onto the tape is implemented. This can ensure that the data requested to be written has been written on the tape before the commit marker is written. When recovering data in case of a tape error, the OCM can be used as an important mark.

An OCM should contain several hundreds or several thousands of PO Info. When an OCM is written, the drive buffer may be flushed, which suspends continuous writing to the tape and decreases the transfer rate of writing. Therefore, OCMs should not be written frequently.

As shown in the following figure, an OCM shall consist of a file mark, followed by an OCM identifier, followed by an OCM header, followed by a PO Info Directory, followed by PO Infos of POs written in the Object Series. PO info shall consist of a PO header, followed by an Object Directory, followed by Object Metadata of objects written in the PO. In other words, PO Info is PO without PO identifier nor object data.



An OCM identifier shall be a 32-byte length static string and used as a landmark of the beginning of an OCM. The value of the identifier shall be

OTFormat 1.0 Level2

in ACSII codes. The value shall be left-aligned and shall be padded with space (20h).

An OCM header shall consist of the following contents.

Offset	Length	Name	Notes	Sample Value
0	8	Directory offset	Number of bytes from the end of the identifier to the directory.	24
8	8	Data offset	Number of bytes from the end of the identifier to the first PO Info.	136
16	8	Number of Packed Objects	Total number of POs contained in the Object Series.	7

- The directory offset and the data offset shall be the number of bytes from the beginning of the OCM header. They shall not include the size of the identifier.

The PO Info Directory shall contain information for each PO. The Nth directory shall contain the following contents. (N is 1-based.)

Offset	Length	Name	Notes	Sample Value
(N-1)*16	8	Packed Object Info Length	Number of bytes of the PO Info.	3528
(N-1)*16 +8	8	Block offset	Number of blocks from the block containing the OCM identifier written on Data Partition to the block containing the PO identifier written on Data Partition.	3

- Although the PR and RCM do not contain a file mark at the beginning of them, the OCM contains a file mark not only at the end but also at the beginning.

11. Partial Reference Format (Level 3 Commit Marker)

A partial reference (PR) is a marker of Partial Tape. A PR shall contain reference data which consists of block offsets and object metadata of all objects written in the Partial Tape. The same PR shall be written in the Data Partition and Reference Partition.

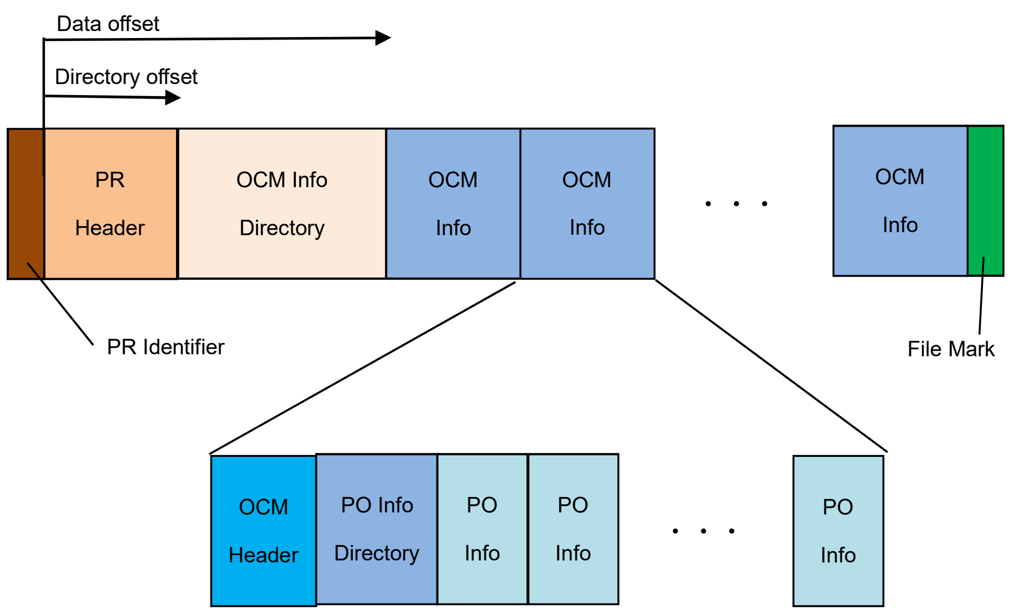
Reference data is used to access an object directly. When an OTFormat tape contains 100 million to 1 billion objects, the reference data size may be over 100 GB. If the transfer rate is 200 MB/second, it will take at least 15 minutes to write the reference data in both partitions for unloading the OTFormat tape. The PR was introduced to reduce the time to unload the OTFormat tape by writing reference data on a piecemeal basis.

When a tape is unloaded, an OCM, a PR and an RCM are written at the end of the Data Partition and the tape is rewound to move to the Reference Partition, then PR and a RCM overwrites the last RCM in the Reference Partition.

The reason why the PR is recorded on both partitions is to increase reliability. Writing a PR in the middle of Data Partition can reduce the time to recover data even when a PR in the Reference Partition cannot be read.

A PR should contain about 1,000 OCM Info. Because partition change is required to write PR, frequent writing PR invokes performance degradation. The procedures to write a PR on a tape shall be write an OCM in the Data Partition, and then write a PR just after the OCM in the Data Partition, and then write an RCM just after the PR in the Data Partition, and then write the same PR in the Reference Partition, and then write the same RCM just after the PR in the Reference Partition.

As shown in the following figure, a PR shall consist of a PR identifier, followed by a PR header, followed by an OCM info directory, followed by OCM Infos of OCMs written in the Partial Tape. An OCM Info shall contain OCM Header, followed by a PO Info Directory, followed by PO Infos. In other words, OCM Info is OCM without OCM Identifier nor file marks.



A PR identifier shall be a 32-byte length static string and used as a landmark of the beginning of a PR. The value of the identifier shall be

OTFormat 1.0 Level3

in ACSII codes. The value shall be left-aligned and shall be padded with space (20h).

A PR header shall consist of the following contents.

Offset	Length	Name	Notes	Sample Value
0	8	Directory offset	Number of bytes from the end of the identifier to the directory.	24
8	8	Data offset	Number of bytes from the end of the identifier to the first OCM Info.	1624
16	8	Number of OCM	Total number of OCMs contained in the Partial Tape.	1000

- The directory offset and the data offset shall be the number of bytes from the beginning of the PR header. They shall not include the size of the identifier.

The OCM Directory shall contain information of all OCMs written in the Partial Tape. The Nth directory shall contain the following contents. (N is 1-based.)

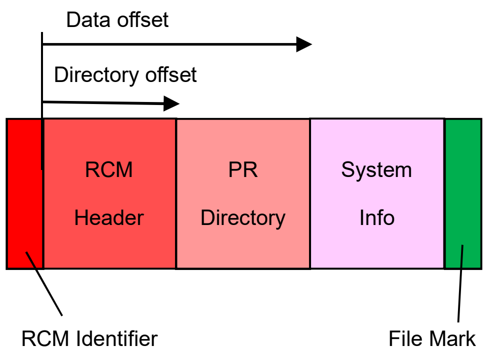
Offset	**Length**	**Name**	**Notes**	**Sample Value**
(N-1)*16	8	OCM Info Length	Number of bytes of the OCM info.	2048
(N-1)*16 +8	8	Block Offset	Number of blocks from the block containing the PR identifier written on Data Partition to the block containing the OCM identifier written on Data Partition.	3

12. Reference Commit Marker Format (Level 4 Commit Marker)

A reference commit marker (RCM) is a marker of a Full Tape. An RCM is written at the beginning of a Full Tape and the end of a Full Tape.

The first RCM and the last RCM shall be written at the same time in the Reference Partition and then the Data Partition when the tape is assigned to a pool. The RCM shall contain the IDs of a system, Pool and Pool Group to which the tape belongs. Pool ID in the first RCM in Reference Partition will be used to determine if the OTFormat tape can be used in the system. The last RCM shall be written in the Data Partition and then the Reference Partition when the tape is unloaded after writing a PO on the tape. When additional POs are written on a tape, the last RCM in both partitions shall be overwritten.

As shown in the following figure, an RCM shall consist of an RCM Identifier, followed by an RCM Header, followed by a PR Directory, followed by a System Info, followed by a file mark. At writing RCM, if there is no PR on the tape, RCM does not contain PR Directory because the number of PR in RCM header is 0. In this case, System Info may be empty i.e. Data Length in RCM header may also be 0.



An RCM identifier shall be a 32-byte length static string and used as a landmark of the beginning of an RCM. The value of identifier shall be

OTFormat 1.0 Level4

in ACSII codes. The value shall be left-aligned and shall be padded with space (20h).

An RCM header shall consist of the following contents.

Offset	Length	Name	Notes	Sample Value
0	8	Directory offset	Number of bytes from the end of the identifier to the directory.	80
8	8	Data offset	Number of bytes from the end of the identifier to the system info.	104
16	8	Data Length	Number of bytes of the system info.	1024
24	8	Number of Partial Reference	Total number of PRs in the partition.	3
32	16	System ID	UUID of a system which writes the RCM.	811c7178-7859-4675-895b-38ff181ae28a
48	16	Pool ID	UUID of a pool to which the tape belongs.	fce0d61d-8f54-4ca4-8d6e-18ee5b80e553
64	16	Pool Group ID	UUID of a pool group to which the pool belongs.	47122cc3-ba6f-44df-b83c-cf730f2d0c41

- The directory offset and the data offset shall be the number of bytes from the beginning of the RCM header. They shall not include the size of the identifier.
- The number of partial references in the first RCM shall be zero.

The PR Directory of the last RCM shall contain block offsets of all PRs written in Data Partition. The Nth directory has the following content. (N is 1-based.)

Offset	Length	Name	Notes	Sample Value
(N-1)*8	8	PR Block	Number of blocks from the block containing the identifier of the RCM written on the Data Partition to the block containing the identifier of the PR written on the Data Partition.	4

The System Info shall contain a BucketList array for buckets to which each PO written on the tape belong. Each JSON object in the BucketList array shall consist of a BucketName string and a BucketID string which store the name and UUID of a bucket. The System Info may contain a PoolGroupName string for a name of a pool group to which the tape belongs.

```
{
  "BucketList": [
    {
        "BucketName": "example-bucket1",
        "BucketID" : " 2ac28943-5c12-4e16-968d-eeb4e1f508bb"
    },
    {
        "BucketName": "example-bucket2",
        "BucketID" : " 0cc2ce97-39fa-4480-bffb-05e0bba07f2d"
    },
    {
        "BucketName": "example-bucket3",
        "BucketID" : " c6ac0737-c482-46bb-8a2d-6027027b85c1"
    }
  ],
  “PoolGroupName”: “example-pool-group1”
}
```

13. MAM (Medium Auxiliary Memory)

In an OTFormat tape cartridge, auxiliary information may be stored in Medium Auxiliary Memory (MAM) specified in the SCSI standards. The MAM is used only to improve efficiency to read the RCM. Even if the MAM attributes become inaccessible or are not updated, the OTFormat tape may still be correctly written and read.

13.1 Volume Change Reference (0009h)

Refer to SCSI standards SSC-4 or later for details of Volume Change References (VCR) (ID: 0009h).

The VCR attribute is used to indicate that records on tape media have been changed. VCR is a non-repeating, unique value associated with a volume coherency point.

13.2 Volume Coherency Information (080Ch)

Refer to SCSI standards SSC-4 or later for details of Volume Coherency Information (VCI) (ID: 080Ch).

The VCI is provided for each partition. Each partition's VCI contains VCR, the number of PRs and a logical block number of RCMs on the tape. Using these values, it is possible to check coherency of these partitions without reading any data recorded on the partitions.

- VCR: this field contains the value of VCR (0009h) at the time when a RCM has completely been written.
- Number of Partial Reference: this field contains the number of PRs written in the partition.
- Block Number: this field contains the logical block number of the block that contains the RCM identifier of an RCM written at the end of the partition.
- Application Client Specific Information (ACSI): this field contains implementation identifier, VCI version and Volume ID.

Volume Coherency Information (080Ch)

Name	**T10 SSC-4 Name**	**Size**
VCR Length	VOLUME CHANGE REFERENCE VALUE LENGTH	1 byte
VCR	VOLUME CHANGE REFERENCE VALUE	8 bytes
Number of Partial Reference	VOLUME COHERENCY COUNT	8 bytes
Block number	VOLUME COHERENCY SET IDENTIFIER	8 bytes
Application Client Specific Information Length	APPLICATION CLIENT SPECIFIC INFORMATION LENGTH	2 bytes
Application Client Specific Information	APPLICATION CLIENT SPECIFIC INFORMATION	25 bytes

ACSI format for OTFormat

Offset	Length	Format	Value	Notes
0	8	ASCII	‘OTFormat’	Implementation identifier
8	1	BINARY	0*x*01	VCI version
9	16	BINARY	<Volume ID>	UUID

13.3 Host-type Attributes (08xxh)

Refer to SCSI standards SPC-4 or later for details of Host-type Attributes.

・ Application Vendor: this attribute shall be set to the name of a vendor who created the OTFormat software used when the tape was formatted in OTFormat format. T10 vendor ID string shall be used for the attribute. The attribute shall be left-aligned and padded with space (20h).

・ Application Name: this attribute shall be set to the ASCII string “OTFormat”, followed by at least one space (20h). This may be followed by a vendor-specific ASCII string for discriminating the application. An ASCII string which does not reach 32 bytes totally shall be padded by space (20h).

・ Application Version: this attribute shall be set to the version of an application used to format the tape. If the creator in the OTFormat label has a version ID, the attribute shall be consistent with it. The attribute shall be left-aligned and padded with space (20h).

・ Barcode: this attribute should be set to the same character string as the physical cartridge label. In typical methods for utilization, the first six bytes of the “Barcode” match the volume identifier in the VOL1 label. The attribute shall be left-aligned and padded with space (20h).

・ Medium Globally Unique Identifier: this attribute may consist of a System ID in the first 16 bytes and a Volume ID allocated during formatting in the following 16 bytes. If they are set, the last four bytes shall be padded with 0 (00h). The value of the System ID in this attribute shall be consistent with the value in RCMs. The value of the Volume ID in this attribute shall be consistent with the VolumeUuid in the OTFormat label.

・ Media Pool Globally Unique Identifier: this attribute may consist of a Pool ID in the first 16 bytes and a Pool Group ID in the following 16 bytes. If they are set, the last four bytes shall be padded with 0 (00h). The value of the Pool ID and the Pool Group ID in this attribute shall be consistent with the values in RCMs.

Host-type Attributes for OTFormat

Attribute Name	**Identifier**	**Size**	**Format**	**Support**
APPLICATION VENDOR	0800h	8 bytes	ASCII	Mandatory
APPLICATION NAME	0801h	32 bytes	ASCII	Mandatory
APPLICATION VERSION	0802h	8 bytes	ASCII	Mandatory
BARCODE	0806h	32 bytes	ASCII	Mandatory

MEDIUM GLOBALLY UNIQUE IDENTIFIER	0820h	36 bytes	BINARY	Optional
MEDIA POOL GLOBALLY UNIQUE IDENTIFIER	0821h	36 bytes	BINARY	Optional

Appendix A. Example of Block Offsets

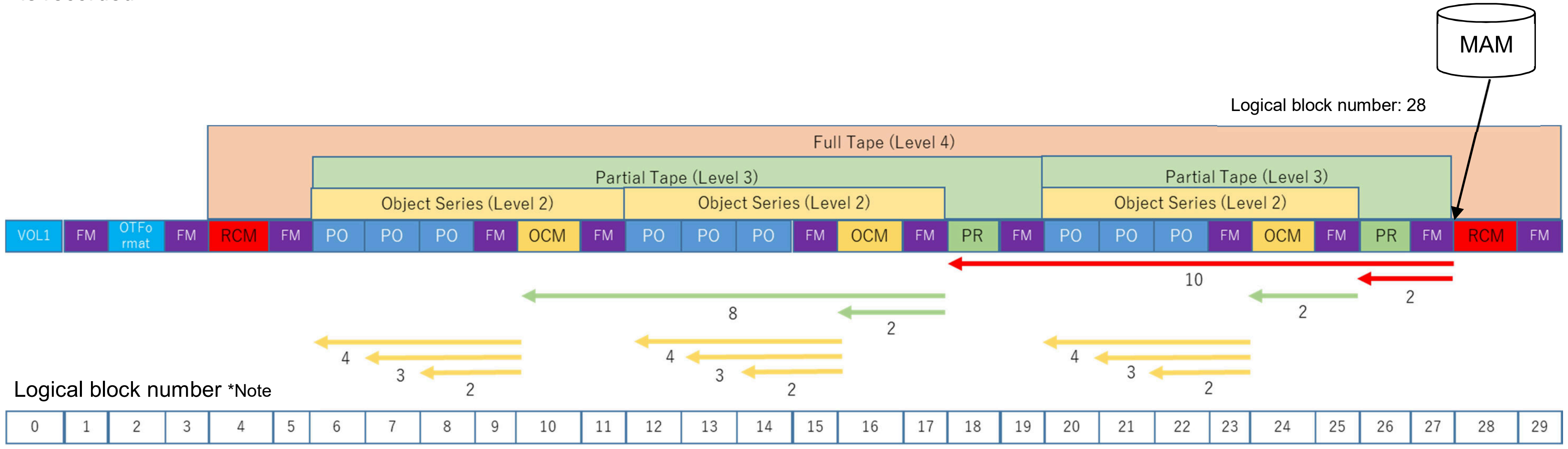
The following figure visualizes block offsets contained in RCM, PR and OCM. The numbers under the arrows are the values of block offsets.

The red arrows indicate the block offsets contained in RCMs. These offsets are from the block containing an RCM identifier to the block containing each PR identifier in the Full Tape on Data Partition.

The green arrows indicate the block offsets contained in PRs. These offsets are from the block containing a PR identifier to the block containing each OCM identifier in the Partial Tape.

The yellow arrows indicate the block offsets contained in OCMs. These offsets are from the block containing an OCM identifier to the block containing each PO identifier in the Object Series.

A logical block number of the first block of the last RCM is recorded in MAM. In OTFormat, this is the only instance a logical block numbers of the tape is recorded.



[* Note]
- Although PO, OCM and PR typically consist of plural blocks, they are shown to fit within one block in this figure.
- Although file marks are contained in OCM, PR and RCM, they are described separately in this figure.
- A file mark is counted as one block.