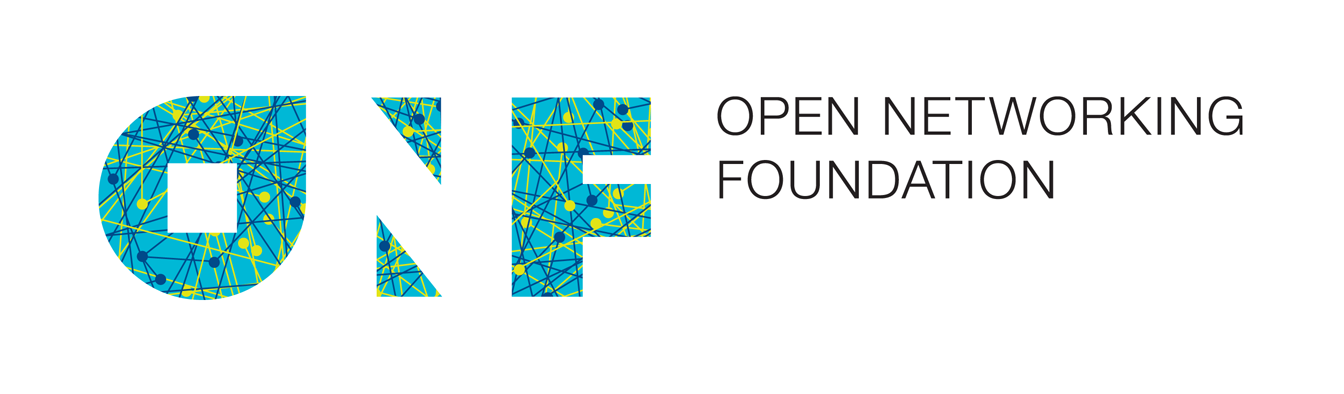
<config>  
<output path=’C:\Users\ndavis\git\OnfInfoModelOutput\ModelDescriptions\TR-512.2\_v1.3\_OnfCoreIm-ForwardingAndTermination.docx' />  
</config>

<context model=’C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreModel.uml' element=’{0}’ importedBundles='gmf;papyrus' searchMetamodels='true'/>

<gendoc><drop/>

Change path substrings above from “{path for output files}\” to your local path for the output files and “{path for CoreModel}\” to your local path for the Core Model. <drop/>

DELETE: Prior to publishing this –gd.docx (including for review), change path substrings above from “C:\Users\ndavis\git\OnfInfoModelOutput\” to “{path for output files}\” and from “C:\Users\ndavis\git\ONFInfoModel\OnfModel\” to “{path for CoreModel}\” <drop/>



Core Information Model (CoreModel)

TR-512.15

Compute

Version 1.6

September 2023

ONF Document Type: Technical Recommendation

ONF Document Name: Core Information Model version 1.6

**Disclaimer**

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Any marks and brands contained herein are the property of their respective owners.

Open Networking Foundation

2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303

[www.opennetworking.org](http://www.opennetworking.org)

©2023 Open Networking Foundation. All rights reserved.

Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation, in the United States and/or in other countries. All other brands, products, or service names are or may be trademarks or service marks of, and are used to identify, products or services of their respective owners.

Finalizing this document once generated… delete this text prior to publication:

* Replace “{{..}}” with square brackets (which trip up Gendoc)
* Select text in document from beginning of table of contents (first line) to end of document
  + Click menu item “Update Field” (on this large block of text)
    - if “Update Table…” dialogue appears select “Update entire table”
  + Repeat “update fields” 2 more times (on the same large block of text)
    - if “Update Table…” dialogue appears select “Update entire table”
* Remove reviewer comment

Note that the table of contents and figures need to be updated several times as the table length changes the page numbering and the cross references will need to be re-updated.

Table of Contents

[Disclaimer 2](#_Toc16002872)

[Open Networking Foundation 2](#_Toc16002873)

[Document History 4](#_Toc16002874)

[1 Introduction 5](#_Toc16002875)

[1.1 References 5](#_Toc16002876)

[1.2 Definitions 5](#_Toc16002877)

[1.3 Conventions 5](#_Toc16002878)

[1.4 Viewing UML diagrams 5](#_Toc16002879)

[1.5 Understanding the figures 5](#_Toc16002880)

[2 Introduction to Storage 6](#_Toc16002881)

[2.1 Introduction 6](#_Toc16002882)

[2.2 Challenges 6](#_Toc16002883)

[2.3 Storage Options 7](#_Toc16002884)

[2.4 Storage or Memory ? 7](#_Toc16002885)

[2.5 Existing Standard Storage Models 8](#_Toc16002886)

[2.6 Partitioning and Aggregation 9](#_Toc16002887)

[2.7 Storage Pooling 10](#_Toc16002888)

[3 Storage Model detail 11](#_Toc16002889)

[4 Storage Model Example - Single Disk 14](#_Toc16002890)

[5 Further Storage Model Examples 15](#_Toc16002891)

[6 Fragment: Insert class <drop/> 16](#_Toc16002892)

[7 Fragment: Insert standard diagram <drop/> 17](#_Toc16002893)

[8 Fragment: Insert small diagram <drop/> 18](#_Toc16002894)

[9 Fragment: Insert attribute row brief not Obsolete<drop/> 18](#_Toc16002895)

[10 Fragment: Insert attribute row brief <drop/> 19](#_Toc16002896)

[11 Fragment: Start attribute table brief <drop/> 19](#_Toc16002897)

[12 Fragment: Insert Attribute table brief <drop/> 19](#_Toc16002898)

[13 Fragment: Insert Ten Specified Attribute table brief <drop/> 20](#_Toc16002899)

[14 Fragment: Insert DataType <drop/> 21](#_Toc16002900)

[15 Fragment: Start Data Type attribute table brief <drop/> 22](#_Toc16002901)

[16 Fragment: Insert Data Type Attribute table brief <drop/> 22](#_Toc16002902)

[17 Fragment: Insert enums <drop/> 22](#_Toc16002903)

[17.1.1.1 [dt.name/] 22](#_Toc16002904)

List of Figures

[Figure 1 – Storage Options 7](#_Toc16087097)

[Figure 2 - StorageExtent as a ‘piece of tape’ 10](#_Toc16087098)

[Figure 3 – StorageExtent concatenation and striping 10](#_Toc16087099)

[Figure 4 - Storage Model Class Diagram 14](#_Toc16087100)

[Figure 5 - Example - Single Disk 15](#_Toc16087101)

[Figure 7-1 [diagramTitle/] 18](#_Toc16087102)

[Figure 8-1 [diagramTitle/] 19](#_Toc16087103)

Document History

| **Version** | **Date** | **Description of Change** |
| --- | --- | --- |
| 1.0 | September 2023 | Initial Version |
|  |  |  |
|  |  |  |
|  |  |  |

# Introduction

This document is an addendum to the TR-512\_v1.5 ONF Core Information Model and forms part of the description of the ONF-CIM. For general overview material and references to the other parts refer to [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_v1.3_OnfCoreIm-Overview.pdf).

## References

For a full list of references see [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_v1.3_OnfCoreIm-Overview.pdf).

## Definitions

For a full list of definition see [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_v1.3_OnfCoreIm-Overview.pdf).

## Conventions

See [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_v1.3_OnfCoreIm-Overview.pdf) for an explanation of:

* UML conventions
* Lifecycle Stereotypes
* Diagram symbol set

## Viewing UML diagrams

Some of the UML diagrams are very dense. To view them either zoom (sometimes to 400%) or open the associated image file (and zoom appropriately) or open the corresponding UML diagram via Papyrus (for each figure with a UML diagram the UML model diagram name is provided under the figure or within the figure).

## Understanding the figures

Figures showing fragments of the model using standard UML symbols and also figures illustrating application of the model are provided throughout this document. Many of the application-oriented figures also provide UML class diagrams for the corresponding model fragments (see [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_v1.3_OnfCoreIm-Overview.pdf) for diagram symbol sets). All UML diagrams depict a subset of the relationships between the classes, such as inheritance (i.e. specialization), association relationships (such as aggregation and composition), and conditional features or capabilities. Some UML diagrams also show further details of the individual classes, such as their attributes and the data types used by the attributes.

# Introduction to CPU and Memory

A data dictionary that sets out the details of all classes, data types and attributes is also provided ([TR-512.DD](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.DD_v1.3_OnfCoreIm-DataDictionary.pdf)).

Note that this model excludes CPU and Memory physical inventory, which is covered by the existing Equipment model.

## Overview

CPU and Memory are two parts of the compute model (Storage is the third and is covered in a separate document).

With current compute architectures, CPU and storage are closely tied together, often via a CPU bus and memory bus).

This model is designed to represent compute architectures in a technology independent manner for management of those compute resources.

## Existing open standards

There doesn’t appear to be a single comprehensive model that could be adopted.

The DMTF CIM and SNIA Swordfish / DMTF Redfish has some related classes and OpenConfig has CPU and memory in system / openconfig-system.yang .

## CPU

The original CPUs consisted of many physical units ( <http://www.megaprocessor.com/index.html> ).

They then were implemented in a single chip ( <https://spectrum.ieee.org/tech-history/silicon-revolution/chip-hall-of-fame-intel-4004-microprocessor> ).

Now a single chip (socket) may contain many CPU cores, and a core may run more than one thread. So we need to be clear what our definition of a ‘logical CPU’ is.

If a CPU is symmetric (all cores the same specification) then use one physical entry per CPU.

If a CPU is asymmetric, then use one physical entry per group of similar CPU cores

For example an asymmetric CPU may have 4 + 4 cores ( 4 \* 1.8 GHz Type-A + 1.4 GHz Type-B) and hence 2 entries , each of 4 cores would be used

Note that CPU chip(s) may be FRU or non FRU inventory – here we are interested in the processor function.

## Memory

Memory chip(s) / SIMM DIMM modules may be FRU or non FRU inventory – here we are interested in the memory capabilities and capacity.

## Pooling

Note that the decision was made to have a single compute pool rather than separate Storage, CPU and memory pools[[1]](#footnote-1), because :

* CPU and memory are usually tightly coupled (vie the CPU and memory buses) and the pool can then allocate these consistently
* *Sometimes* storage is tightly coupled with CPU and memory and the pool can then allocate these consistently (local attached un-sharable storage)

Note also that the pools aren’t hierarchical (deliberately no ComputePool contained in self-join)

* The association StorageExtentPoolEntryIsLogical allows an output from one pool to become the input of another pool
* This needs to form a directed acyclic graph (no loops)

Note that there is no association linking the pool inputs and outputs. The ordering of the inputs allows the input to output extent mapping to be determined.

Assume that there will be a large number of simple pools rather than few large complex pools with complex mappings.

## Partitioning and Aggregation

The partitioning and aggregation of CPU and memory is subtly different from that of Storage.

With Storage, in theory each extent can be considered separately. Some extents could come from a disk A, some from disk B. With a sensible pool allocation there is no need to worry about segmenting the pool.

With CPU and storage we have 2 issues :

1. The pool needs to remember and enforce segmentation (resource chunks)
2. The pool needs to pair CPU and Memory segments

For example if there are 2 blade servers Blade-A and Blade-B.

* I can’t allocate a software process CPU from Blade-A and memory Blade-B.
* I can’t give a software thread half its CPU requirements from Blade-A and half from Blade-B.
* I can’t give a software thread half its memory requirements from Blade-A and half from Blade-B.

A multi-threaded application may be able to run across multiple CPU/memory pairs but it would have to know the segmentation.

This model will use the following :

* Class ComputePoolSegment is defined to allow the definition of ‘segments’
* Inputs to the pool can be optionally assigned to a segment
* Each pool input and output also has an optional segmentId UUID
* If an input entry has a segmentId, than no output can be assigned that ‘crosses’ the segment (Each output can only relate to 1 segmentId)
* Multiple inputs can only be combined if both have no segmentId or if both have the same segmentId
* If an input entry has a segmentId, then it should be propagated to any outputs (fill out the output segmentId with the input segmentId)
* If a CPU output has a segmentId then it can only be used with memory with the same segmentId (and visa-versa)
* Each segment should have its own internal number range in the pool

## Items for Further Investigation

What units should we use for memory sizes – bytes, kiB, MiB … ?

What units should we use for CPU - Hz, kHz, MHz … ?

* Note that Kubernetes works in units of CPU, where   
  “One CPU, in Kubernetes, is equivalent to a *Hyperthread* on a bare-metal Intel processor with Hyperthreading”
* A CPU hardware thread is also called a vCPU (virtual CPU)

Note that we should try and avoid using floating point numbers for the quantities as the rounding proves problematic

# CPU and Memory model detail

ComputePool – a pool that manages ComputePoolEntries. These entries can consist of Storage, Memory and CPU resources. This document only considers CPU and memory entries.

ComputePoolEntry – An abstract class used as the parent of Storage, Memory and CPU pool entries. Note that this class is subclassed by input and output variants – clarifying the role each entry plays and also allowing the inputs and outputs to be separately ordered.

ComputePoolInputEntry – An abstract class used as the parent of Storage, Memory and CPU pool input entries.

ComputePoolOutputEntry – An abstract class used as the parent of Storage, Memory and CPU pool output entries.

ComputePoolSegment – allows for instance CPU and Memory that need to be kept together to be grouped

A screenshot of a diagram

Description automatically generated

Figure 1 – Compute Pool

## CPU

CpuPoolPhysicalEntry – a ComputePoolInputEntry for a CPU. It contains attributes specific to the CPU implementation and can be related to the Equipment providing the CPU.

CpuPoolLogicalEntry – a ComputePoolInputEntry for CPU capacity and capability from another ComputePool

CpuCoreAllocation – the output from a ComputePool. This output can be allocated to another ComputePool where it could be further manipulated.

A screenshot of a computer diagram

Description automatically generated

Figure 2 – CPU model

## Memory

MemoryPoolPhysicalEntry – a ComputePoolInputEntry for memory. It contains attributes specific to the memory implementation and can be related to the Equipment providing the memory.

MemoryPoolLogicalEntry – a ComputePoolInputEntry for memory capacity and capability from another ComputePool

MemoryBlockAllocation - – the output from a ComputePool. This output can be allocated to another ComputePool where it could be further manipulated.

A diagram of a computer

Description automatically generated

Figure 3 – Memory model

## Relationship to Running Software

It may also be of interest to show the consumption of CPU and Memory capacity by running software (which is what CPU and memory is provided for).

The Software model is covered in TR-512.12.

A screenshot of a computer program

Description automatically generated

Figure 4 - Running software use of CPU and Memory

## Examples

### A simple compute example

A screenshot of a computer program

Description automatically generated

Figure 5 - Simple Compute example

This simple example shows how the concepts in the model fit together.

There is traceability from the physical equipment instances to the CPU and Memory functionality and from there to the running software that is using the functionality.

A screenshot of a computer

Description automatically generated

Figure 6 - Compute example instance diagram

# Introduction to Storage

A data dictionary that sets out the details of all classes, data types and attributes is also provided ([TR-512.DD](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.DD_v1.3_OnfCoreIm-DataDictionary.pdf)).

## Introduction

This document proposes a generic model for managing storage.

More specifically, it will cover management of Block, File and Object storage, both directly attached and over a network. This includes standalone hosts with local storage, raid, SCSI as well as network based storage, including enterprise and cloud storage.

Note that this document specifically excludes storage physical inventory, which is already covered by the existing ONF CIM Equipment model.

## Challenges

The challenges in producing an abstract, standard storage model include :

* The large number of variations in storage options
* the lack of even a standard terminology.

For example, the definition of a LUN is problematic :

* + Is it a Logical Unit Number (just the number) ?
  + A Logical Unit ?
* “A LUN, is a number used to identify a logical unit, which is a device addressed by the SCSI protocol or Storage Area Network protocols which encapsulate SCSI, such as Fibre Channel or iSCSI[”  
  https://en.wikipedia.org/wiki/Logical\_unit\_number](https://en.wikipedia.org/wiki/Logical_unit_number)
* “a logical unit *number* (LUN) is a slice or portion of a configured set of disks that is presentable to a host and mounted as a volume within the OS.”  
  <https://www.computerweekly.com/answer/What-is-a-LUN-and-why-do-we-need-storage-LUNs>

## Storage Options

Storage can be provided in many forms and some of the options commonly used today are shown below.

With each of the options, there could be more than one protocol used, and the diagram shows some of these in pink..

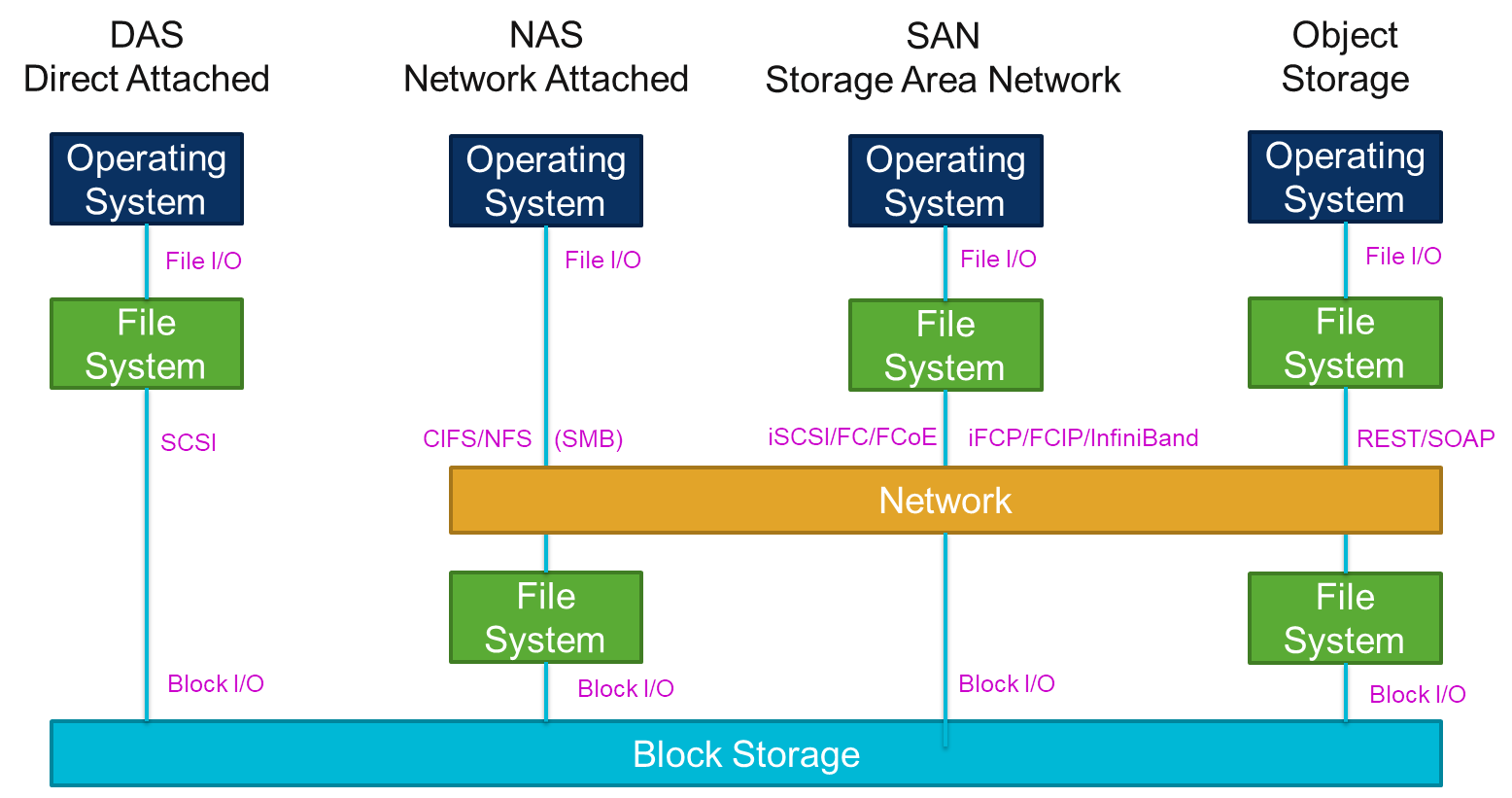


Figure 1 – Storage Options

## Storage or Memory ?

Note that there is some similarity between storage and memory. Both allow data to be ‘at rest’, ready for later retrieval. Also there are “RAM drives” or “RAM disks” where memory can be used as a volatile drive and “virtual memory” where volatile memory is paged in and out of disk to increase the apparent amount of main memory.

The storage model is appropriate when the access is via storage (file, block, object) protocols and for memory when access is via the memory protocols (memory location).

## Existing Standard Storage Models

Extensive research has only identified two open standard storage models, the DMTF CIM model and the SNIA Swordfish / DMTF Redfish model.

<https://www.dmtf.org/standards/cim>

<https://redfish.dmtf.org/redfish/schema_index>

The key DMTF CIM concepts are :

* Core / **StorageExtent** – “describes the capabilities and management of the various media that exist to store data and allow data retrieval.”
* Device / **MediaPartition** – “a presentation of a contiguous range of logical blocks and has identifying data written on/to it.”
* Device / **LogicalDisk** – “a presentation of a contiguous range of logical blocks that is identifiable by applications such as filesystems via the Name field.”
* Device / **StorageVolume** – “a StorageExtent that is published for use outside of the scoping System.”
* Device / **StoragePool** – “a conglomeration of storage capacity for the purpose of assignment and allocation based on service characteristics”
* Device / **CompositeExtentBasedOn** – “defines how data is striped across StorageExtents.”
* Device / **CompositeExtent** – “is used to model the distribution of user data across one or more underlying StorageExtents, which may or not be protected by some redundancy mechanism. CompositeExtents represent a contiguous range of logical blocks.”
* Physical / **PhysicalMedia** – “represents any type of documentation or storage medium, such as tapes, CDROMs, etc.”

The key Swordfish / Redfish concepts are :

* **Drive** – “a single physical disk drive for a system, including links to associated Volumes.”
* **Simple Storage** – “a storage controller and its directly-attached devices”
* **Storage** – “a set of storage controllers (physical or virtual) and the resources such as volumes that can be accessed from that subsystem.”
* **Volume** – “properties used to describe a volume, virtual disk, LUN, or other logical storage entity for any system.”

The definitions above show that it is important to clearly define the storage concepts and how they relate.

## Storage Extent

We will define StorageExtent as the key unit of storage capacity that the rest of the model is built around.

The extent is a block or segment of storage (bytes).

Normally we are not interested in managing individual extents, so our model uses ranges of extents.

## Partitioning and Aggregation

The model needs to support both partitioning and aggregation of StorageExtent ranges. To help in the understanding of what is required, a simple example is worked through below, representing a StorageExtent range as a ‘piece of tape’.



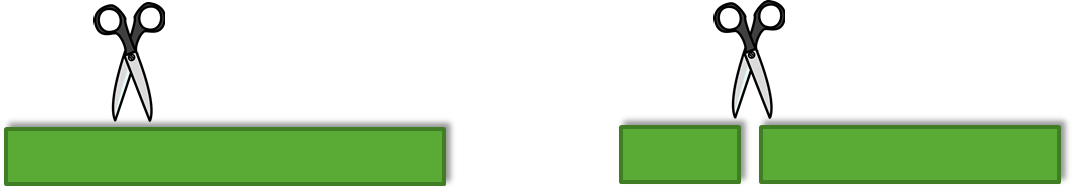
Figure 2 - StorageExtent as a ‘piece of tape’

We will allow the tape to be aggregated in two ways, by end to end concatenation and by striping.



Figure 3 – StorageExtent concatenation and striping

We will only allow the tape to be cut into sub extents (the opposite of concatenation)



Note that the operations are ‘closed’ that is both the inputs and the results of the operations are StorageExtent ranges, allowing the operations to be performed recursively.

## Storage Pooling

Originally storage was based on local storage due to the limitations of the hardware at the time. To support shared storage (provided over a network), the model needs to support pooling of physical storage that can then be allocated logically to various consumers.

To do that, the model defines a ComputePool with ComputePoolEntries.

There are two types of entries :

* Pool inputs for SSD, PhysicalDisk, VM VirtualDisk and LogicalEntry
* A pool output as an extent allocation (volume)

Note that the extent allocation from a pool can be a LogicalEntry to another pool allowing for allocation chaining.

Note that the decision was made to have a single compute pool rather than separate Storage, CPU and memory pools[[2]](#footnote-2), because :

* CPU and memory are usually tightly coupled and the pool can then allocate these consistently
* *Sometimes* storage is tightly coupled with CPU and memory and the pool can then allocate these consistently

Note also that the pools aren’t hierarchical (deliberately no ComputePool contained in self-join)

* The association StorageExtentPoolEntryIsLogical allows an output from one pool to become the input of another pool
* This needs to form a directed acyclic graph (no loops)

Note that there is no association linking the pool inputs and outputs. The ordering of the inputs allows the input to output extent mapping to be determined.

Assume that there will be a large number of simple pools rather than few large complex pools with complex mappings.

# Storage Model detail

StorageExtentRange – a datatype that standardizes the representation of a range of extents (storage capacity)

ComputePool – a pool that manages ComputePoolEntries. These entries can consist of Storage, Memory and CPU resources. This document only considers storage entries.

ComputePoolEntry – An abstract class used as the parent of Storage, Memory and CPU pool entries. Note that this class is subclassed by input and output variants – clarifying the role each entry plays and also allowing the inputs and outputs to be separately ordered.

ComputePoolInputEntry – An abstract class used as the parent of Storage, Memory and CPU pool input entries.

ComputePoolOutputEntry – An abstract class used as the parent of Storage, Memory and CPU pool output entries.

StorageExtentPoolPhysicalSsdEntry – a ComputePoolInputEntry for SSD storage. It contains attributes specific to the SSD implementation and can be related to the Equipment providing the storage.

StorageExtentPoolPhysicalDiskEntry – a ComputePoolInputEntry for a Physical Disk. It contains attributes specific to the physical disk implementation and can be related to the Equipment providing the storage.

StorageExtentPoolVirtualEntry – a ComputePoolInputEntry for a VM virtual disk implemented as a File in a FileSystem. The model allows traceability back from the File to the FileSystem, to the StorageExtentAllocation and back to the underlying hardware.

StorageExtentPoollLogicalEntry – a ComputePoolInputEntry for logical storage from another ComputePool

StorageExtentAllocation – the output from a ComputePool. This output can have a file system installed for direct use or it could be allocated to another ComputePool where it could be further manipulated.

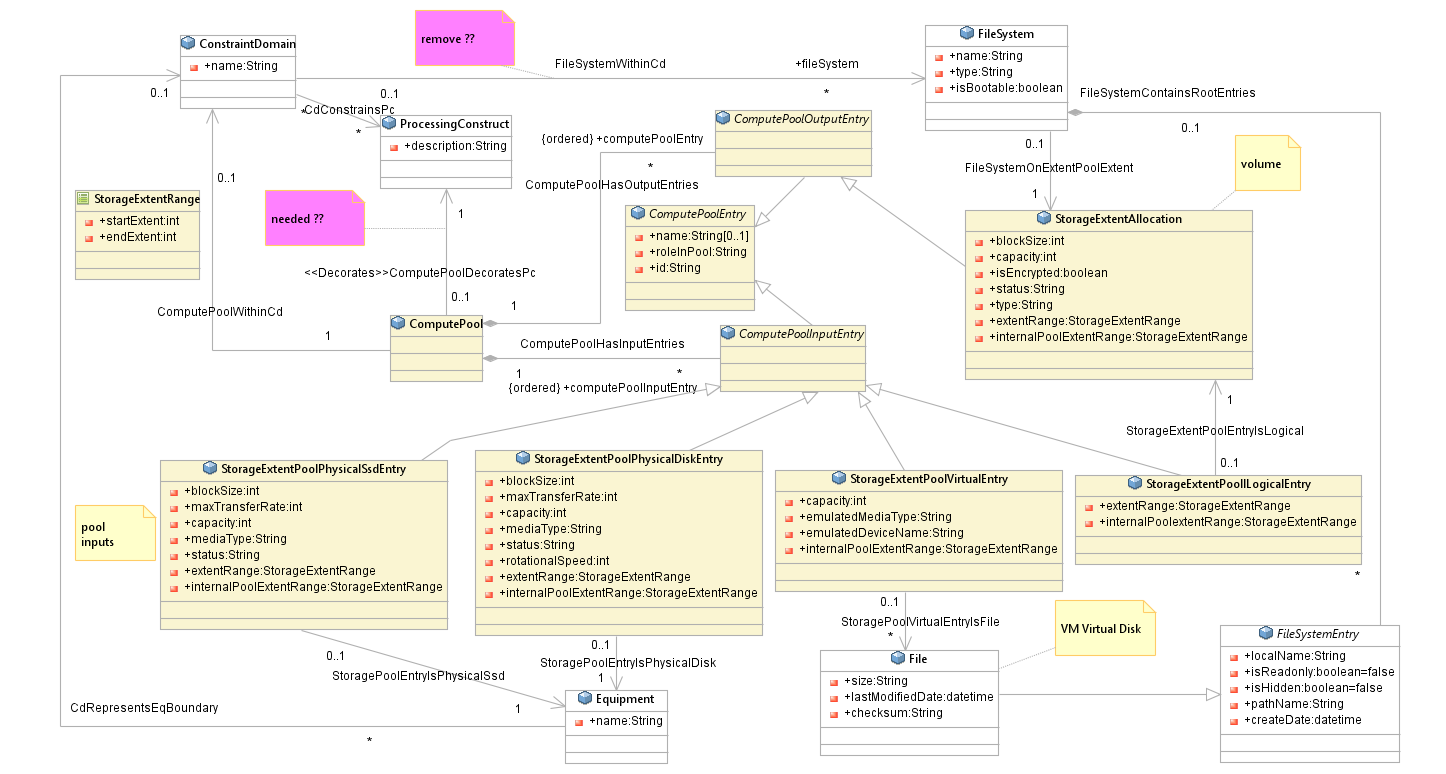


Figure 4 - Storage Model Class Diagram

Note that with association StorageExtentPoolEntryIsLogical, the ends of an instance will be on different ComputePools (it’s showing that the input to a pool was an output from another pool).

# Storage Model Example - Single Disk

A single attached disk is the simplest example (one in – one out).

In this case there is one instance of Equipment that maps to one pool input into one pool and one pool output which then has a FileSystem installed on it.

Note the overhead that our general model imposes on this case.

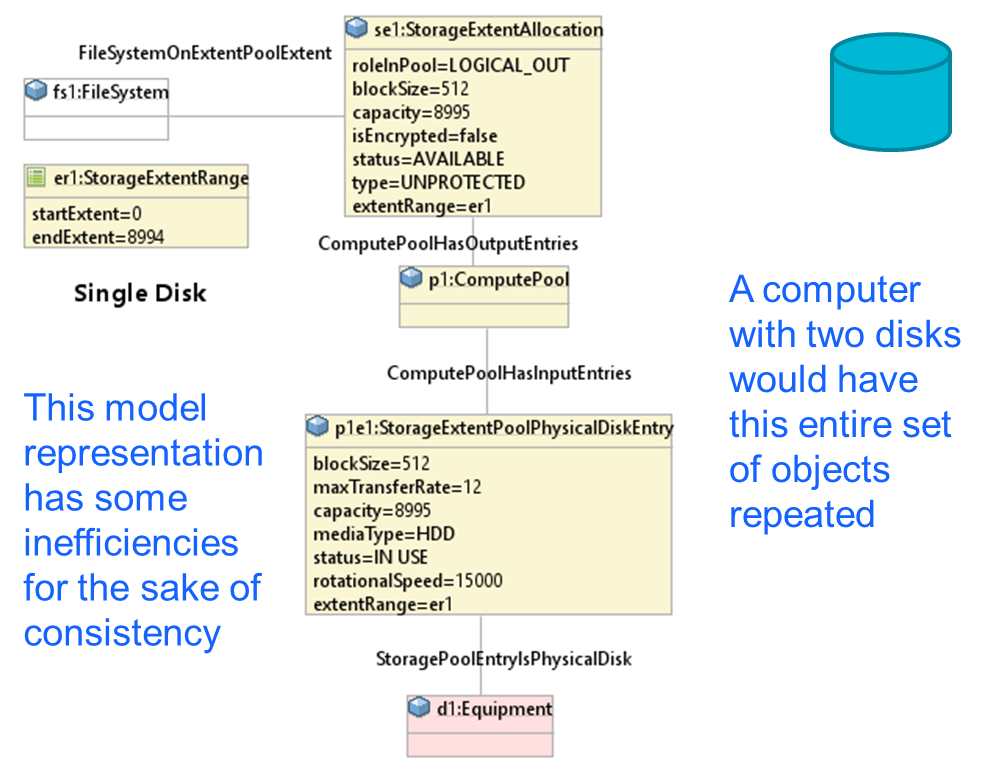


Figure 5 - Example - Single Disk

# Further Storage Model Examples

Further examples are in the appendix A.14

**End of Document**

</gendoc><drop/>

To take latest template: <drop/>

* delete text from “Template version…” to end of file <drop/>
* insert a line in “Normal” style<drop/>
* insert text (Insert 🡪 Object 🡪 Text from File… (alt njf)) from: <drop/>
  + TR-512.GT\_v1.3\_OnfCoreIm-CommonGendocTemplate-Fragments.docx <drop/>

Template version 0.0.10 17 September 2017 <drop/>

# Fragment: Insert class <drop/>

<fragment name=’insertClass’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>  
<arg name=’className’ type=’String’/><drop/>  
<arg name=’packageName’ type=’String’/><drop/>  
[if (not cl.qualifiedName.contains(packageName))]<drop/>  
[else] <drop/>  
[if(cl.name.contains(className))]<drop/>

Qualified Name: [cl.qualifiedName/]

[for (co:Comment | cl.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>  
[if (cl.isAbstract)]<drop/>

This class is abstract.

[/if]<drop/>

[if (cl.oclAsType(uml::Class).general ->notEmpty())]<drop/>

Inherits properties from:

[for (gen:Class | cl.oclAsType(uml::Class).general)]<drop/>

* [gen.name/]

[/for]<drop/>

[/if]<drop/>

[for (st:Stereotype | cl.getAppliedStereotypes())]<drop/>  
[if(not st.name.contains(‘OpenModelClass’))]<drop/>

This class is [st.name/].

[else] <drop/>  
[/if]<drop/>  
[/for]<drop/>  
[else] <drop/>  
[/if]  
[/if]  
</fragment><drop/>

# Fragment: Insert standard diagram <drop/>

<fragment name=’insertStandardDiagram’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Package’/><drop/>  
<arg name=’diagramName’ type=’String’/><drop/>  
<arg name=’diagramTitle’ type=’String’/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>

[if d.name.contains(diagramName)]

<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = ‘false’></image>

CoreModel diagram: [d.name/]

Figure 7-1 [diagramTitle/]

[else]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert small diagram <drop/>

<fragment name=’insertSmallDiagram’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Package’/><drop/>  
<arg name=’diagramName’ type=’String’/><drop/>  
<arg name=’diagramTitle’ type=’String’/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>

[if d.name.contains(diagramName)]

<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = ‘false’></image>

CoreModel diagram: [d.name/]

Figure 8-1 [diagramTitle/]

[else]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert attribute row brief not Obsolete<drop/>

<fragment name=’insertAttributeRowBriefNotObsolete’ importedBundles=’commons;gmf;papyrus’><drop/>

Does not work unless we have Mature stereotype… <drop/>  
<arg name=’p’ type=’uml::Property’/><drop/>

[for (st:Stereotype | p.getAppliedStereotypes())]<drop/>

[if(not st.name.contains(‘OpenModelAttribute’))]

[if(not st.name.contains(‘Obsolete’))]

| [p.name/] | [for (st:Stereotype | p.getAppliedStereotypes())]<drop/>  [if(not st.name.contains(‘OpenModelAttribute’))] [st.name/]  [/if]<drop/>  [/for]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> | [if p.ownedComment->notEmpty()]<drop/>  [for (c:Comment | p.ownedComment)] <drop/>  [cleanAndFormat(c.\_body.clean())/]  [/for]  [else] [if (p.name.contains (‘\_’))]See referenced class  [else]To be provided  [/if]<drop/>  [/if]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> |
| --- | --- | --- |

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert attribute row brief <drop/>

<fragment name=’insertAttributeRowBrief’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Property’/><drop/>

| [p.name/] | [for (st:Stereotype | p.getAppliedStereotypes())]<drop/>  [if(not st.name.contains(‘OpenModelAttribute’))] [st.name/]  [/if]<drop/>  [/for]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> | [if p.ownedComment->notEmpty()]<drop/>  [for (c:Comment | p.ownedComment)] <drop/>  [cleanAndFormat(c.\_body.clean())/]  [/for]  [else] [if (p.name.contains (‘\_’))]See referenced class  [else]To be provided  [/if]<drop/>  [/if]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> |
| --- | --- | --- |

</fragment><drop/>

# Fragment: Start attribute table brief <drop/>

<fragment name=’insertAttributeTableHeader’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Lifecycle Stereotype (empty = Mature)** | **Description** |

</fragment><drop/>

# Fragment: Insert Attribute table brief <drop/>

<fragment name=’insertAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>  
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [cl.name/]

<table><drop/>

[cl.insertAttributeTableHeader ()/]

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (not p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/for]<drop/>

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert Ten Specified Attribute table brief <drop/>

<fragment name=’insertTenSpecifiedAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>

<arg name=’p1’ type=‘String’/><drop/>

<arg name=’p2’ type=‘String’/><drop/>  
<arg name=’p3’ type=‘String’/><drop/>  
<arg name=’p4’ type=‘String’/><drop/>  
<arg name=’p5’ type=‘String’/><drop/>  
<arg name=’p6’ type=‘String’/><drop/>  
<arg name=’p7’ type=‘String’/><drop/>  
<arg name=’p8’ type=‘String’/><drop/>  
<arg name=’p9’ type=‘String’/><drop/>  
<arg name=’p10’ type=‘String’/><drop/>  
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [cl.name/]

<table><drop/>

[cl.insertAttributeTableHeader ()/]

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4) or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or p.name.contains(p9) or p.name.contains(p10))]<drop/>

[if (not p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/if]<drop/>

[if (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4) or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or p.name.contains(p9) or p.name.contains(p10))]<drop/>

[if (p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert DataType <drop/>

<fragment name=’insertDataType’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>  
<arg name=’dataTypeName’ type=’String’/><drop/>  
<arg name=’packageName’ type=’String’/><drop/>  
[if (dt.qualifiedName.contains(packageName))]<drop/>  
[if(dt.name.contains(dataTypeName))]<drop/>

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>  
[if (dt.oclAsType(uml::DataType).general ->notEmpty())]<drop/>

Inherits properties from:

[for (tp:DataType | dt.oclAsType(uml::DataType).general)]<drop/>

* [tp.name/]

[/for]<drop/>

[for (gen:Class | dt.oclAsType(uml::DataType).general)]<drop/>

* [gen.name/]

[/for]<drop/>

[/if]<drop/>

[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>  
This class is [st.name/].

[/for]<drop/>  
[else] <drop/>  
[/if]  
[/if]  
</fragment><drop/>

# Fragment: Start Data Type attribute table brief <drop/>

<fragment name=’insertDataTypeAttributeTableHeader’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Lifecycle Stereotype (empty = Mature)** | **Description** |

</fragment><drop/>

# Fragment: Insert Data Type Attribute table brief <drop/>

<fragment name=’insertDataTypeAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertDataTypeAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>  
[if dt.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [dt.name/]

<table><drop/>

[dt.insertDataTypeAttributeTableHeader ()/]

[for (p:Property|dt.ownedAttribute)]<drop/>

[p.insertAttributeRowBrief ()/]

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert enums <drop/>

<fragment name=’insertEnums’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>

#### [dt.name/]

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>

Applied stereotypes:

[if dt.getAppliedStereotypes()->notEmpty()] <drop/>

[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>

* [st.name/]

[/for]<drop/>

[else] No stereotypes applied

[/if]<drop/>

[if (dt.oclAsType(uml::DataType).general ->notEmpty())]<drop/>

Inherits literals from:

[for (tp:DataType | dt.oclAsType(uml::DataType).general)]<drop/>

* [tp.name/]

[/for]

[/if]<drop/>

[if (dt.oclAsType(Enumeration).ownedLiteral->notEmpty())]<drop/>

Contains Enumeration Literals:

[for (e:EnumerationLiteral|dt.oclAsType(Enumeration).ownedLiteral)]<drop/>

* [e.name/]:
  + [for (co:Comment | e.ownedComment)]<drop/>
  + <dropEmpty>[cleanAndFormat(co.\_body.clean())/]
  + </dropEmpty>[/for]<drop/>
  + [if dt.getAppliedStereotypes()->notEmpty()] <drop/>
  + Applied stereotypes:
    - [for (st:Stereotype | e.getAppliedStereotypes())]<drop/>
    - [st.name/]
    - [/for]<drop/>
  + [/if]<drop/>

[/for]<drop/>

[/if]<drop/>

</fragment><drop/>

1. Note that this document only considers CPU and memory entries. [↑](#footnote-ref-1)
2. Note that this document only considers storage entries. [↑](#footnote-ref-2)