



# Hammerspace Objectives

## Overview

Hammerspace Service Level Objectives are a new paradigm that empowers organizations and stakeholders to declare the intent of their data. It couples desired outcomes with programmable conditions. That is the essence of declarative control of data. The significance is that data can, then, be organized through metadata characteristics, inherent or custom, and further tailored to empower desired business outcomes. Declarative statements, such as Hammerspace Objectives, are a far more effective and modern method of creating business outcomes from data than the more archaic imperative policy construct.

In the imperative policy world, such as used by most “policy engines”, you have to be exceedingly detailed about exactly how to accomplish a desired end-state. This often translates to very lengthy and complex instructions that outline exactly how something should be accomplished. This can translate to a hundred lines or more of instructions. The problem only gets worse when trying to scale, upgrade, extend, or troubleshoot imperative policies. What if your workflow and production requirements need ten, twenty, or more imperative policies. It’s easy to imagine that this quickly becomes untenable.

Declarative policies, like those used with Hammerspace Objectives, remove the complexity and inefficiency of the imperative and skip directly to the desired end-state. Telling a system WHAT you want to happen is infinitely simpler and far more elegant than the error-prone, tedious, and time-consuming procedural instructions necessary within the old imperative paradigm. Declarative policies accelerate workflows, operations, and business outcomes.



The goal of this document is to explain key Hammerspace Objective concepts and provide several detailed examples of how to configure Hammerspace Objectives to match common deployment scenarios.

## Table of Contents

### Version 1.2

<b>Introduction</b>	<b>6</b>
Managing Infrastructure at Scale	6
Goal of this Document	6
What Are Hammerspace Objectives and Conditions?	7
Key Terms	9
Share Default Objectives	11
Client Data Path Architecture	12
Applying Objectives	13
Aligned, Partially Aligned, and Unaligned	14
Hammerspace Condition Wizard	16
Condition Wizard - Writing Hammerscript	17
<b>Objective Definitions</b>	<b>18</b>
Placement Objectives	18
Objective: Keep-online	19
Objective: Place-on and Confine-to	19
Objective: Exclude-from	21
Tiering Data Using Objectives	21
Data Availability and Protection Objectives	22
Objective: Availability-#-nines	22
Volume Availability - Default Values	23
Objective: Durability-#-nines	23
Volume Durability - Default Values	24
Objectives: Versioning, Undelete, and File Conflict Resolution (Log-Xfer)	24
Apply using the Share Properties   Advanced menu	25
Advanced Objectives	26
Objective: Access-based-enumeration	26
Objective: Acl-compatibility-mode-ignore-chmod	27
Objective: Acl-compatibility-mode-non-posix	27
Objective: Do-not-cache	27
Objective: Layout-get-on-open	28



Objective: No-atime	28
Objective: Optimize-for-capacity	28
<b>Using Metadata as Objective Conditions</b>	<b>29</b>
Understanding the Document Code Blocks	29
What is Metadata?	30
Hammerspace Metadata	30
Commonly Used Metadata Values	31
Filename or Extension	31
Folder / Path	32
Timestamps	33
LAST_USE_AGE	33
Timestamp Best Practices	34
Custom Metadata	34
Labels	35
Creating Labels	36
Creating a Label with Implied Labels	36
Applying Labels	37
Conditions Examples - Labels	37
Tag	38
Applying Tags	38
Tag with Value Specified	38
Tag with Default Value of True	38
Condition Examples - Tags	39
Condition Example - Tag and Value	39
Condition Example - Tag	39
Delete Tags	39
Keyword	40
Applying Keywords	40
Conditions Examples - Keywords	40
Advanced Metadata Attribute Values	40
DATA_ORIGIN_LOCAL	40
DATA_ORIGIN_REMOTE	41
HAS_KEEP_ON_THIS_SITE	41
HAS_ONLINE_INSTANCE	42
IS_BEING_CREATED	42
IS_DURABLE	42
IS_LIVE	42
IS_SNAP	42



IS_RECENTLY_USED	42
<b>Combining or Modifying Conditions</b>	<b>44</b>
Marking a Condition as a Negative	44
Build a Conditional Statement using Conditions	44
Example 1	45
Example 2	45
Example 3	45
<b>Objective Scenarios and Solutions</b>	<b>46</b>
Scenario #1: One Site - Three-Tier NAS	47
Requirements	47
Key Design and Architecture Takeaways - One Site - Three-Tier NAS	48
Objective Design - All Sites	48
Scenario #2: Two Sites - Global File Shares	50
All Sites - Requirements	50
Key Design and Architecture Takeaways - Two Sites - Global File Shares	51
Objective Design - All Sites	52
Scenario #3: Hub-spoke Multi-Site - Global File Shares	53
Primary Site - Los Angeles - Requirements	54
Key Design and Architecture Takeaways - Los Angeles	54
Objective Design - Los Angeles	55
Spoke site 1 - New York City - Requirements	55
Key Design and Architecture Takeaways - New York City	55
Objective Design - New York City	56
Spoke Site 2 - Miami - Requirements	56
Key Design and Architecture Takeaways - Miami	57
Objective Design - Miami	57
Scenario #4: Mesh Multi-Site - Global File Shares	59
Mesh Site 1 - Seattle - Requirements	59
Key Design and Architecture Takeaways - Seattle	60
Objective Design - Seattle	61
Mesh Site 2 - Houston - Requirements	61
Key Design and Architecture Takeaways - Houston	61
Objective Design - Houston	62
Mesh Site 3 - Rome - Requirements	62
Key Design and Architecture Takeaways - Rome	63
Objective Design - Rome	63
<b>Appendix</b>	<b>65</b>



Hammerspace Toolkit	65
Hammerspace Metadata Plugin for Windows File Explorer	65
Metadata Plugin - Download and Installation	65
File Metadata Example (Full)	67



# Introduction

## Managing Infrastructure at Scale

Building a scalable cluster utilizing different storage classes can be complex. Extending this design across multiple sites adds to the complexity. Hammerspace Objectives provides a way to specify where each cluster should place (or provide multiple copies of) data – even between sites or in cloud object storage volumes. With the power of Hammerspace Objectives, you can customize your data's behavior to match your business workflows, data lifecycle policies, and other needs.

## Goal of this Document

The purpose of this document is to provide an overview of Hammerspace Objectives and the conditions that define when they will be applied.

This document will:

- Provide an overview of the most commonly used Hammerspace Objectives, including those applied to new shares by default.
- Provide an overview of the most commonly used Hammerspace Objective conditions, which can be used to control if and when an objective will be applied.
- Provide multiple, real-life scenarios that translate plain-language requirements into a series of Hammerspace Objectives and conditions.

This document is not meant to be an authoritative source of information on all objectives or conditions, but merely a jumping-off point for configuring Hammerspace to achieve your data management requirements. Consult the Hammerspace documentation for additional information about using the Hammerspace platform.

**Note:** Share snapshots are not configured using objectives, and are therefore out of scope for this document. Consult the **Hammerspace Administration Guide** for information regarding their configuration and use.



## What Are Hammerspace Objectives and Conditions?

Hammerspace Objectives are instructions that we use to implement our data orchestration goals, and control the behavior of the filesystem.

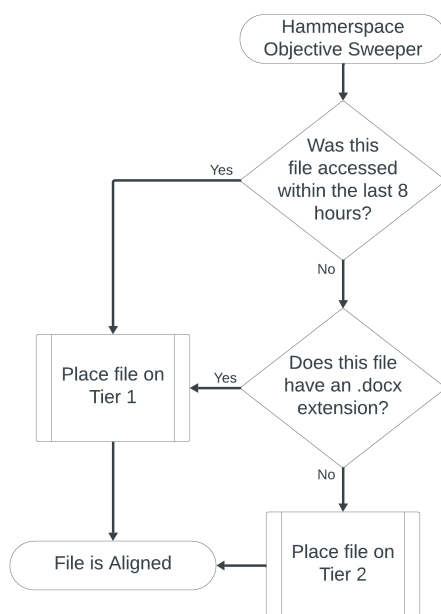
Conditions are rules that can be used to control how objectives are enforced. They can be as broad or as granular as your data orchestration needs require.

To illustrate how an objective works, let's look at one expressed in the following example.

### Tiering Example

**Goal:** Keep on Tier 1 all recently accessed (< 8hrs) files or .docx files

Objective	Condition
place-on-VG-Tier1	LAST_USE_AGE <= 8*HOURS
place-on-VG-Tier1	FNMATCH( "*.docx", NAME)





In this example, there are two conditions that are checked by the Hammerspace Objective Sweeper process, and two possible outcomes. Only one condition needs to be true in order to place the file on “Tier 1”, though it is possible to require that both are true.

Hammerspace Objectives can be broken down into three categories:

- **Storage** - The family of placement objectives controls where your data is placed (or not placed), and how many file instances will be created.
  - These are discussed in the [Placement Objectives](#) and [Data Availability and Protection Objectives](#) sections of this document.
- **Data Protection** - The versioning, undelete, and file conflict resolution objectives complement snapshots to enhance on-cluster data protection, and enable client self-recovery capabilities.
  - These are discussed in the [Versioning, Undelete, and File Conflict Resolution \(Log-Xfer\)](#) section of this document.
- **Behavior** - These objectives control the behavior of the filesystem. Some examples include access-based enumeration, external antivirus integration, filesystem timestamp or ACL behavior, and various NFS/SMB protocol-specific options.
  - Some of these objectives are discussed in the [Advanced Objectives](#) section of this document.

In the [Using Metadata as Objective Conditions](#) section of this document, you will learn how to use conditions to control precisely how objectives are applied. which is an important part of ensuring that your data orchestration goals are met.

In the [Objective Scenarios and Solutions](#) section of this document, you will see examples of how Hammerspace architects translate plain-language data orchestration requirements into Hammerspace Objectives and objective conditions.



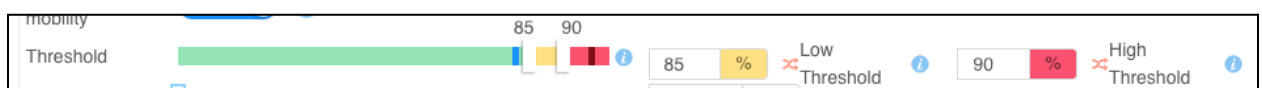


## Key Terms

- **Alignment** - Alignment is used to report if a file is currently adhering to all, some, or none of the currently applied objectives. Alignment is discussed in greater detail in the **Aligned, Partially Aligned, and Unaligned** section of this document.
  - **Aligned** - Aligned indicates that all applicable objectives have been met for a file. The cumulative measurement of all file alignment statuses can be observed using the share dashboard or the Hammerspace toolkit.
  - **Partially Aligned** - Indicates that a file adheres to at least some of the applicable objectives.
  - **Unaligned** - Indicates that a file does not currently adhere to any of the applicable objectives.
- **Condition** - Conditions are rules used to specify when an objective will be applied. If no condition is specified, the objective will be applied to all files.
- **Global file share** - A share added to more than one Hammerspace cluster. Each cluster will have the same view of the share contents even if all files are not available at that site.
- **Hammerscript** - The proprietary scripting language that underpins Hammerspace Objectives and conditions.
- **Instance** - A file placed on Hammerspace will have at least one instance written to the underlying storage - more if the objectives require it.
- **Objective** - Rules that define the intent for managing files. The focus of this document is primarily on using objectives to control the placement of data.
- **Online file** - A file that is considered online when it is in either a DSX volume or a connected NFS storage volume.
- **Offline file** - A file that is considered offline when it is only available in an object storage volume.
- **Metadata** - Data that provides information about other data. Hammerspace Objectives frequently use file metadata to determine what actions to take. Hammerspace also supports custom file metadata such as keywords, tags, labels, and attributes.



- **Objective sweeper** - The objective sweeper process runs continually to evaluate (based on the applied objectives) if files are still aligned, or if they are unaligned and an action needs to be taken.
- **On-demand mobility** - In a global file share environment, on-demand mobilities occur if a user opens a file that is not currently stored in an online cluster volume at their site. Both the site that sends the file and the site that requested the file will note the mobility reason as "on-demand".
- **Resilver** - The process of remirroring or rebuilding data.
- **Shares** - Ordinary file shares used for clients to place data on Hammerspace, Hammerspace supports SMB, NFS, and S3 protocols. Objectives ultimately determine what volumes data is ultimately placed on.
- **Site** - The terms site and cluster are often used interchangeably, both in this document and within the Hammerspace CLI/GUI; both refer to a specific Hammerspace cluster, which may or may not be at another physical location away from a separate Hammerspace cluster.
- **Volume** - Data placed on shares is written to volumes based on the instructions provided by the share objectives. Volume types include DSX volumes, remote NFS storage volumes, object storage volumes, or even volume groups.
- **Volume groups** - Used to group one or more storage systems, volumes, or volume groups so that they can be more easily targeted when configuring share objectives; default objectives will be created for each volume group.
- **Volume threshold** - Hammerspace volumes have a setting for high and low thresholds that are used to control volume utilization; the defaults are 60% (low) and 75% (high). Thresholds can be viewed and edited in the **Volume Properties** as shown in the following screenshot:





- **High threshold** - When a volume high threshold is reached, the cluster will stop placing data and start moving data off until usage falls below low threshold. If all volumes are at the high threshold, data will continue to be placed.
- **Low threshold** - When a volume low threshold is reached, the volume becomes a less preferred location to place data on. The cluster will, however, continue placing data on the volume until it reaches the high threshold.

## Share Default Objectives

The following are default objectives applied to each new share. In most cases, these objectives are left as is and you simply add your objectives. The purpose of these objectives is to define the default behavior for data placed on a Hammerspace cluster. In most cases, you will need to supplement these with your own, or modify them (where possible) to meet your own requirements to ensure that they are not counteracting your own objectives.

Objective	Condition
<b>keep-online**</b>	IS_BEING_CREATED OR (HAS_ONLINE_INSTANCE AND IS_RECENTLY_USED)?ALWAYS
<b>durability-1-nine**</b>	DATA_ORIGIN_LOCAL?ALWAYS
<b>durability-3-nines</b>	DATA_ORIGIN_LOCAL AND IS_DURABLE
<b>availability-1-nine</b>	DATA_ORIGIN_LOCAL?ALWAYS
<b>layout-get-on-open</b>	IS_BEING_CREATED OR HAS_ONLINE_INSTANCE
<b>keep-online</b>	IS_BEING_CREATED OR HAS_KEEP_ON_THIS_SITE
<b>optimize-for-capacity</b>	TRUE
<b>delegate-on-open</b>	TRUE

**\*\* This objective cannot be removed**



These default objectives can be broken down into three categories:

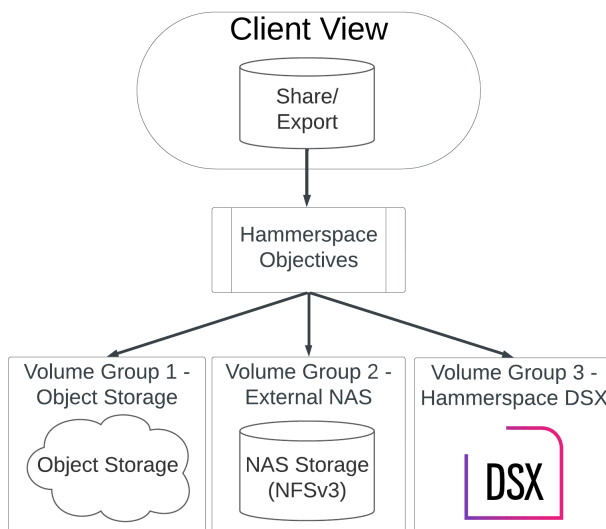
- Make sure new or active files are kept in online volumes for a period of time (objectives 1 and 6) but later tiered to offline volumes (if present) if that time has elapsed (objective 7).
- To ensure a minimum level of storage protection (objectives 2, 3, and 4).
- Set the default behavior for clients using the NFSv4.2 protocol (objectives 5 and 8).

These objectives are defined in greater detail in the [Objective Definitions](#) section of this document while the conditions are discussed in the [Using Metadata as Objective Conditions](#) section.

## Client Data Path Architecture



One of the differentiating features of Hammerspace is the relationship between the file share and the underlying storage that it places files on. A Hammerspace share is, in a way, just a virtual endpoint presented to clients to write data to. Where that data is actually written depends on the share objectives, which means that by simply interacting with a file, the client is indirectly influencing where data will be placed.

The following diagram illustrates how the client actually sees their data, versus the different ways in which the data may be placed. In this example, we have three potential volume group destinations, each containing volumes with specific characteristics. Files will ultimately be placed according to the objective configuration, which is constantly being re-evaluated in response to changes in file usage patterns, current date and time, and so on.



In this diagram, the client sees only the share itself. The client has no visibility into the objectives that control where the data is placed, and has no visibility into the actual volume where the file is located.

**Note:** If the only available file instance is located in an object storage (offline) volume, Windows users will see that as an offline file as shown in the following screenshot.

Name	Date modified	Type	Size
 conf.agent	11/4/2024 8:59 AM	AGENT File	1 KB
 data.json	11/4/2024 11:18 AM	JSON File	1 KB

## Applying Objectives

Objectives are applied on a per-share, per-cluster basis. In a global file share environment, each site will need its own objectives defined, and those objectives will need to be applied to the share at that site.

**Best Practice:** Always think of objectives on a per-site basis. Each site has different requirements, and the objectives only apply to the site at which they were applied.



Objectives can optionally be applied with conditions that control when they are activated. These conditions, written in the Hammerscript, can be based on the action being performed (file being created, file being opened, etc), the file characteristics (extension, name, etc.), or even the file type (is a snapshot, is a live file, etc.). The [Using Metadata as Objective Conditions](#) section of this document will review some of the most common metadata objects that objectives target.

Objectives can also be configured so that they can be overridden by other upstream objectives or removed by users with filesystem write attribute permissions; (TRUE) - the default setting, or not (ALWAYS).

**Note:**

- Always remember that just because an objective is applied, that does not mean that it can be enforced or will be enforced in real-time. Consider scenarios where a destination volume is full, or an object storage bucket is unavailable because Internet access is down. In these examples, the cluster will write the file to wherever it can, consider the file as unaligned, and attempt to uphold the objectives as soon as it can and align the file.
  - Alignment is discussed in greater detail in the next section.
- There may be times where one objective outweighs the other, which is why it is important to consider the goals of each applied objective on an individual basis.

## **Aligned, Partially Aligned, and Unaligned**

In the [Key Terms](#) section, we defined Aligned, Partially Aligned, and Unaligned. These terms are used to define if all, some, or none of the objectives that apply to a given file are currently being met. The Hammerspace GUI and Toolkit both can provide both high level alignment stats, measured at the root of the share, and the stats for individual files.

The following screenshot shows the File Alignment status for three different shares. The colored bars reflect alignment at the share level, and include green (Aligned), yellow (Partially Aligned), and orange (Unaligned).

Using the GUI file browser, you can see the status of individual files as shown in the following screenshots:

© 2025 HAMMERSPACE



If you click on the colored dot, a window will open that shows more detailed information regarding the alignment status. In the following screenshot, we see that the file aligned is blocked because a place-on action has not yet occurred:

File Alignment	
Placement	
Place On	●
Confine To	●
Exclude From	●
Performance	
Read Bandwidth	●
Read IOPS	●
Read Latency	●
Write Bandwidth	●
Write IOPS	●
Write Latency	●
Protection	
Availability	●
Durability	●
Online Delay	●

## Hammerspace Condition Wizard

When editing a share to apply objectives, the Hammerspace condition wizard can assist you in writing the Hammerscript for the optional conditions that control if an objective is applied. The wizard supports the most common Hammerspace conditions.

The following screenshot is an example of how to use the condition wizard to build a Hammerscript expression. Select a value from the drop-down menu underneath **Step 2: Define conditions**. Complete the fields to build the condition (`FNMATCH("*.orbit",NAME)?TRUE`).





Step 2: Define conditions

FILE_NAME	Like	*.orbit	+
-----------	------	---------	---

☐ ALWAYS ☒ TRUE ☐ NEVER

FNMATCH("\*.orbit", NAME)?TRUE ✓

☐ Add Another Dismiss Apply

## Condition Wizard - Writing Hammerscript

Searching for other possible values, like tags in this case, can be completed in the wizard as shown in the example below. To use these condition values, you must understand their syntax, which is defined for many common conditions in the [Using Metadata as Objective Conditions](#) section of this document.

Step 2: Define conditions

Operand	Operator	Value	+
---------	----------	-------	---

☐ ALWAYS ☐ TRUE ☐ NEVER

HAS\_T|

- HAS\_TAG
- HAS\_TAG\_INHERITED
- HAS\_TAG\_LOCAL
- HAS\_TAG\_OBJECT



## Objective Definitions

The following is an overview of some of the objectives discussed in the design scenarios in this document, and examples of how they are used. These are in addition to those described in the [Share Default Objectives](#) section of this document.

This document only covers a subset of the objectives, specifically those which are most commonly used. For a full list of Hammerspace Objectives, consult the *Hammerspace Administration Guide*.

### Placement Objectives

There are multiple objectives that will direct the placement of files on a Hammerspace cluster. In this section, we will discuss those objectives, and provide examples of how they are used.

**Note:** Placement objectives are not required to access remote files in a global file share environment. If a client at a site opens a file that is not available on volumes locally, that file will be mobilized to that site and kept online as long as it is open, or when closed as long as the objectives dictate.

**Best Practice:** In a global file share environment, objective designs should consider what will happen if the connection between sites goes down.

- If you rely primarily on on-demand mobilities to transfer files to users across sites when requested, any interruption in site to site communication will cause files not online at that site to be temporarily unavailable. It is important to consider this when architecting your objectives.
- If you will use placement or **keep-online** objectives to ensure that each site will maintain access to all files in the event another site is unavailable due to disaster or maintenance, every share in a site must have a **keep-online**, **place-on**, or **confine-to** objective that targets all files. The destination of the files does not matter. All that matters is that the sites have



objectives that direct the files to be placed on or kept online somewhere.

- The most common method to ensure that all sites can access all files, even when they cannot reach each other, is to configure all sites to write an instance file to the same shared object storage volume. The volume will contain only 1 instance of the file, but each site needs to be told to try and write its own instance to it. Note that if you only target a subset of your files, it is likely only that subset will be reachable if communication between sites is interrupted.

### **Objective: Keep-online**

This objective will place data on any file-based (online) volume, including DSX-based volumes or third-party NFS volumes in the cluster. It is also automatically assigned to data that is modified and/or created, though if only the default objectives and share settings are used, data can be moved to offline storage volumes in as little as 5 minutes.

**Best Practice:** Keep-online is the simplest way to ensure that files are kept in an online storage volume, though it will use any available volume to do so. Most of the exercises in this document will focus on other techniques for keeping important files online.

### **Objective: Place-on and Confine-to**

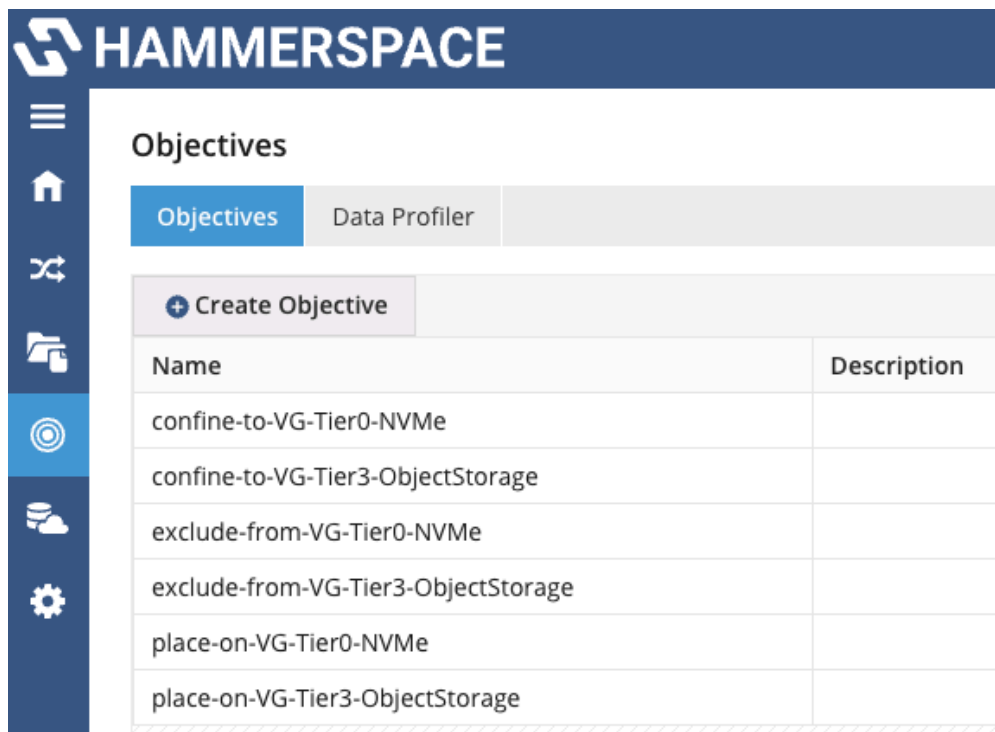
In this section, we will review how we explicitly define where data will be placed.

By default, a new Hammerspace share will try to place newly created data on any available online volume. This is achieved using a default **keep-online** objective with an **IS\_BEING\_CREATED OR (HAS\_ONLINE\_INSTANCE AND IS\_RECENTLY\_USED)?ALWAYS** condition that ensures that new files are kept in an online volume for at least 5 minutes.

The two primary objectives for explicitly stating where data will be placed are **place-on** and **confine-to**. **Place-on** asks for **at least one** instance of a file to be placed on the specified location, while

confine-to restricts **all** instances to be within the specified location.

**Best Practice:** Hammerspace recommends that you create Volume Groups that contain the target volumes, and use the default place-on and confine-to objectives that will be created for those volume groups. The following screenshot shows the default objectives that were created for two different volume groups.



The screenshot shows the HAMMERSPACE interface with the 'Objectives' tab selected. A table lists the following objectives:

Name	Description
confine-to-VG-Tier0-NVMe	
confine-to-VG-Tier3-ObjectStorage	
exclude-from-VG-Tier0-NVMe	
exclude-from-VG-Tier3-ObjectStorage	
place-on-VG-Tier0-NVMe	
place-on-VG-Tier3-ObjectStorage	

If no conditions are used, these objectives will apply to all files in the share, including those located in snapshots. If an objective cannot be enforced, the ways a cluster may react will vary. Some examples using the above objectives include:

- If the **Tier0** (online) volume group is the target and has reached the high threshold, the cluster will evacuate either the largest or oldest files to the **Tier3** (offline) volume group to free up space, and then write the file to **Tier0**.



- If the **Tier3** (offline) volume group is the target and has reached the high threshold, the cluster will write the files to the **Tier0** (online) volume group and keep checking to see if the **Tier3** frees up enough space to be below the low threshold. If the available space is below the low threshold, the file will be moved there.

### Objective: Exclude-from

The **exclude-from-\*** objective is used to prevent the specified volume group from being used. A common use case for exclude-from is when you are using keep-online, availability, and durability objectives that don't target specific volumes/volume groups, but you still want to exclude certain ones from being used.

**Best Practice:** If it's critical that a particular volume not be used, say to adhere to European GDPR regulations, consider using an exclude-from objective.

### Tiering Data Using Objectives

One of the most common uses for objectives is to tier data to multiple storage classes. This is often used to tier data off expensive online volumes to much cheaper object storage volumes or even slower online volumes.

In the following example, we will use one of the more common metadata values for determining when data should be tiered, `LAST_USE_AGE`. Our target volume groups will be the ones used in the previous section.

Objective	Condition
<b>place-on-VG-Tier0-NVME</b>	<code>LAST_USE_AGE &lt;= 30*DAYS</code>
<b>place-on-VG-Tier3-ObjectStorage</b>	<code>LAST_USE_AGE &gt; 30*DAYS</code>

The behavior of these objectives is fairly straightforward:

- If a file was last closed within the past 30 days (including exactly 30 days), it will be placed in VG-Tier0-NVME (a volume group containing online storage volumes).



- If a file was last closed more than 30 days ago, it will be placed in VG-Tier3-ObjectStorage (a volume group containing object storage volumes, which are considered an offline storage volume).

## Data Availability and Protection Objectives

In this section, we will review objectives related to data availability and protection.

### Objective: Availability-#-nines

Three Availability objectives enable the admin to specify how many nines of Availability the data should have. Each storage volume is assigned an Availability value. Data will be placed either on a storage volume that meets the Availability number (minimum is 1) or multiple file instances may be created to meet the requested number of nines.

**Note:** If you configure a share with an availability objective that causes multiple file instances to be created, and the Hammerspace volume containing some or all of those files becomes marked as SUSPECTED (unavailable) within the cluster, the cluster will resilver the impacted files after 30 minutes by default. If the volume SUSPECTED status later clears, the cluster will automatically remove any excess file instances.




The default Availability objectives allow you to specify either 1, 3, or 5 nines. Custom objectives can be created that allow for up to 9 nines.

### Best Practices:

- Use availability objectives to control how many file instances will be created so that if a storage volume becomes unavailable, a file can still be read from some other volume. The availability objective will always try to place the file instances on different failure domains (meaning different hosts) to ensure this happens.
- Double-check your volume availability and durability values! While Hammerspace does apply defaults, not all volumes are created equal. For example, a self-hosted object storage volume



on a standalone S3 server is likely less durable and available than one managed by AWS, Google, or Microsoft Azure, or a DSX volume backed by a RAID is better than one that is not. To edit the settings, click the pencil icon to the right of the volume name for cluster **Infrastructure**:

Infrastructure										
Volumes   Volume Groups   Storage Systems										
+ Add Volume				Q Search						
	Name	Activity (10 M...	Volume Group	Storage Syste...	Used	Free	Size	Read O...	Percentage Used	Actions
✓	AZ1:1	0 IOPS	alldsx, all	dk-a-dsx-1.as...	641 MB	85.2 GB	85.8 GB	No	1%	  

Update the **Availability** or **Reliability** values as needed:

Protection	Defined
Availability	99% (2-9s) ▼
Durability	99.9% (3-9s) ▼
Online Delay	—

**Best Practice:** The volume Durability value should always be larger than the Availability value.

### Volume Availability - Default Values

Hammerspace volumes are assigned the following default Availability values that can be changed if required:

- All DSX or remote NFS (online) volumes: 2
- All Object storage (offline) volumes: 4

**Note:** Based on these default values, the share default objectives will only create one copy of every file.

### Objective: Durability-#-nines

Three durability objectives enable the admin to specify how many nines of Durability the data should have. Each storage volume is assigned a



Durability value. Data will be placed either on a storage volume that meets the Durability number (minimum is 1) or multiple file instances may be created to meet the requested number of nines.

**Note:** If you configure a share with a durability objective that causes multiple file instances to be created, the cluster will **only** resilver the impacted files if the Hammerspace volume is decommissioned or failed. Decommissioning a volume gracefully moves the file instances it contains to other volumes, while failing a volume marks all file instances that it contains as permanently lost. Hammerspace volumes are typically only failed if their underlying storage has failed.

The default Durability objectives allow you to specify either 1, 3, or 5 nines. Custom objectives can be created that allow for up to 9 nines.

**Best Practice:** Durability and availability are calculated separately. Based on the cluster volume configuration, each may require a different number of file instances to be created.

### Volume Durability - Default Values

Hammerspace volumes are assigned the following default Durability values that can be changed if required:

- All DSX or remote NFS (online) volumes: 3
- All Object storage (offline) volumes: 9

**Note:** Based on these default values, the share default objectives will only create one instance of every file.

### Objectives: Versioning, Undelete, and File Conflict Resolution (Log-Xfer)

The versioning, undelete, and site versioning (log-xfer) objectives enable more granular control over data recoverability. They use the same methods as Hammerspace snapshots to accomplish this, but unlike snapshots, they are applied as an objective rather than as a share option.

**Note:**





- As with all objectives, these can be applied at the root of the share or within the contents of the share. This enables you to vary the settings within the share folder structure.
- If you use any of these objectives with a single-site share, meaning a share that is **not** configured as a global file system, you **must** schedule a recurring snapshot for the target share to ensure that the expired files will be removed.
  - Shares that are members of a global file system use snapshots internally to assist with file replication, these snapshots are able to remove these expired files.

These objectives are available in **1-hour**, **1-day**, **1-month**, and **1-week** versions, the objective names and definitions are as follows:

- **Log-xfer-\*** - Used only with shares that are part of a global file system; it retains a version of the file when it is saved in a different site than the one where the file was last modified. The last site that modified the file is also considered the owner.
- **Undelete-\*** - Retains files that have been deleted.
- **Versioning-\*** - After a file has been closed for 3 minutes, the versioning objective retains the previous version of the file.

Each of these objectives places their saved instance of the files in the share **.snapshot/current** directory. The files are removed once both the objective retention period elapses and the next snapshot expires.

**Best Practice:** Hammerspace recommends using exclusions to prevent unneeded files from being targeted by these objectives. For example, a file that frequently updates a temporary file might cause excessive file versioning, which would needlessly consume cluster resources. When applying these objectives using the **Share Properties | Advanced** menu (described in the next section), you are presented with some of the more common extensions that customers typically exclude. These can be customized based on your own needs, which Hammerspace recommends to ensure that your data protection requirements are still being met.



## Apply using the Share Properties | Advanced menu

To set these options at the root of a share, you can either add them as ordinary objectives or edit the share and configure them under **Share Details | Advanced** as shown. Note that the true objective names are not used here, though it is still the same objectives that will be applied.

Advanced

Access-based enumeration	<input type="checkbox"/> Enabled
Access time update	<input type="checkbox"/> Enabled
Retain deleted files	<input checked="" type="checkbox"/> Enabled
Create versioned files	<input type="checkbox"/> Enabled
File conflict resolution	<input type="checkbox"/> Enabled
WORM	<input type="checkbox"/> Enabled

The following screenshot shows the default extensions that will be excluded by each of these objectives. As stated before, review this list and remove or add extensions or file names as needed. If the file name is not complete, you must use wildcards to ensure that it will be matched.

Retain for: ☒ 1 Hour ☐ 1 Day ☐ 1 Week ☐ 1 Month

Exclude Files: \*.tmp \*.TMP .nfs\* \*.swp \*.swx \*.asd ~\*.docx ~\*.xlsx ~\*.pptx \*.lck \*.dwl2



## Advanced Objectives

This section will review other objectives that are unrelated to data placement, but are among the most common that are required.

### Objective: Access-based-enumeration

Enables Access-Based Enumeration (ABE) for both NFS and SMB clients, preventing users from seeing files and folders they don't have permission to access.

If adding the ABE objective to a share, Hammerspace also recommends adding the `DO-NOT-CACHE` objective to ensure that client permissions are not cached. Cached permissions prevent ABE from receiving permissions updates in real time, and can cause it to display files and folders that a user can no longer access.

**Note:** ABE requires additional cluster resources to continually assess and deliver a custom share view to each user. It should not be used unless explicitly required.

### Objective: Acl-compatibility-mode-ignore-chmod

The `acl-compatibility-mode-ignore-chmod` objective blocks permission changes from Unix clients; it is commonly used with shares with specific NFS security requirements.

### Objective: Acl-compatibility-mode-non-posix

The `acl-compatibility-mode-non-posix` objective should be added to any share that is either used only by SMB clients, or will be used by both SMB and NFS clients. This objective changes the Access Control List (ACL) compatibility mode to prevent the creation of inverse ACLs common in NFS-only POSIX file systems. If present, these ACLs can cause unexpected denies for SMB clients.

**Best Practice:** Contact Hammerspace support if you want this setting enabled at the cluster level, which is recommended if all shares will have either SMB-only or SMB and NFS clients. This ensures that when



new shares are created, they automatically have this setting applied. No objective is required.

**Objective: Do-not-cache**

The **do-not-cache** objective prevents a share from caching client permissions to ensure it always has current data regarding user permissions (including group memberships). If using access-based-enumeration, add this objective to ensure that the view of the filesystem is always current.

**Objective: Layout-get-on-open**

Controls the layout behavior for NFS 4.2 protocol. Leaving it on generally improves the performance when files are opened and then written to. If using NFS v3 or SMB, this indirectly impacts the behavior as the DSX node will be holding the layout vs. the client.

**Objective: No-atime**

The **no-atime** objective prevents a share from updating the access timestamps of files. It is most commonly used in shares or folders where the atime of files is continually updated, and therefore has limited use for objectives or auditing. This also has the benefit of reducing the metadata update workload on the cluster for those files.

**Note:** Disabling atime updates is not recommended if your file auditing requirements dictate that all timestamps be accurate regardless of their usage patterns.

**Objective: Optimize-for-capacity**

This objective encourages files to move towards Object/Cloud storage whenever possible after the **keep-online** objective has expired and/or files are no longer in use.

**Best Practice:** If your Hammerspace cluster has an attached object storage volume, and you notice that files are moving to offline object storage much faster than you want, consider removing the share default **optimize-for-capacity** objective OR adding a **keep-online** or **place-on** objective that explicitly defines how long to keep data in online volumes. In nearly every case, you will be adding your own **place-on** or



**keep-online** objective, so this default objective can be left in place. Alternatively, you can also modify the share `RECENTLY_USED_DURATION` value which will extend how long files are kept online by default. Refer to the [IS\\_RECENTLY\\_USED](#) section of this document for additional details.



## Using Metadata as Objective Conditions

In this section, we will review some of the more common metadata values that are used when setting conditions for objectives.

An example of the full metadata output of a file is provided in the [Appendix](#), and can be obtained using the Hammerspace toolkit command.

```
$ hs eval -e this telemetry.txt
```

## Understanding the Document Code Blocks

You will see three different styles of codeblocks in the examples provided in the remainder of this document. These include the following.

The `hs` command indicates that this command is part of the Hammerspace Toolkit, which is available for Windows, Linux, and Mac:

```
$ hs eval -e this orbit.txt
```

The `admin@cluster>` prompt indicates that the command is executed from the Hammerspace CLI, which is available when logging into a Hammerspace cluster management IP address/FQDN using SSH and with an account with the proper access.

```
admin@NYC> label-create --name "Project Gemini"
```

**Note:** Consult the *Hammerspace Administration Guide* for information regarding the Hammerspace CLI.

Code blocks that contain examples of Hammerscript will lack any sort of command prompt, and contain text similar to the below.

```
# Match files named jupiter*.* (case sensitive)
FNMATCH("jupiter*.*",NAME)
```



## What is Metadata?

Simply put, metadata is data or information about other data.

Anybody who has used a modern computer and operating system is familiar with concepts of files and directories within a file system. Directories and file names are the most basic metadata that we use on a daily basis. They are the most basic structures of a POSIX file system. POSIX also includes basic file properties such as:

- Size
- Ownership (user and group)
- Permissions (Unix style rwx or an Access Control List)
- Timestamps, including last access time (atime), last time the file contents were modified (mtime), and last time the file metadata or inode (think file header) was changed (ctime). Some also include a birth or create time.

## Hammerspace Metadata

Hammerspace manages three classes of metadata:

- POSIX metadata including directory structure, file names, and basic file properties
- Internal – Statistics, instance location, etc.
  - More detailed file usage data (beyond atime and mtime)
  - Extensive inode information – e.g. volume(s) and path(s) to instances
- Custom rich metadata that can be defined by the administrator or user
  - Can be manually created or extracted and applied by script
  - Customer or third-party tools and applications can directly access and manipulate metadata
  - This document discusses four examples: tags, attributes, labels, and keywords



## Commonly Used Metadata Values

In this section, we will explore the most common metadata values used to control the application of objectives, and (when applicable) how that metadata is set.

### Filename or Extension

One of the most useful metadata values for Hammerspace Objectives are the names and extensions of the files themselves. By using the FNMATCH command and wildcards, we can create conditions that match either or both.

**Note:** FNMATCH commands are **case-sensitive**, so you may need to include multiple variations of the same pattern.

The following example shows a condition that matches all files that start with **mars**:

```
# Match files named mars*.* (case sensitive)
FNMATCH("mars*.*",NAME)
```

**Note:** Hammerspace conditions are written in the Hammerscript scripting language.

This example matches all files that have an **orbit** extension:

```
# Match files with the extension *.orbit (case sensitive)
FNMATCH("*.orbit",NAME)
```

You can also create FNMATCH statements for multiple values. The following is an example of a negative match for 2 extensions, **orbit** and **launch**.

```
# Match files that do NOT have an *.orbit or *.launch extension (case sensitive)
SUM({||#A=FNMATCH({{"*.orbit"; "*.launch"}[ROW],NAME)}[2])=0
```

Negative matches are useful when you want to exclude certain files from having the objective applied.





**Note:** The [2] in the above pattern equals the number of items to match (or not match), it should be incremented if adding more values.

To make the above a positive pattern match, change the 0 to a 1 as seen in the following example:

```
# Match files that have an *.orbit or *.launch extension (case sensitive)
SUM({||#A=FNMATCH({{"mars.*"; "*.orbit"}[ROW],NAME)}[2])==1
```

If you need to add additional patterns, simply add them to the list, and increment the value [3] that tracks how many are present as shown in the following three-filename negative pattern match example:

```
# Match files that do NOT have an *.mars, *.orbit, or *.launch extension (case sensitive)
SUM({||#A=FNMATCH({{"mars.*"; "*.orbit"; "*.launch"}[ROW],NAME)}[3])==0
```

## Folder / Path

Folder paths can be matched using a similar format as file names and extensions, though the type is **PATH**, not **NAME**. The example below would match all folders named jupiter, excluding or including the contents as required:

```
# Match all folders named "jupiter" and include the contents (case sensitive)
FNMATCH("*/jupiter/*",PATH)
```

You can also create multi-path matches, similar to file names. The below example is a negative pattern match for **mars** and **venus** folders:

```
# Match all folders NOT named "mars" or "venus" and include the contents (case sensitive)
SUM({||#A=FNMATCH({{"*/mars/*"; "*/venus/*"}[ROW],PATH)}[2])==0
```

**Note:** As always, these values are **case-sensitive**. Add more folder name variants if needed to ensure you match folders whose case is different. Also, change this to a positive pattern match by changing the 0 to a 1.



## Timestamps

Hammerspace retains all the typical file timestamp attributes that you are familiar with:

- [CREATE\\_AGE](#) - The file creation time; not a POSIX timestamp but referred to as `crttime` in XFS
- [CHANGE\\_AGE](#) - The last time the file inode information was changed (example: file permissions were changed); `ctime` in POSIX
- [ACCESS\\_AGE](#) - The last time the file was accessed; `atime` in POSIX
- [MODIFY\\_AGE](#) - The last time the file was modified; `mtime` in POSIX

It is important to note that these timestamps are global. With global file shares, any objectives that target timestamps will be impacted by file operations, no matter where they occur.

**Best Practice:** Hammerspace assimilations will cause the file `ACCESS_AGE` attribute to be set to the time the file was assimilated. If you intend to do timestamp-dependent placement of files, you may need to alter which timestamp you initially use. This scenario is described in further detail in the [LAST\\_USE\\_AGE](#) section of this document since that attribute is also impacted.

## LAST\_USE\_AGE

The `LAST_USE_AGE` attribute is equal to the time a file was last closed, regardless of whether or not it was edited. If for some reason the value is not set, `LAST_USE_AGE` uses the more recent of either file access or modification times.

Hammerspace recommends using `LAST_USE_AGE` for any time-based tiering of files to ensure files that are either read or modified are kept online for the period specified. For comparison, if you were to use `MODIFY_AGE`, files that were opened but not modified would only remain online for the `IS_RECENTLY_USED` default duration of 5 minutes.

A condition for a single time period that includes `LAST_USE_AGE` will look similar to this:



```
# Match all files that were last closed 30 or more days ago  
LAST_USE_AGE >= 30*DAYS
```

If you are trying to specify a range of time, perhaps to place a middle tier of files, you would connect two values with an AND statement:

```
# Match all files that were last closed from 7 to 30 days ago.  
LAST_USE_AGE<=30*DAYS AND LAST_USE_AGE>=7*DAYS
```

## Timestamp Best Practices

Hammerspace recommends using LAST\_USE\_AGE value, as using MODIFY\_AGE will not target files that were opened but not edited, which in some scenarios may not be the behavior you want.

For example, consider what would happen if you have objectives that move data between online to offline storage based only on the MODIFY\_AGE value. You now have a file that needs to be moved from offline to online storage to be opened, but it was not modified. Based on the other default objectives, that file would likely be moved back down to offline storage after 5 minutes. If this file were opened again, the cycle would repeat itself. At the very least, this will lead to slower file open times and (if frequent enough) complaints from end users.

Additionally, the ACCESS\_AGE value should not initially be used for files that Hammerspace has assimilated since that value will initially be set to the time of assimilation. As time passes, and the true file ACCESS\_AGE is established, it may be a suitable value for your time-based conditions.

## Custom Metadata

Hammerspace provides four types of custom metadata: tags, attributes, labels, and keywords. The acronym TALK can be used to remember them. The distinction between the types is two dimensional:



- Whether there is a predefined schema or set of metadata the user can apply
- Whether the type uses a simple string or key-value pair

Metadata Type	Schema	Non-Schema
Simple Strings	Label	Keyword
Key-Value Pairs	Attribute	Tag

A common use case for custom metadata is to enable end users to perform an action that causes an existing objective to be applied. Some real world examples of this include:

- If a file has the label “archive”, then place it on an object storage volume and no longer keep an instance online.
- If a file has the tag “render”, then keep an instance online in the cloud render farm site.
- If a file has the keyword “compliance”, then write an instance to both an online and object storage volume.

Attributes are defined by Hammerspace and are generally not configurable by the administrator, although they can be applied and values set by the user.

The remainder of this section focuses on the three most commonly used custom metadata values: Labels, Tags, and Keywords.

## Labels

Labels are a built-in metadata object that you can use to rigidly define the values you want to use for classifying your data on Hammerspace. While the label metadata is available globally, you must create them on each participating cluster and be sure to use the exact same name and case if you want users to interact with labels.

**Note:** In Hammerspace global file share environments, labels must have the same serial number on each cluster. When creating labels, verify that the **Internal** IDs match. If the IDs do not match, use the



`label-delete` command to delete the label on the cluster with the lower serial number, and re-add it until the serial number matches.

### Creating Labels

To create a label, use the CLI `label-create` command:

```
# Create a label named "Project Apollo"
admin@NYC> label-create --name "Project Apollo"
Name:                Project Apollo
Internal ID:         4
```

If users will interact with the labels, run the same command on other participating clusters. Note the different serial number in this example:

```
admin@LA> label-create --name "Project Apollo"
Name:                Project Apollo
Internal ID:         1
```

In this case, we would need to delete the label on the LA cluster, re-add it, and then delete and re-add it two more times until the **Internal ID** was 4. Here is the last set of delete and create commands that shows our tag with **Internal ID: 4**:

```
# Delete the label named "Project Apollo"
admin@NYC> label-delete --name "Project Apollo"
success

admin@NYC> label-create --name "Project Apollo"
Name:                Project Apollo
Internal ID:         4
```

### Creating a Label with Implied Labels

An implied label is one that will be added in addition to the current label. To create a label that includes implied labels, use the `--implied-labels` option. This is useful in scenarios where you have labels that are targeted by a variety of objectives, and you want to simplify applying multiple labels with only one command for users.

```
# Create a label named "Mission Alpha", with an implied label of "Project Apollo"
```



```
admin@LA> label-create --name "Mission Alpha" --implied-labels "Project Apollo"
```

**Note:** You first must create the target implied labels before you create a label that references them. The rules regarding label internal ID's still apply for all labels.

To associate a label with multiple parent labels in the hierarchy, specify multiple implied labels as shown in the following example:

```
admin@LA> label-create --name <label> --implied-labels <parent1, parent2 ...>
```

### Applying Labels

As with other customer metadata, labels can be applied to a file or folder.

```
# Add label "Mission Alpha" to file telemetry.txt
$ hs label add "Mission Alpha" telemetry.txt
```

Remember that we defined "Mission Alpha" to have an implied label. In the example below, we see that both labels are present.

```
# List all applied labels for file telemetry.txt
$ hs label list telemetry.txt
LABELS_TABLE{
  | LABEL_ = LABEL('Mission Alpha'),
  | COUNT = 1;

  | LABEL_ = LABEL('Project Apollo'),
  | COUNT = 1}
```

### Conditions Examples - Labels

The following matches the example label (Mission Alpha) from this section:

```
# Match files that have the label "Mission Alpha" (case sensitive)
HAS_LABEL(LABEL('Mission Alpha'))
```



## Tag

Tags do not use a schema, meaning they are free-form. When using tags, it is important to have well-defined standards as the objective conditions must reference the exact tag name, including case. Tags are a key-value pair; if specifying a value, the value must be passed to the Hammerspace toolkit (hstk) in quotes, meaning the quotes must be “escaped” by using outer quotes or backslashes.

In these examples, the outer double quotes “escape” the inner single quotes from the shell. If no value is specified, the tag will be set to true.

### Applying Tags

Tags are added to files using the `hs tag set` command:

#### *Tag with Value Specified*

In this example, the tag name is *Project* and the value is *Space Launch*; the (required) outer double quotes “escape” the inner single quotes from the shell:

```
# Set tag "Project" with the value "Space Launch" on file telemetry.txt
$ hs tag set Project -e "'Space Launch'" telemetry.txt

# List tags for file telemetry.txt
$ hs tag list telemetry.txt

TAGS_TABLE{
  |NAME = "Project",
  |VALUE = "Space Launch"}
```

#### *Tag with Default Value of True*

In this example, the tag name is *Space Launch*; since no value is specified, the tag will be set to true:

```
# Set tag "Space Launch" with the default value "TRUE" on file telemetry.txt
$ hs tag set 'Space Launch' telemetry.txt

# List tags for file telemetry.txt
```



```
$ hs tag list telemetry.txt
```

```
TAGS_TABLE{  
  |NAME = "Space Launch",  
  |VALUE = "TRUE"}
```

### Condition Examples - Tags

The format used to match a tag within an objective condition will vary based on whether you want to check for the presence of a tag, or both the presence of a tag and its value. Examples of both are provided below.

#### Condition Example - Tag and Value

This example matches the example tag (**Project**) that has a value (**Space Launch**):

```
# Match files that have the tag "Project" with the value "Space Launch" (case sensitive)  
GET_TAG("Project")== "Space Launch"
```

#### Condition Example - Tag

This example matches the example tag (**Space Launch**) with any value:

```
# Match files that have the tag "Space Launch" (case sensitive)  
HAS_TAG("Space Launch")
```

### Delete Tags

You can delete tags using the `hs tag delete` command; you do not need to supply the tag value (if present):

```
# Delete tag "Space Launch" from file telemetry.txt  
$ hs tag delete 'Space Launch' telemetry.txt  
  
# List tags for file telemetry.txt  
$ hs tag list telemetry.txt  
  
TAGS_TABLE{}
```





## Keyword

Keywords do not use a schema and are free-form. It is important to have well-defined standards as the objectives must match the keyword exactly.

### Applying Keywords

Tags are added to files using the `hs keyword add` command:

```
# Add keyword "mars" to file telemetry.txt
$ hs keyword add mars telemetry.txt

# List keywords for file telemetry.txt
$ hs keyword list telemetry.txt

KEYWORDS_TABLE{
  |KEYWORD = "mars"}
```

### Conditions Examples - Keywords

Keywords are matched using the `has_keyword` command:

```
# Match files that have keyword "mars" (case sensitive)
HAS_KEYWORD("mars")
```

## Advanced Metadata Attribute Values

In this section, we will review some additional metadata attribute values that are either part of the default share objectives, or are used in common scenarios. To explore the full list of default file attributes, refer to the [Appendix](#) section [File Metadata Example \(Full\)](#).

### DATA\_ORIGIN\_LOCAL

The **DATA\_ORIGIN\_LOCAL** condition would be true for files that were either created or modified on a share at the local site. In a global file share environment, the last site to create or modify a file will be the only one where **DATA\_ORIGIN\_LOCAL** is true.



**Note:** If **DATA\_ORIGIN\_LOCAL** is true, that means that the site is also the owner of the file.

This condition is used with multiple default share objectives regarding durability and availability to ensure that this default objective only applies to files located on this site, and does not cause files from other sites to be copied to this cluster unnecessarily.

### **DATA\_ORIGIN\_REMOTE**

In a global file share environment, **DATA\_ORIGIN\_REMOTE** condition matches files on the local site that were either created or modified on a share in a remote site.

### **HAS\_KEEP\_ON\_THIS\_SITE**

**HAS\_KEEP\_ON\_THIS\_SITE** matches files or folders that have had the local Hammerspace cluster name added to the participant site table within the file metadata and is one of the conditions used with the default **keep-online** objective that is applied to all Hammerspace shares. Remember that if you are using the default Hammerspace share objectives setting, this value will cause data to be kept online in the site specified.

This value can be set using the Hammerspace toolkit using the following command:

```
# Mark file telemetry.txt with a keep-on-site designation for site <site-name>
$ hs keep-on-site add <site-name> telemetry.txt
```

**Note:** To remove the site, replace **add** with **delete** in command above.

A list of sites can be obtained using the following command:

```
# List all sites where the current share is available
$ hs keep-on-site available
```



**Note:** If the data is in a share that is not a global file share, only the local cluster name will be returned.

### **HAS\_ONLINE\_INSTANCE**

**HAS\_ONLINE\_INSTANCE** matches files that have an online instance, such as on a connected NFS storage volume OR a DSX node. Files that are located in object storage do not meet this requirement.

### **IS\_BEING\_CREATED**

**IS\_BEING\_CREATED** is a condition that applies to newly created files, including those that were copied into a share.

### **IS\_DURABLE**

**IS\_DURABLE** is true when a file is located on a volume(s) that meet the durability requirements as defined by the objectives.

### **IS\_LIVE**

**IS\_LIVE** is a metadata value used for files that are in the live file.

### **IS\_SNAP**

**IS\_SNAP** is set for files that are located in a snapshot and is frequently used in global file system environments to copy snapshot data to an object storage volume, providing a means for an “off Hammerspace” placement of snapshot data.

Despite being configured at only one site (Hammerspace recommendation), snapshots occur throughout the global file system. It is essential to consider the placement of those files that are placed into the snapshot.

### **IS\_RECENTLY\_USED**

By default, the **IS\_RECENTLY\_USED** value is set to 5 minutes and is used as part of all default share objectives. This ensures that new files on a share are kept online for at least some period of time.



The default value can be changed on a per-share basis using the Hammerspace toolkit. Below is an example command that would be executed from within the root of the share to change it to one day:

```
# Set the recently used duration value for the current folder and its contents to 1 day
$ hs attribute set RECENTLY_USED_DURATION -e "1 day"
```

To verify the current setting, execute the following command from within the target folder:

```
# Get the recently used duration value for the current folder (note that it may be originally set on a parent folder)
$ hs attribute get RECENTLY_USED_DURATION
```

This is a simple way to change the default settings for a share without needing to actually change the objectives themselves.



## Combining or Modifying Conditions

In this section, we will explore how we combine multiple conditions into longer statements, and how we apply a negative match to conditions. With just a basic understanding of how conditions can be combined or modified, you will quickly learn how to craft complex ones that target exactly the data you need.

### Marking a Condition as a Negative

By using a **!**, you can apply a negative to many conditions. This is useful when you want to use a condition to exclude files, rather than target them.

Here are some examples of negative matches that will exclude files.

Match files that do NOT have the **Space Launch** tag:

```
# Match files that do NOT have the tag "Space Launch" (case sensitive)
!HAS_TAG("Space Launch")
```

Match files that do not start with **mars** (lowercase):

```
# Match files that do NOT have the name "mars*.*" (case sensitive)
!FNMATCH("mars*.*",NAME)
```

Match files that do not have the **Project Apollo** label:

```
# Match files that do NOT have the label "Project Apollo"
!HAS_LABEL(LABEL('Project Apollo'))
```

### Build a Conditional Statement using Conditions

Objective conditions do not need to be singular. You can combine them with **if** or **or** statements and even **( )**. In this section, we will review three different conditions that also leverage conditional statements.



**Note:** Each of the examples uses `<=` or `>=` conditionals for date statements, meaning that the times referenced will also match the condition. Remove the `=` if you do not want to match the time referenced, and only the value that precedes or follows it.

### Example 1

Desired match: Live files only (no snapshot data), used within the last 6 months with the "Space Launch" tag OR contained within the jupiter folder and used within the last 30 days.

**Equivalent condition written in Hammerscript:**

```
IS_LIVE AND ((LAST_USE_AGE<=6*MONTHS AND HAS_TAG("Space Launch")) OR  
(FNMATCH("*/jupiter/*",PATH) AND LAST_USE_AGE<=30*DAYS))
```

### Example 2

Desired match: All files that start with `mars` (all lowercase) OR files with the `orbit` extension that were created less than 60 days ago.

**Equivalent condition written in Hammerscript:**

```
FNMATCH("mars*.*",NAME) OR (CREATE_AGE <= 60*DAYS AND FNMATCH("*.orbit",NAME))
```

### Example 3

Desired match: Live files last used between 6 months and 1 year ago.

**Equivalent condition written in Hammerscript:**

```
IS_LIVE AND (LAST_USE_AGE>=6*MONTHS AND LAST_USE_AGE<=1*YEARS)
```



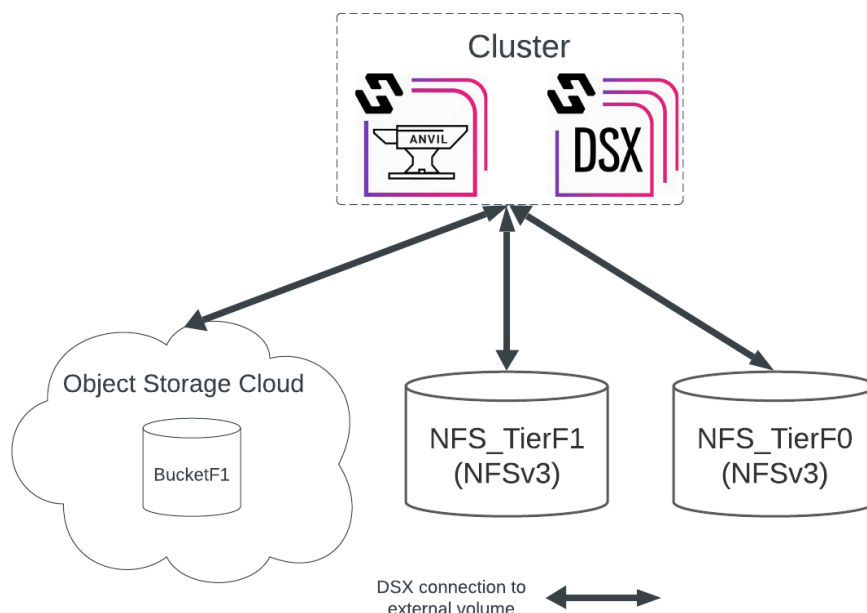
## Objective Scenarios and Solutions

The first step you should take before determining what objectives and conditions to use is simply to write down what you want in plain language. You need to do this for every site, as objectives are created and applied on a per-site basis.

In this section, we will review four common Hammerspace deployments, and show how we turn plain-language requirements into Hammerspace Objectives. We will map the plain-language data placement requirements to their actual share objectives in each example. In many cases, you will see that a single objective with multiple conditions satisfies multiple individual data placement requirements.

## Scenario #1: One Site - Three-Tier NAS

In this scenario, the customer has one site and wants to leverage multiple tiers of local (online) storage, and a cloud tier for older data and data recovery. The following diagram shows their proposed Hammerspace infrastructure:



The objective packages are the same for all shares.

## Requirements

- Single cluster
  - 3x DSX nodes, no local storage volumes
  - Connected to NFSv3 NVMe volume: NFS\_TierF0
  - Connected to NFSv3 SSD volume: NFS\_TierF1
  - Connected to object storage: BucketF1
- Data placement and other requirements
  1. Data used within the last 14 days is placed on NFS\_TierF0
  2. Data used within the last 14 to 45 days is placed on NFS\_TierF1
  3. Data last used more than 45 days ago is placed on BucketF1





4. The most recent snapshot is kept in NFS\_Tier1, all other snapshots are moved to BucketF1
5. File versioning enabled, keep for 1 day
6. File undelete enabled, keep for 1 day

### Key Design and Architecture Takeaways - One Site - Three-Tier NAS

- All data will be stored on external NFSv3 volumes, so the DSX nodes do not require data disks (only boot disks)
- A special condition is required to target only the latest snapshot, and all snapshots older than that
- Volume groups should be created for each storage tier, and the volumes for that tier should be added to the appropriate group

### Objective Design - All Sites

Objective	Condition	Meets Requirement
place-on-NFS-TierF0_VG	IS_LIVE AND LAST_USE_AGE < 14*HOURS	1
place-on-NFS-TierF1_VG	IS_LIVE AND LAST_USE_AGE >= 14*HOURS AND LAST_USE_AGE <= 45*HOURS	2
place-on-BucketF1_VG	IS_LIVE AND LAST_USE_AGE > 45*HOURS	3
place-on-NFS-TierF1_VG	IS_SNAP AND VERSION == 3	4
place-on-BucketF1_VG	IS_SNAP AND VERSION > 3	4
undelete-1-day	TRUE	5
versioning-1-day	TRUE	6

#### Note:

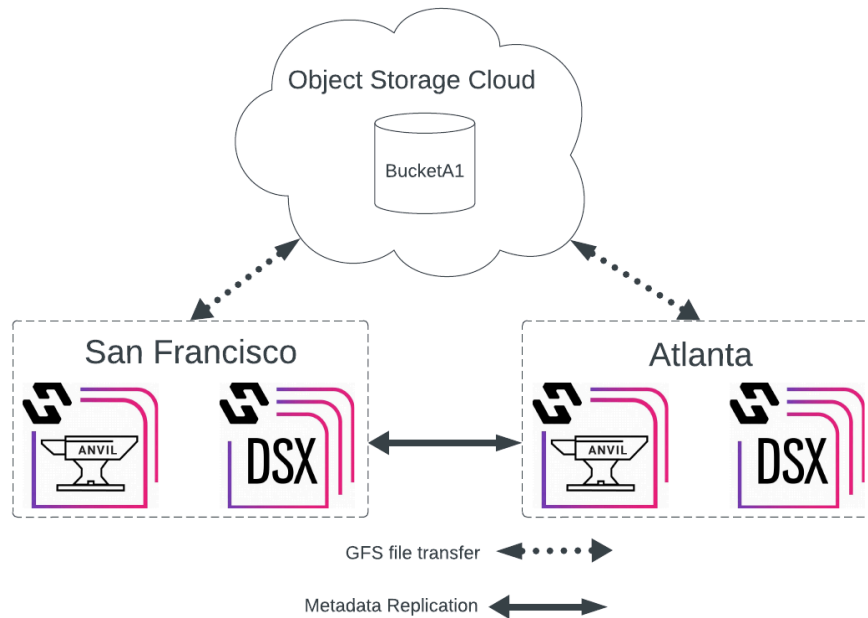
- IS\_LIVE targets files in the live tree, and excludes snapshots
- IS\_SNAP targets files in snapshots, and excludes the live file tree



- `VERSION==3` is the condition that matches only the most recent snapshot, `VERSION>3` matches all older versions
- To make the table easier to follow, the objectives used to tier live data were kept separate from the objectives used to place snapshots, however this is not required. You could have combined these conditions using OR statements such as this:
  - `(IS_LIVE AND LAST_USE_AGE >= 14*HOURS AND LAST_USE_AGE <= 48*HOURS) OR (IS_SNAP AND VERSION == 3)`
  - `(IS_LIVE AND LAST_USE_AGE > 48*HOURS) OR (IS_SNAP AND VERSION > 3)`

## Scenario #2: Two Sites - Global File Shares

In this scenario, the customer has two similar sites and wants to use a Hammerspace global namespace to extend all shares to both sites. The following diagram shows their proposed Hammerspace infrastructure:



Their cluster designs, share data placement requirements, and objective packages are the same on both sites.

### All Sites - Requirements

- San Francisco site
  - 2x DSX nodes, 1 RAIDed volume each
  - Connected to shared object storage: BucketA1
- Atlanta site
  - 2x DSX nodes, 1 RAIDed volume each
  - Connected to shared object storage: BucketA1
- Data placement and other requirements (global)
  1. Last 30 days used live data kept online
  2. All data copied to object storage after 5 minutes for DR purposes



3. Data beyond 30 days tiered to object storage
  4. Snapshots placed on object storage
  5. Shares are multi-protocol
  6. File versioning - kept for 1 day, exclude .work/.WORK files
  7. Users must not see files or folders that they don't have permissions to access
- Objectives
    - All sites/shares

### Key Design and Architecture Takeaways - Two Sites - Global File Shares

- The same objectives can be used for all sites.
- Each site has only 1 volume per DSX node, so a simple **keep-online** objective will be fine for keeping needed files online.
- Files will be copied to object storage 5 minutes after creation, creating the recommended second instance. This ensures that if a DSX node was unavailable, the file can always be retrieved from object storage.
- We don't have to tell an online file to be downtiered to offline object storage. Since we already have an instance of each file in object storage for DR purposes, the online instance will simply be dropped (removed) when the keep-online objective condition no longer applies.
- FNMATCH conditions are case sensitive. Verify that it is ok to match only **work** or **WORK**. If additional variations are required, such as **Work**, the supplied condition will need to be expanded.
- Snapshots are enabled on global file shares, so an objective should be added to ensure all global snapshot data is protected against site failure.
- All shares are multi-protocol. The **ACL-COMPATIBILITY-MODE-NON-POSIX** objective will be required.
- Access-based Enumeration is required to prevent users from seeing files and folders they do not have access to, so we will also need to add the **do-not-cache** objective.
- Volume groups should be created for each storage tier, and the volumes for that tier should be added to the appropriate group



## Objective Design - All Sites

Objective	Condition	Meets Requirement
keep-online	IS_LIVE AND LAST_USE_AGE ≤ 30*HOURS	1, 3
place-on-BUCKETA1_VG	(IS_LIVE AND LAST_USE_AGE ≥ 5*HOURS) OR IS_SNAP	2, 4
acl-compatibility-mode-non-posix	TRUE	5
versioning-1-day	SUM((CASE WHEN FNMATCH(("*.work"; "*.WORK")) THEN 1 ELSE 0) OVER (PARTITION BY NAME ORDER BY ROW)) = 0	6
access-based-enumeration	TRUE	7
do-not-cache	TRUE	7

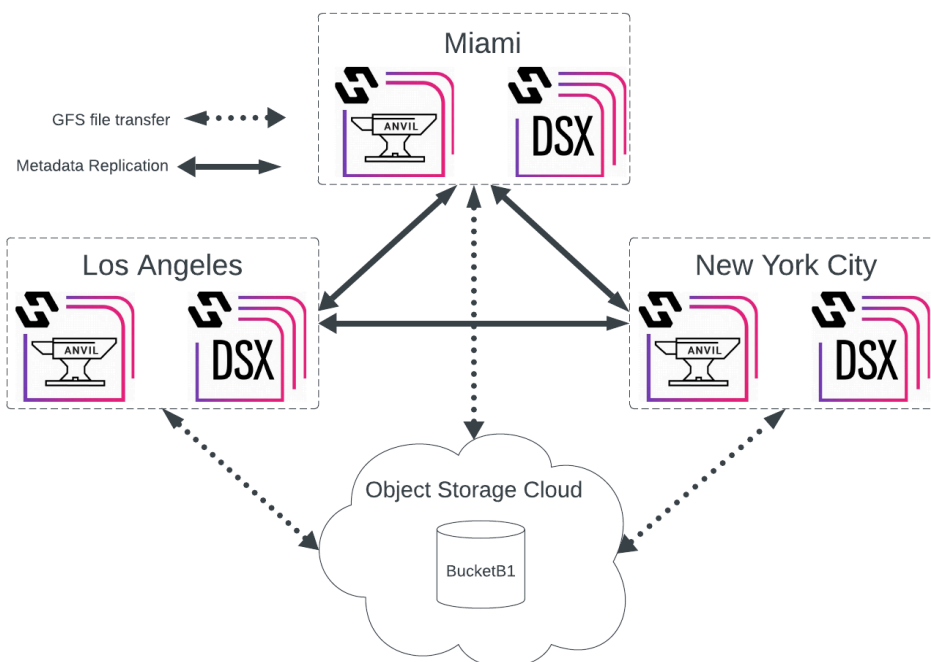
**Best Practice:** Files cannot be copied to object storage unless they have been closed for 3 minutes, which is one of the reasons why Hammerspace recommends waiting to copy files there. In this document, all examples wait 5 minutes before attempting to copy files to object storage.

### Scenario #3: Hub-spoke Multi-Site - Global File Shares

In this scenario, we have a hub-spoke Hammerspace global file share environment where a hub site keeps all data used in the last year online, while smaller spoke sites keep a much smaller amount online. Shared object storage volumes are used as an offline tier for files that have not been used for varying amounts of time.

**Note:** Hammerspace really doesn't have the concept of hub-spoke sites. We use that term in this document because it's assumed the spoke sites are smaller and will likely store less data online. Ultimately, it is your objectives that will determine what each site will do with regard to the data within global file shares.

The following diagram shows their proposed Hammerspace infrastructure:



Their cluster designs, share data placement requirements, and objective packages are outlined for each site individually.



## Primary Site - Los Angeles - Requirements

- Cluster configuration
  - 2x DSX nodes, 1 HDD and 1 SSD RAIDed volume each
  - Connected shared object storage: BucketB1
- Data placement requirements
  1. Newly created files are put on DSX SSD volume
  2. Files 4K and smaller must be placed on DSX SSD volume
  3. Last 1 year used live data put on DSX HDD volumes
  4. All files are written to object storage after 5 minutes of creation/editing
  5. Data last used more than 1 year ago stored in object storage only
  6. Snapshots placed on object storage
  7. File undelete - kept for 1 day - excluding "scratch" folders

## Key Design and Architecture Takeaways - Los Angeles

- There are two different types of online storage volumes (**hdd** and **ssd**), so you will need to create a volume group for each and place the correct volumes in them.
- Files will be copied to object storage 5 minutes after creation, creating the recommended second instance. This ensures that if a DSX node was unavailable, the file can always be retrieved from object storage.
- We don't have to tell an online file to be downtiered to offline object storage. Since we already have an instance in object storage for DR purposes, the online instance will simply be dropped (removed) when the keep-online objective condition no longer applies.
- Snapshots are enabled on global file shares, so an objective should be added to ensure all global snapshot data is protected against site failure.
- Volume groups should be created for each storage tier, and the volumes for that tier should be added to the appropriate group



## Objective Design - Los Angeles

Objective	Condition	Meets Requirement
place-on-SSD_VG	IS_BEING_CREATED OR (IS_LIVE AND SIZE<=4*KBYTES AND LAST_USE_AGE<=1*YEARS?TRUE)	1, 2, 5
place-on-HDD_VG	IS_LIVE AND LAST_USE_AGE <= 1*YEAR	3, 5
place-on-BUCKETA1_VG	(IS_LIVE AND LAST_USE_AGE >= 5*MINUTES) OR IS_SNAP	4, 6
undelete-1-day	!FNMATCH("*/scratch/*",PATH)	7

## Spoke site 1 - New York City - Requirements

- Cluster configuration
  - 2x DSX nodes, 1 volume each
  - Connected shared object storage: BucketB1
- Data placement requirements
  1. Files created or opened or saved by local users are kept online for 60 days
  2. Files/folders with "NYC" label are kept online, excluding folders named "temp"
  3. Data last used more than 60 days ago stored in object storage only
  4. All data copied to object storage after 1 hour for DR purposes
  5. Snapshots placed on object storage

## Key Design and Architecture Takeaways - New York City

- While it will only be used on this site, the **NYC** label should be created on all sites, and the internal ID of the label should be verified to match across all sites.
- NYC will only be copying files to object storage after 1 hour has passed, versus 5 minutes in LA. This is ok, but note that the Recovery Point Objective (RPO) for the sites will be different.





Consider changing it to 5 minutes so that all sites have a similar RPO.

- We don't have to tell an online file to be downtiered to offline object storage. Since we already have an instance in object storage for DR purposes, the online instance will simply be dropped (removed) when the keep-online objective condition no longer applies.
- Snapshots are enabled on global file shares, so an objective should be added to ensure all global snapshot data is protected against site failure.
- FNMATCH conditions are case sensitive, verify that it is ok to match only temp (lowercase), else add additional match conditions.

### Objective Design - New York City

Objective	Condition	Meets Requirement
keep-online	IS_LIVE AND HAS_LABEL(LABEL('NYC')) AND !FNMATCH("*/temp/*", PATH)	2
keep-online	IS_LIVE AND LAST_USE_AGE <= 60*HOURS AND (HAS_ONLINE_INSTANCE OR DATA_ORIGIN_LOCAL)	1, 3
place-on-BUCKETB1_VG	(IS_LIVE AND LAST_USE_AGE >= 1*HOURS) OR IS_SNAP	4, 5

### Spoke Site 2 - Miami - Requirements

- Cluster configuration
  - 2x DSX nodes, 1 RAIDed volume each
  - Connected shared object storage: BucketB1
- Data placement requirements
  1. Files created/used by local users are kept online for 1 month
  2. Files/folders with "MIA" label are kept online, excluding files with .tmp/.TMP extension



3. All data copied to object storage after 1 hour for DR purposes
4. Snapshots placed on object storage

### Key Design and Architecture Takeaways - Miami

- There is a requirement to keep files local once this site uses them, so we will need to use a combination of **HAS\_ONLINE\_INSTANCE** OR **DATA\_ORIGIN\_LOCAL**.
- While it will only be used on this site, the **MIA** label should be created on all sites, and the internal ID of the label should be verified to match across all sites.
- MIA will only be copying files to object storage after 1 hour has passed, versus 5 minutes in LA. This is ok, but note that the Recovery Point Objective (RPO) for the sites will be different. Consider changing it to 5 minutes so that all sites have a similar RPO.
- We don't have to tell an online file to be downtiered to offline object storage. Since we already have an instance in object storage for DR purposes, the online instance will simply be dropped (removed) when the keep-online objective condition no longer applies.
- Snapshots are enabled on global file shares, so an objective should be added to ensure all global snapshot data is protected against site failure.
- FNMATCH conditions are case sensitive, verify that it is ok to match only **TMP** or **tmp**, else add additional match conditions.
- Volume groups should be created for each storage tier, and the volumes for that tier should be added to the appropriate group

### Objective Design - Miami

Objective	Condition	Meets Requirement
keep-online	IS_LIVE AND HAS_LABEL(LABEL('MIA')) AND SUM({    #A=FNMATCH(({ "*.tmp" ; "*.TMP" }))[ROW], NAME) }[2])=	2



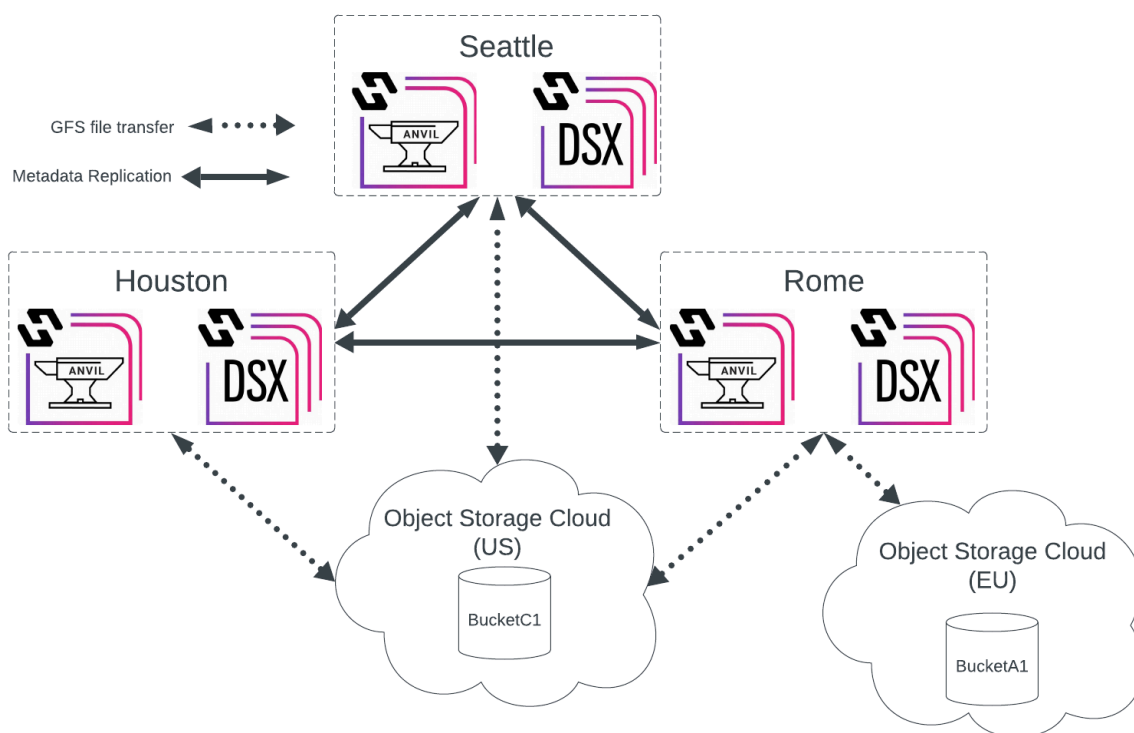
	=0	
<b>keep-online</b>	IS LIVE AND LAST_USE_AGE <= 1*MONTHS AND (HAS_ONLINE_INSTANCE OR DATA_ORIGIN_LOCAL)	1
<b>place-on-BUCKETB1_VG</b>	(IS_LIVE AND LAST_USE_AGE >= 1*HOURS) OR IS_SNAP	3, 4



## Scenario #4: Mesh Multi-Site - Global File Shares

In this design scenario, we have a multi-site Hammerspace global file share environment where each site has unique requirements for data availability, and each cluster has multiple classes of disks to be used for online storage. Shared object storage volumes are used as an offline tier for files that have not been used for varying amounts of time.

The following diagram shows their proposed Hammerspace infrastructure:



Their cluster designs, share data placement requirements, and objective packages are outlined for each site individually.

### Mesh Site 1 - Seattle - Requirements

- Cluster configuration
  - 2x DSX nodes, 1 HDD and 1 SSD RAIDed volume each



- Connected shared object storage: US-BucketC1
- Data placement requirements
  1. Last 6 months used live files with \*.mov and \*.png extension should be placed on DSX HDD volume
  2. Last 6 months used live files (excluding \*.mov and \*.png) should be placed on DSX SSD volumes
  3. All files are written to object storage after 5 minutes of creation/editing
  4. Data beyond 6 months stored in object storage only
  5. Snapshots placed on object storage

### **Key Design and Architecture Takeaways - Seattle**

- There are two different types of online storage volumes (**hdd** and **ssd**), so you will need to create a volume group for each and place the correct volumes in them.
- Since certain file types will only be placed on a specific volume type, we need to ensure those same files will not be stored on the other volume type.
- FNMATCH conditions are case sensitive, verify that it is ok to match only png and mov (lowercase), else add additional match conditions.
- Files will be copied to object storage 5 minutes after creation, creating the recommended second instance. This ensures that if a DSX node is unavailable, the file can always be retrieved from object storage.
- We don't have to tell an online file to be downtiered to offline object storage. Since we already have an instance in object storage for DR purposes, the online instance will simply be dropped (removed) when the keep-online objective condition no longer applies.
- Snapshots are enabled on global file shares, so an objective should be added to ensure all global snapshot data is protected against site failure.



## Objective Design - Seattle

Objective	Condition	Meets Requirement
place-on-HDD_VG	IS_LIVE AND LAST_USE_AGE <= 6*MONTHS AND, SUM({  #A=FNMATCH(({"*.mov"; "*.png"})[ROW],NAME))[2])=1	1, 4
place-on-SSD_VG	IS_LIVE AND LAST_USE_AGE <= 6*MONTHS AND SUM({  #A=FNMATCH(({"*.mov"; "*.png"})[ROW],NAME))[2])=0	2, 4
place-on-US-BUCKET1_VG	(IS_LIVE AND LAST_USE_AGE >= 5*MINUTES) OR IS_SNAP	3, 5

## Mesh Site 2 - Houston - Requirements

- Cluster configuration
  - 3x DSX nodes, 1 HDD and 1 SSD volume each
    - DSX volumes are on standalone, non-RAIDED drives
  - Connected shared object storage: US-BucketC1
- Data placement requirements
  1. Last 4 months used live data put on DSX HDD volumes
  2. Newly created files should be placed on DSX SSD volumes
  3. Data beyond 4 months stored in object storage only
  4. Snapshots placed on object storage

## Key Design and Architecture Takeaways - Houston

- There are two different types of online storage volumes (**hdd** and **ssd**), so you will need to create a volume group for each and place the correct volumes in them.
- Unlike Seattle, there is no listed requirement to write all files to object storage. *Per Hammerspace best practices*, and to support continuity in the event of a site failure, this objective should be added to every site that hosts global file shares.
- Given that we are writing all data to object storage for DR purposes, there is less urgency to protect against the failure of



the non-RAIDED hdd and ssd volumes. If for some reason writing all data to the object storage is not practical, we should consider adding an **availability-3-nines** objective to each share so that files will be written to two different DSX nodes (each node will be considered a failure domain, rather than each volume).

- We don't have to tell an online file to be downtiered to offline object storage. Since we have decided to place an instance in object storage for DR purposes, the online instance will simply be dropped (removed) when the **keep-online** objective condition no longer applies.
- Snapshots are enabled on global file shares. An objective should be added to ensure all global snapshot data is protected against site failure.
- Volume groups should be created for each storage tier, and the volumes for that tier should be added to the appropriate group

### Objective Design - Houston

Objective	Condition	Meets Requirement
place-on-HDD_VG	IS_LIVE AND LAST_USE_AGE <= 4*MONTHS	1, 3
place-on-SSD_VG	IS_BEING_CREATED	2
place-on-US-BUCKET1_VG	(IS_LIVE AND LAST_USE_AGE >= 5*MINUTES) OR IS_SNAP	4, DR best practice

### Mesh Site 3 - Rome - Requirements

- Cluster configuration
  - 2x DSX nodes, 1 HDD and 1 SSD RAIDed volume each
  - Connected shared object storage: US-BucketC1
  - Connected non-shared object storage: EU-BucketA1
- Data placement requirements
  1. Last 1 month's used live data put on DSX HDD volumes
  2. Newly created files should be placed on DSX SSD volumes

3. All files are written to the appropriate object storage after 15 minutes of creation/editing
4. Data on the standalone Rome\_HR share should have extra precautions taken to ensure that it isn't written to US-BucketC1, instead bucket EU-BucketA1 should be used.
5. Data beyond 1 months stored in object storage only
6. Snapshots placed on object storage

### Key Design and Architecture Takeaways - Rome

- There are two different types of online storage volumes (**hdd** and **ssd**), so you will need to create a volume group for each and place the correct volumes in them.
- For all shares but the Rome\_HR share, we don't have to tell an online file to be downtiered to offline object storage. Since we have decided to place an instance in object storage for DR purposes, the online instance will simply be dropped (removed) when the **keep-online** objective condition no longer applies.
- For the Rome\_HR share, we need to take every precaution available to ensure that files will not be written to US-based object storage volumes.
- Snapshots are enabled on global file shares, so even though it is not mentioned an objective should be added to ensure all global snapshot data is protected against site failure.
- While not an objective, note that the Rome\_HR share is not a global file share, so snapshots for this share will need to be enabled on the Rome site directly.
- Volume groups should be created for each storage tier, and the volumes for that tier should be added to the appropriate group

### Objective Design - Rome

Objective	Condition	Meets Requirement
place-on-SSD_VG	IS_BEING_CREATED	2
place-on-HDD_VG	IS_LIVE AND LAST_USE_AGE <= 1*MONTHS	1, 5





<b>place-on-US-BUCKETC1_VG</b>	(IS_LIVE AND LAST_USE_AGE >= 15*MINUTES) OR IS_SNAP	3, 6
<b>confine-to-SSD_VG**</b>	IS_BEING_CREATED	2, 4
<b>confine-to-HDD_VG**</b>	IS_LIVE AND LAST_USE_AGE <= 1*MONTHS	1, 4, 5
<b>confine-to-EU-BUCKETA1_VG**</b>	(IS_LIVE AND LAST_USE_AGE >= 15*MINUTES) OR IS_SNAP	3, 4, 6
<b>exclude-from-US-BUCKETC1_VG**</b>	N/A	4

**\*\* Objectives apply to Rome\_HR share only**



# Appendix

## Hammerspace Toolkit

The Hammerspace toolkit is available for Windows, Mac, and Linux, and can be downloaded from the [Hammerspace GitHub repository](#).

## Hammerspace Metadata Plugin for Windows File Explorer

The Hammerspace Metadata Plugin provides Microsoft Windows users with a simpler method for interacting with file and folder tags, labels, and keywords.

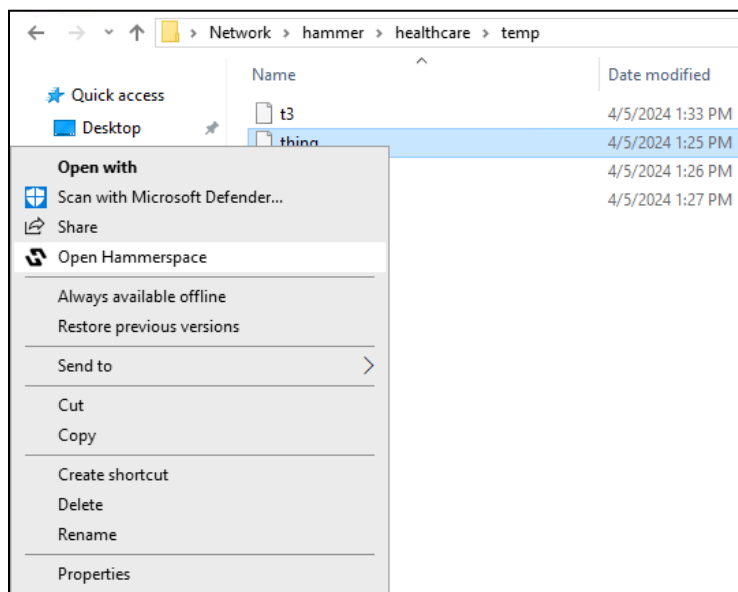
### Metadata Plugin - Download and Installation

You can download the Hammerspace Metadata Plugin for Windows File Explorer from Flexera (Flexnet), the host for Hammerspace cluster software. Look for the following:

<input type="checkbox"/> + File Description	File Size	File Added	File Name
<input type="checkbox"/> + Metadata Plugin for Windows	131.58MB	May 06, 2024	<a href="#">HammerspacePluginInstaller-1.0.0.62.zip</a>

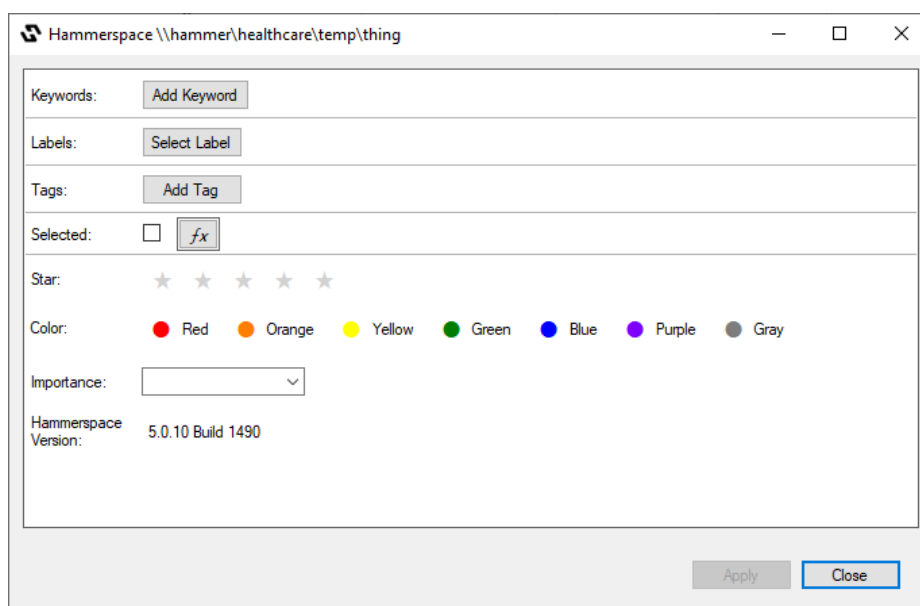
Unzip the downloaded package, run (double click) the [HammerspaceClassicInstaller.msi](#) file, and follow the prompts.

Once the installer completes, open Windows File Explorer and navigate to a Hammerspace share. Then, right-click a file and select **Open Hammerspace**.





Here you'll find the main plugin screen, and note that you can use the plugin to apply keywords, labels, and tags. The plugin also allows you to set custom metadata values for **Star**, **Color**, and **Importance**. These are some of the other additional metadata values you can set and use as conditions.



## File Metadata Example (Full)

The following is the raw output of a file's Hammerspace metadata that contains values for many of the conditions discussed in this document.

Obtain this information for files located on a Hammerspace share using the `hs eval -e this` command as shown in the following example:

```
$ hs eval -e this telemetry.txtt
ITEM_TABLE{
|NOW = LOCAL_TIME('2024-09-24 12:53:52'),
|INODE = INODE_NUMBER('961'),
|IS_BEING_CREATED = FALSE,
|NAME = "file-95",
|TYPE = ITEM_TYPE('FILE'),
|MODE = 0664,
|NLINK = 1,
|OWNER = USER('1000'),
|OWNER_GROUP = GROUP('1000'),
|SIZE = 4.096 KBYTES,
|SPACE_USED = 4.096 KBYTES,
```



```
|CHANGE_TIME = LOCAL_TIME('2024-09-05 17:15:31'),
|ACCESS_TIME = LOCAL_TIME('2024-09-05 16:58:05'),
|MODIFY_TIME = LOCAL_TIME('2024-09-05 17:15:31'),
|CREATE_TIME = LOCAL_TIME('2024-09-05 16:58:05'),
|LAST_USE_TIME = LOCAL_TIME('2024-09-05 17:15:31'),
|SYMLINK = #EMPTY,
|RDEV = RDEV_TABLE{
    |MAJOR = 0,
    |MINOR = 0},
|ACTIVITY = 0 OPERATIONS / SECOND,
|ERRORS = ERROR_NONE,
|ERRORS_REFRESHED = ERROR_NONE,
|IS_MODIFIED_AFTER_SNAP = FALSE,
|VERSION = 1,
|VERSION_OLDEST = 2,
|VERSION_NEWEST = 1,
|ALIGNMENT_DETAILS = ALIGNMENT_DETAILS_TABLE{
    |CHANGE_TIME = LOCAL_TIME('1969-12-31 19:00:00'),
    |LAST_CHECKED_TIME = LOCAL_TIME('1969-12-31 19:00:00'),
    |OVERALL = ALIGNMENT('ALIGNED'),
    |PLACE_ON = ALIGNMENT('ALIGNED'),
    |EXCLUDE_FROM = ALIGNMENT('ALIGNED'),
    |CONFINED_TO = ALIGNMENT('ALIGNED'),
    |DO_NOT_MOVE = ALIGNMENT('UNKNOWN'),
    |DURABILITY = ALIGNMENT('ALIGNED'),
    |ONLINE_DELAY = ALIGNMENT('ALIGNED'),
    |AVAILABILITY = ALIGNMENT('ALIGNED'),
    |READ_BANDWIDTH = ALIGNMENT('ALIGNED'),
    |READ_IOPS = ALIGNMENT('ALIGNED'),
    |READ_LATENCY = ALIGNMENT('ALIGNED'),
    |WRITE_BANDWIDTH = ALIGNMENT('ALIGNED'),
    |WRITE_IOPS = ALIGNMENT('ALIGNED'),
    |WRITE_LATENCY = ALIGNMENT('ALIGNED'),
    |STATE = ALIGNMENT_STATE('WAITING_FOR_TARGET_WITH_SPACE'),
    |#R = 2},
|ALIGNMENT_ASPECTS = {
    ASPECT('OVERALL'), ALIGNMENT('ALIGNED');
    ASPECT('PLACE ON'), ALIGNMENT('ALIGNED');
    ASPECT('EXCLUDE FROM'), ALIGNMENT('ALIGNED');
    ASPECT('CONFINED TO'), ALIGNMENT('ALIGNED');
    ASPECT('DURABILITY'), ALIGNMENT('ALIGNED');
    ASPECT('ONLINE DELAY'), ALIGNMENT('ALIGNED');
    ASPECT('AVAILABILITY'), ALIGNMENT('ALIGNED');
    ASPECT('READ BANDWIDTH'), ALIGNMENT('ALIGNED');
    ASPECT('READ IOPS'), ALIGNMENT('ALIGNED');
    ASPECT('READ LATENCY'), ALIGNMENT('ALIGNED');
    ASPECT('WRITE BANDWIDTH'), ALIGNMENT('ALIGNED');
    ASPECT('WRITE IOPS'), ALIGNMENT('ALIGNED');
    ASPECT('WRITE LATENCY'), ALIGNMENT('ALIGNED')},
```



```
|INSTANCES = INSTANCES_TABLE{
    |CREATE_TIME = LOCAL_TIME('2024-03-20 08:37:56'),
    |START_TIME = LOCAL_TIME('2024-03-20 08:37:56'),
    |END_TIME = LOCAL_TIME('2024-09-24 12:53:52'),
    |VOLUME = STORAGE_VOLUME('AZ2:VOL2-2'),
    |START_OFFSET = 0 BYTES,
    |END_OFFSET = 4.096 KBYTES,
    |SPACE_USED = 4.096 KBYTES,
    |ID = 9223372036854817e3,
    |NSHARED = 1,
    |READABLE = 1,
    |WRITEABLE = 1,
    |UID = 10,
    |GID = 10,
    |MTIME = LOCAL_TIME('2024-09-05 17:18:19'),
    |ATIME = LOCAL_TIME('2024-03-20 08:37:56'),
    |ERROR_COUNT = 0,
    |PATH = "/PrimaryData/Instances/1/0/7624/v40-f1-8000000000009dc6",
    |SIGNATURE =
    "000293DCC16BDF6A6C58A1ECA946B140D813A4EF1F843F0FCFE6ECA0D1BBDB8D9CE5D000000000001000",
    |SHARE_PERCENTAGE = 100%,
    |LENGTH = 4.096 KBYTES,
    |SPARSENESS = 100%,
    |ONLINE_DELAY = 0 SECONDS,
    |IS_ONLINE = TRUE,
    |IS_CURRENT = TRUE,
    |IS_BEING_CREATED = FALSE,
    |IS_BEING_MOVED = FALSE,
    |PERCENTAGE_COMPLETE = 100%},
|HASH =
"000293DCC16BDF6A6C58A1ECA946B140D813A4EF1F843F0FCFE6ECA0D1BBDB8D9CE5D000000000001000",
|ASSIMILATION_STATUS = #EMPTY,
|SWEEP_DETAILS = SWEEP_DETAILS_TABLE{
    |NUMBER = 166821,
    |START_TIME = LOCAL_TIME('2024-09-24 12:53:10'),
    |END_TIME = LOCAL_TIME('2024-09-24 12:53:48'),
    |COUNT = 1.002 KFILES,
    |ERROR_COUNTS = ERROR_STATS_TABLE{
        |ERROR = ERROR_NUMBER('1'),
        |COUNT = 6},
    |TOTAL_COUNT = 1.002 KFILES,
    |RATE = 26.243 FILES / SECOND,
    |PROGRESS = 100%,
    |ELAPSED_TIME = 38.182 SECONDS,
    |REMAINING_TIME = 0 SECONDS,
    |COMPLETION_TIME = LOCAL_TIME('2024-09-24 12:53:48');

    |NUMBER = 166822,
    |START_TIME = LOCAL_TIME('2024-09-24 12:53:48'),
```



```
|END_TIME = LOCAL_TIME('2024-09-24 12:53:52'),
|COUNT = 79 FILES,
|ERROR_COUNTS = ERROR_STATS_TABLE{
    |ERROR = ERROR_NUMBER('1'),
    |COUNT = 1},
|TOTAL_COUNT = 1.002 KFILES,
|RATE = 21.02 FILES / SECOND,
|PROGRESS = 7.8842%,
|ELAPSED_TIME = 3.75837 SECONDS,
|REMAINING_TIME = 43.911 SECONDS,
|COMPLETION_TIME = LOCAL_TIME('2024-09-24 12:54:36')),
|BOOT_EPOCH = 4,
|REPLICATION_DETAILS = REPLICATION_DETAILS_TABLE{
    |SITE = SITE('NYC-HS-A'),
    |INTERVAL = 5 SECONDS,
    |SEND_TIME = LOCAL_TIME('2024-09-24 12:53:44'),
    |RCV_TIME = LOCAL_TIME('2024-09-24 12:53:37'),
    |SNAPS = REPLICATION_DETAILS_SUB_TABLE{
        |SENT_REQUESTS = 76557,
        |SENT_BYTES = 0 BYTES,
        |SUCCEEDED_REQUESTS = 76556,
        |SUCCEEDED_BYTES = 0 BYTES,
        |ERRORED_REQUESTS = 1,
        |ERRORED_BYTES = 0 BYTES,
        |COMPLETED_REQUESTS = 76557,
        |COMPLETED_BYTES = 0 BYTES,
        |PENDING_REQUESTS = 0,
        |PENDING_BYTES = 0 BYTES},
    |DIRENTS = REPLICATION_DETAILS_SUB_TABLE{
        |SENT_REQUESTS = 0,
        |SENT_BYTES = 0 BYTES,
        |SUCCEEDED_REQUESTS = 0,
        |SUCCEEDED_BYTES = 0 BYTES,
        |ERRORED_REQUESTS = 0,
        |ERRORED_BYTES = 0 BYTES,
        |COMPLETED_REQUESTS = 0,
        |COMPLETED_BYTES = 0 BYTES,
        |PENDING_REQUESTS = 0,
        |PENDING_BYTES = 0 BYTES},
    |INODES = REPLICATION_DETAILS_SUB_TABLE{
        |SENT_REQUESTS = 0,
        |SENT_BYTES = 0 BYTES,
        |SUCCEEDED_REQUESTS = 0,
        |SUCCEEDED_BYTES = 0 BYTES,
        |ERRORED_REQUESTS = 0,
        |ERRORED_BYTES = 0 BYTES,
        |COMPLETED_REQUESTS = 0,
        |COMPLETED_BYTES = 0 BYTES,
        |PENDING_REQUESTS = 0,
```



```
|PENDING_BYTES = 0 BYTES},
|TOTAL = REPLICATION_DETAILS_SUB_TABLE{
|SENT_REQUESTS = 76557,
|SENT_BYTES = 0 BYTES,
|SUCCEEDED_REQUESTS = 76556,
|SUCCEEDED_BYTES = 0 BYTES,
|ERRORED_REQUESTS = 1,
|ERRORED_BYTES = 0 BYTES,
|COMPLETED_REQUESTS = 76557,
|COMPLETED_BYTES = 0 BYTES,
|PENDING_REQUESTS = 0,
|PENDING_BYTES = 0 BYTES},
|SEND_AGE = 8.0824 SECONDS,
|RECV_AGE = 14.333 SECONDS},
|ASSIMILATION_DETAILS = ASSIMILATION_DETAILS_TABLE{},
|HAS_PERMANENT_DATA_LOSS = FALSE,
|HAS_PERMANENT_DATA_LOSS_REFRESHED = FALSE,
|HAS_ERROR = FALSE,
|ASSIMILATION_ERROR = ERROR_NONE,
|REPLICATION_ERROR = ERROR_NONE,
|VERSION_DETAILS = VERSION_DETAILS_TABLE{
|TIME = LOCAL_TIME('2024-09-24 12:53:52'),
|PATH = "./file-95";

|TIME = LOCAL_TIME('2024-09-24 12:53:52'),
|PATH = "./.snapshot/current/file-95"},
|VERSIONS_TOTAL = 2,
|VERSION_TIME = LOCAL_TIME('2024-09-24 12:53:52'),
|LAST_EXISTED_TIME = LOCAL_TIME('2024-09-24 12:53:52'),
|SPARSENESS = 100%,
|OVERALL_ALIGNMENT = ALIGNMENT('ALIGNED'),
|OVERALL_ALIGNMENT_STATE = ALIGNMENT_STATE('WAITING_FOR_TARGET_WITH_SPACE'),
|ALIGNMENT_CHANGE_TIME = LOCAL_TIME('1969-12-31 19:00:00'),
|DATA_ORIGIN_LOCAL = FALSE,
|DATA_ORIGIN_REMOTE = TRUE,
|DATA_ORIGIN_SITE = PARTICIPANTS_TABLE{
|ID = 1,
|SHARE_ID = 4,
|ADDRESS = "192.200.10.175",
|PORT = 9097,
|INTERVAL = 5 SECONDS,
|SITE_PARTICIPANT_ID = 2,
|ADMIN_STATUS = ADMINISTRATIVE_STATUS('ENABLED'),
|OPER_STATUS = OPERATIONAL_STATUS('UP'),
|SITE_NAME = "NYC-HS-A"},
|IS_OPEN = FALSE,
|HAS_ONLINE_INSTANCE = TRUE,
|IS_ONLINE = TRUE,
|IS_BEING_MOVED = FALSE,
```



```
|IS_OFFLINE = FALSE,
|IS_VOLATILE = FALSE,
|IS_DURABLE = TRUE,
|IS_UNAVAILABLE = FALSE,
|IS_AVAILABLE = TRUE,
|IS_ACTIVE = FALSE,
|IS_INACTIVE = TRUE,
|IS_LIVE = TRUE,
|IS_SNAP = FALSE,
|IS_UNDELETE = 0,
|IS_DELETED = FALSE,
|IS_FILE = TRUE,
|IS_SYMLINK = FALSE,
|IS_DIRECTORY = FALSE,
|HAS_DIRECTORY_ENTRIES = FALSE,
|IS_DEVICE = FALSE,
|IS_THREAT = FALSE,
|IS_NON_THREAT = FALSE,
|IS_THREAT_UNKNOWN = TRUE,
|MAY_BE_THREAT = TRUE,
|CREATE_AGE = (18 DAYS+19:55:51),
|CHANGE_AGE = (18 DAYS+19:38:21),
|ACCESS_AGE = (18 DAYS+19:55:51),
|MODIFY_AGE = (18 DAYS+19:38:21),
|LAST_USE_AGE = (18 DAYS+19:38:21),
|ACTUAL_CREATE_AGE = (18 DAYS+19:55:51),
|ACTUAL_ACCESS_AGE = (18 DAYS+19:55:51),
|ACTUAL_MODIFY_AGE = (18 DAYS+19:38:21),
|ACTUAL_LAST_USE_AGE = (18 DAYS+19:38:21),
|VERSION_AGE = 0 SECONDS,
|DELETED_AGE = 0 SECONDS,
|ALIGNMENT_CHANGE_AGE = (54 YEARS+280 DAYS),
|ONLINE_DELAY = 0 SECONDS,
|ORIGIN_SITE = SITE('LA-HS-B'),
|ATTRIBUTES = ITEM_ATTRIBUTES_TYPED_TABLE{
    |VIRUS_SCAN = VIRUS_SCAN_STATE('UNSCANNED'),
    |MIME = MIME_TYPE_TABLE{
        |STRING = "/"},
    |DO_NOT_MOVE = FALSE,
    |CONSIDER_VOLATILE = FALSE,
    |RECENTLY_USED_DURATION = 00:05:00,
    |SELECTED = FALSE,
    |SELECTED2 = FALSE,
    |PROMOTE = FALSE,
    |DEMOTE = FALSE,
    |COLORS = COLORS_TABLE{},
    |CSI_DETAILS = CSI_DETAILS_TABLE{}},
|OBJECT = OBJECT_NUMBER('1125899906842719'),
|DEPTH = 2,    |PATH = "./file-95",
```





```
|DIRECTORY_STATS = DIRECTORY_STATS_TABLE{
    |FILE_COUNT = 1 FILE,
    |SPACE_USED = 1.234 MBYTES,
    |ACTIVITY = 0.011574 OPERATIONS / SECOND,
    |LAST_USE_TIME = LOCAL_TIME('1901-05-15 20:00:00'),
    |LAST_USE_AGE = (123 YEARS+162 DAYS),
    |ACTIVITY_LEVEL = IOPS_LEVEL('UNDER 10 IOPS'),
    |HEAT = TEMPERATURE(|ACTIVITY=0.011574 OPERATIONS / SECOND),
    |HEAT_LEVEL = TEMPERATURE_LEVEL('UNDER 10 IOPS')},
|DPATH = "./file-95",
|IS_ROOT = FALSE,
|ALL_LABELS = LABELS_TABLE{},
|ACTUAL_CREATE_TIME = LOCAL_TIME('2024-09-05 16:58:05'),
|ACTUAL_MODIFY_TIME = LOCAL_TIME('2024-09-05 17:15:31'),
|ACTUAL_ACCESS_TIME = LOCAL_TIME('2024-09-05 16:58:05'),
|ACTUAL_LAST_USE_TIME = LOCAL_TIME('2024-09-05 17:15:31'),
|READONLY = FALSE,
|HIDDEN = FALSE,
|SYSTEM = FALSE,
|OFFLINE = FALSE,
|DIRECTORY = FALSE,
|ARCHIVE = FALSE,
|NORMAL = TRUE,
|REPARSEPOINT = FALSE,
|NOSCRUBDATA = FALSE,
|IS_NEWBORN = FALSE,
|IS_RECENTLY_USED = FALSE,
|IS_UNUSED_SINCE_CREATION = FALSE,
|ACTIVITY_LEVEL = IOPS_LEVEL('UNDER 10 IOPS'),
|HEAT = TEMPERATURE(|LAST_USE_AGE=(18 DAYS+19:38:21)),
|HEAT_LEVEL = TEMPERATURE_LEVEL('2 TO 3 WEEKS OLD'),
|SPACE_USED_LEVEL = SIZE_LEVEL('4 TO 32 KBYTES'),
|ACCESS_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|MODIFY_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|CHANGE_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|CREATE_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|LAST_USE_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|ACTUAL_ACCESS_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|ACTUAL_MODIFY_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|ACTUAL_CREATE_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|ACTUAL_LAST_USE_AGE_LEVEL = TIMESPAN_LEVEL('2 TO 3 WEEKS OLD'),
|PARENT_SHARE = SHARE('test_kent'),
|PARTICIPANTS = PARTICIPANTS_TABLE{
    |ID = 0,
    |SHARE_ID = 42,
    |ADDRESS = "192.200.10.160",
    |PORT = 9097,
    |INTERVAL = 0 SECONDS,
    |SITE_PARTICIPANT_ID = 0,
```



```
|ADMIN_STATUS = ADMINISTRATIVE_STATUS('ENABLED'),  
|OPER_STATUS = OPERATIONAL_STATUS('UP'),  
|SITE_NAME = "LA-HS-A";  
  
|ID = 1,  
|SHARE_ID = 4,  
|ADDRESS = "192.200.10.175",  
|PORT = 9097,  
|INTERVAL = 5 SECONDS,  
|SITE_PARTICIPANT_ID = 2,  
|ADMIN_STATUS = ADMINISTRATIVE_STATUS('ENABLED'),  
|OPER_STATUS = OPERATIONAL_STATUS('UP'),  
|SITE_NAME = "NYC-HS-A"},  
|DIRECTORY_ENTRIES = ITEM_TABLE{}}
```