WORM Solution Guide

HAMMERSPACE

Table of Contents

About This Documentation

Data immutability, commonly referred to as WORM, is the ability to prevent selected data from being changed or deleted. Often this is simply a useful option to have, but in some industries, and for some data, it's a legal requirement. Technology has advanced considerably since the days of punch cards and paper tape — the original "Write Once Read Many" media — and there are many options for storing data immutably, and many reasons an organization may wish to do so. After briefly describing today's WORM technology and use cases, this document reviews the WORM features in Hammerspace software, shows in detail how to configure them, and outlines best practices for using them.

For support resources, see the Hammerspace Support Hub and the Hammerspace Licensing and Delivery Portal.

Do not install additional or third-party software, agents, or packages directly on Hammerspace Anvil (metadata) or DSX (data) nodes. These nodes run a managed appliance software stack; anything installed outside that stack is unsupported and is removed during a software upgrade. Such software can also interfere with node operation. If you need monitoring or integration with an external tool, contact Hammerspace Support for a supported approach.

Chapter 1. Data Immutability — Definition and Use Cases

This section defines data immutability, explains the difference between Compliance and Enterprise WORM, and outlines the use cases that drive the need for immutable storage.

1.1. Defining WORM

Data immutability is the general term for preserving digital data in a form where it is not possible to change it or delete it once written. This may seem like a black and white definition, but because there are varying technology solutions, business requirements, and use cases, it contains a considerable amount of gray.

Immutability was originally based on media such as the magneto-optical disk, where the physics of the recording system enforced read-only behavior. This media could physically only be written a single time, earning it the moniker "write once, read many" or WORM. Tape systems such as LTO-WORM rely on firmware (embedded software) in the tape drive and special media to enforce immutability. Many other systems rely entirely on software to provide immutability, even though the underlying media on which the data is stored (HDDs and/or SSDs) is capable of being erased and rewritten. Hammerspace is such a software-based system.

Figure 1. WORM technology over time and how each generation is enforced

Some define "data immutability" as the feature, and "WORM" as one of the technologies that may be used to achieve it. Most, however, just call it all WORM, which is much easier to say, type, and fit into a GUI or CLI. Because WORM is how the feature is referred to in the Hammerspace UI, the remainder of this document will use that term to refer to data immutability.

1.2. Compliance vs. Enterprise WORM

Hardware-based WORM technology is more secure than software-based approaches because there is no circumstance under which it can be bypassed or "turned off" by an administrator. But with security comes inconvenience. What if files only need to be immutable for a specific period of time? Or they need to be deleted when they are older than 7 years? Or dealing with unusual media like optical disks or tape is too disruptive or expensive? In these situations, software-based WORM is a better fit.

Specific business requirements may limit the acceptable options. Some industries are highly regulated and require certain types of data to be stored immutably using systems that meet specific regulatory requirements. Securities brokers and dealers, for example, must store records on a system that is certified to meet the requirements of SEC Rule 17a-4(f). This is only one example; an alphabet soup of similar regulations exists across many industries and global regions.

When WORM isn’t mandated by regulations, it is still a useful tool for safeguarding assets from modification or deletion. Ad agencies save final versions of video and print ads. Semiconductor companies archive finished chip designs. In general, any industry that produces digital products and needs to protect completed designs may want to use WORM technology to do so. WORM is also an excellent defense against threats like ransomware when it is used to protect static data assets or backup images.

Two categories of WORM technology exist to meet these different requirements, commonly referred to as "Compliance" WORM, and "Enterprise" WORM. Compliance WORM solutions are certified to comply with one or more industry regulations requiring immutability and are thus suitable for storing regulated data in those specific industries. All other solutions fall into the Enterprise WORM category. It is possible for a single product to have multiple modes, where it may be operated in Compliance mode, Enterprise mode, or both.

Chapter 2. Hammerspace WORM Capabilities

Hammerspace provides Enterprise WORM functionality within its own file system, though it can also integrate with third-party Compliance WORM systems. It provides many flexible options to enable organizations to protect valuable digital data against unauthorized changes or deletion.

WORM is implemented in Hammerspace using objectives, which are flexible policies that can define where files live, when they move, and many other aspects of their behavior. When applied to a file, directory, or entire share, WORM objectives prevent subject files from being changed or deleted. Because of the flexibility of objectives and Hammerspace in general, there are many different ways to configure WORM to meet different business requirements.

Hammerspace WORM has the following key characteristics:

- **Multi-protocol:** WORM objectives apply across NFS, SMB, and S3 client protocols, and may even be set via S3 using standard S3 API calls.
- **Storage independent:** Shares in Hammerspace are abstractions that by default are not tied to any specific underlying storage system. This is very different from traditional WORM that is implemented in a separate storage system or partition. In the traditional case, files must always live on the siloed WORM storage, which may be difficult to use or slow. With Hammerspace, files may be dynamically and transparently orchestrated between storage systems based on objectives, while remaining in the same logical location and retaining WORM protection.
- **Granular:** WORM objectives may be applied at the file, directory, or share level, or a combination of these. When applied at the share or directory level, objectives are inherited. All files placed in a directory having the `worm-operation` objective will be affected, as will any subdirectories created there. By using the ALWAYS and TRUE objective applicability options judiciously, it is possible to create layered structures with different WORM characteristics. One example is creating a share with a default WORM retention period, and having some directories within the share configured with a longer retention period, for example for a legal hold use case.
- **Compatible with WORM storage:** Selected files may be protected on third-party WORM storage. Examples include object storage or cloud storage that supports WORM (aka "Object Lock") and WORM tape behind a supported S3-to-tape gateway. Hammerspace brings previously siloed WORM repositories into the global namespace.

Some of the options that may be used with WORM objectives include:

- **Expiration date:** A WORM share may have an expiration date, after which files again become writable and deletable. This date may be extended further into the future but not reduced.
- **Effective time:** WORM objectives normally take effect when a file is created. If desired, there may be a delay before WORM takes effect. For example, only after a file has been closed for 15 minutes, 15 days, etc. Similarly, one may define a duration after which the WORM objective expires. For example, 1 year from the date the file was created.
- **Snapshots:** Hammerspace snapshots are read only, but may be removed by an admin or a schedule. To provide protection against deletion, objectives may be used to copy snapshots to immutable storage, such

as an AWS bucket with Object Lock enabled.

- **Clones:** Hammerspace file and directory clones may be protected with WORM objectives.
- **Privileged delete:** As with most other Enterprise WORM solutions, Hammerspace allows administrators to delete WORM-protected files. This is done via the `privileged-delete` command, which may be executed from the administrative CLI of the Hammerspace system. This command can only delete WORM-protected files, not regular files. *It should be used with extreme care, keeping in mind the organization's requirements and policies around the affected data.*

Chapter 3. Immutability Objectives in Hammerspace

There are several ways to make data immutable in Hammerspace using different objectives. To avoid confusion, it's important to understand how the different objectives work and when to use each one. The three objectives important for WORM are:

- `deny-write` — When active, this objective prevents subject files from being modified. Appending data, overwriting data, and filename changes are not allowed. It is typically paired with `deny-delete`, though it may be used by itself.
- `deny-delete` — When active, this objective prevents subject files from being deleted. `deny-delete` is typically used along with `deny-write`, though it may be used by itself.
- `worm-operation` — This is a compound objective which combines the `deny-delete` and `deny-write` objectives with some attributes and conditional logic. Its definition in Hammerscript looks like this (carriage returns added for clarity):

```
IF ATTRIBUTES.WORM_EXPIRE_DATE>NOW
OR ATTRIBUTES.LEGAL_HOLD_EXPIRE_DATE>NOW
THEN {SLO('deny-delete'),SLO('deny-write')}
```

This is an excellent example of the flexibility of Hammerspace objectives, and follows a simple IF A OR B THEN C format. There are two attributes referenced, `WORM_EXPIRE_DATE` and `LEGAL_HOLD_EXPIRE_DATE`. If the value of either of these attributes is in the future (`>NOW`), then the two listed Service Level Objectives (SLOs) `deny-delete` and `deny-write` take effect. Attributes are just one category of metadata in Hammerspace that can be used to define objectives.

In the Hammerspace GUI, the `WORM_EXPIRE_DATE` attribute is set when enabling `worm-operation` on a share using the checkbox on the Share Details tab, as shown below. If `worm-operation` is configured elsewhere, such as on the Objectives tab of the share properties, or on the command line, `WORM_EXPIRE_DATE` must be set manually using the Hammerspace Toolkit (HSTK) `hs attribute` command.

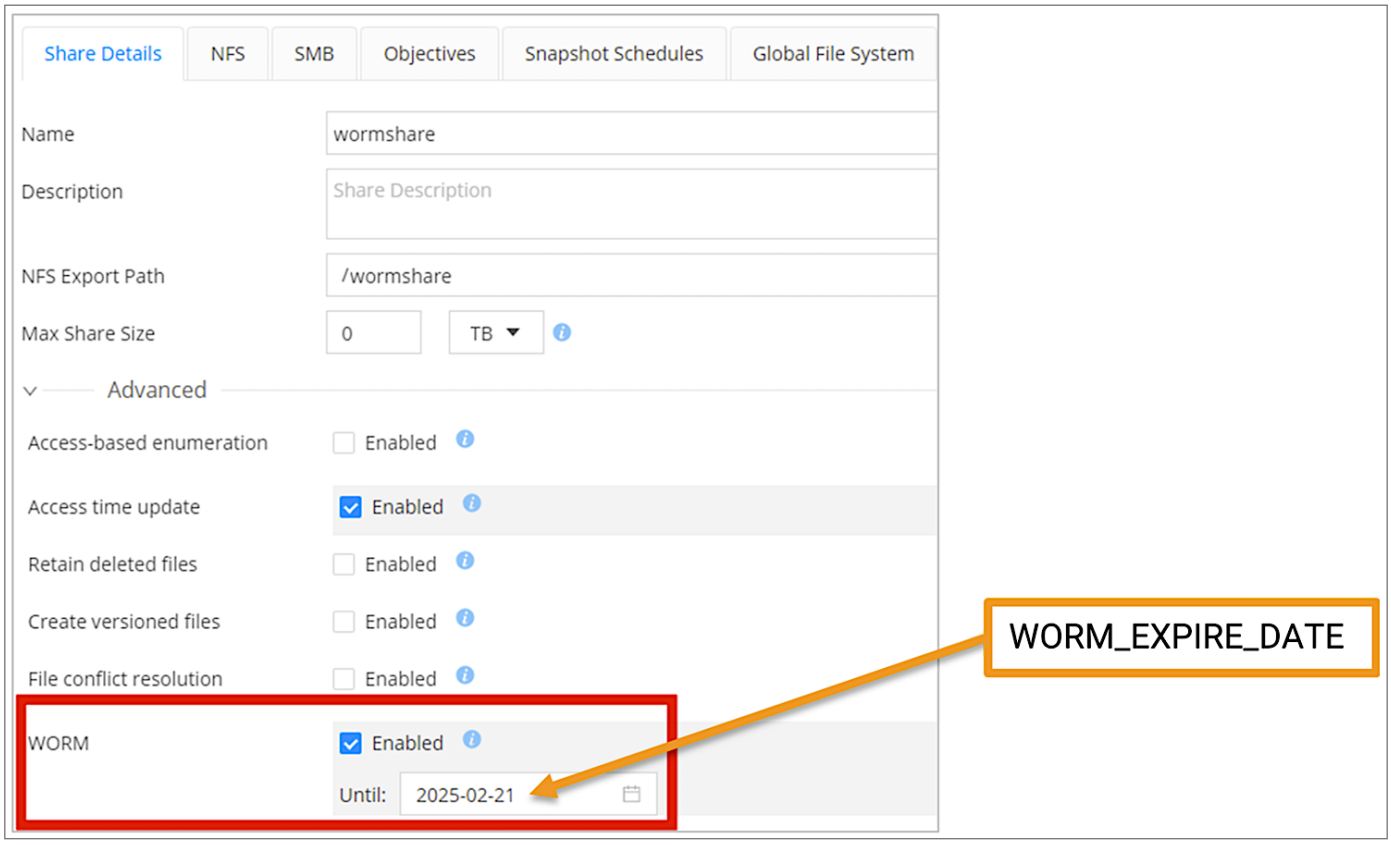

Figure 2. Share Details tab showing the WORM checkbox and expiration date field

`LEGAL_HOLD_EXPIRE_DATE` does not currently appear in the GUI, but like all attributes, it may be set using HSTK or through other command line methods.

For more general information on attributes, refer to the Hammerspace Objectives Solutions Guide found on the Hammerspace Support Hub.

So, when should `worm-operation` be used vs. `deny-write` and `deny-delete`? It depends on the goal. If the need is simply to create a share where the entire contents will be protected from changes and deletions until a particular date, use `worm-operation`. For more sophisticated needs, as with most of the use case examples later in this document, use `deny-write` and `deny-delete` to provide immutability.

Chapter 4. WORM and the Hammerspace S3 Interface

Hammerspace shares may be exposed to clients via multiple protocols simultaneously, including SMB, NFS, and S3. Directories and files appear as buckets and objects when exposed via S3.

In AWS S3 API terms, WORM is known as Object Lock. AWS offers a wide variety of configuration options for Object Lock. Hammerspace supports a subset consisting of four Object Lock-related S3 APIs:

- `PutObjectLockConfiguration` — Sets Object Lock policy on a bucket
- `GetObjectLockConfiguration` — Displays Object Lock policy
- `PutObjectLegalHold` — Sets legal hold status on an object
- `GetObjectLegalHold` — Displays legal hold status of an object

Object Lock provides two ways to manage object retention, retention periods and legal holds. Retention periods have a defined expiration date, and legal holds do not.

4.1. S3 Object Lock Retention Period

An Object Lock retention period is a policy that applies to an S3 bucket and all objects in that bucket. With Hammerspace, all objects in the bucket are protected against deletion using `deny-write` and `deny-delete` objectives, and the Hammerspace attribute `WORM_EXPIRE_DATE` is set to match the requested policy.

When a retention period is specified, one of two retention modes, Compliance or Governance, must also be specified. Hammerspace only supports the stricter Compliance mode at this time. Attempts to configure Governance mode will return a `(501): Not Implemented` error.

The following command sets an Object Lock policy on a bucket named `s3bucket` with a retention period of 30 days using AWS CLI with Windows PowerShell. The `{"RequestCharged"}` response indicates that the command was executed successfully.

Set a 30-day Object Lock retention period on a bucket

Command:

```
PS> aws s3api put-object-lock-configuration --bucket s3bucket --object-lock-configuration
'{\"ObjectLockEnabled\": \"Enabled\", \"Rule\": { \"DefaultRetention\": { \"Mode\": \"COMPLIANCE\",
\"Days\": 30 }}}'
```

Expected output:

```
{
    "RequestCharged"; "RequestCharged"
}
```

The JSON is easier to read when doing the corresponding get operation to view the configuration:

View the Object Lock configuration on a bucket

Command:

```
PS> aws s3api get-object-lock-configuration --bucket s3bucket
```

Expected output:

```
{
    "ObjectLockConfiguration": {
        "ObjectLockEnabled": "Enabled",
        "Rule": {
            "DefaultRetention": {
                "Mode": "COMPLIANCE",
                "Days": 30
            }
        }
    }
}
```

After setting the bucket policy, the Hammerspace `WORM_EXPIRE_DATE` attribute reflects the configured retention period:

Verify the WORM_EXPIRE_DATE attribute

Command:

```
PS> Y:\s3bucket\> hs attribute get WORM_EXPIRE_DATE .
```

Expected output:

```
LOCAL_TIME('2025-04-22 12:28:47')
```

Retention periods are not reciprocal. Enabling a retention period on a bucket sets `deny-write` and `deny-delete` objectives on the corresponding directory, but setting the `deny-write` and `deny-delete` objectives on a directory will **not** cause a retention period to be enabled on the corresponding bucket.

4.2. S3 Object Lock Legal Hold

Legal hold is a property that applies to individual objects. It is a simple ON/OFF toggle with no concept of an expiration date. The status (ON or OFF) is set using the `PutObjectLegalHold` API by a bucket owner. The `GetObjectLegalHold` API displays the current status of an object.

Similar to `WORM_EXPIRE_DATE`, Hammerspace defines an attribute named `LEGAL_HOLD_EXPIRE_DATE`. This attribute is not used in conjunction with legal hold, however, since legal hold does not use expiration dates.

Unlike retention period, legal hold is reciprocal. Enabling legal hold on an object will set `deny-write` and `deny-delete` objectives on the corresponding file, and setting those objectives on a file **will** toggle the legal hold status to ON as viewed by `GetObjectLegalHold`.

The commands shown below set object lock on an object, and verify the status has been set ON.

Set legal hold on an object

Command:

```
PS> aws s3api put-object-legal-hold --bucket s3bucket --key client42 --legal-hold Status=ON
```

Expected output:

```
{
    "RequestCharged": "RequestCharged"
}
```

Verify the legal hold status of an object

Command:

```
PS> aws s3api get-object-legal-hold --bucket s3bucket --key client42
```

Expected output:

```
{
    "LegalHold": {
        "Status": "ON"
    }
}
```

Chapter 5. Deployment Considerations and Best Practices

While WORM is a straightforward concept, it must be implemented thoughtfully since it impacts how users and applications access data. This section contains recommendations and things to consider before deploying WORM storage with Hammerspace, along with specific configuration and product-related best practices.

5.1. WORM Deployment Considerations

Understand the business requirements: As with any IT systems deployment, thoroughly understanding the business requirements is key to ensuring that they will be met. For WORM deployments, this includes answering questions like these:

- Is Compliance WORM required or is Enterprise WORM sufficient? If the system must be certified to meet specific industry regulations, Compliance WORM is probably required.
- What is driving the WORM storage requirement? Is it a particular application, regulation or business need? (e.g. intellectual property protection, ransomware resistance, etc.)
- Is WORM a new requirement or is there an existing system that requires integration with Hammerspace?

Understand the data: What data types need to be protected with WORM? Where are the files stored — on which type or types of storage? Must all copies of subject files reside on WORM storage, or is it desirable or required to also have copies that can be modified?

Understand the applications: Which applications and users generate the data that needs WORM protection? Which applications and users need to use the data after it is locked down? Do the applications understand WORM storage and behave appropriately?

Most applications assume they are using regular read/write data storage. Some, such as certain backup and archive applications, may understand how to use WORM storage directly. Non-WORM-aware applications may work normally with WORM storage, simply generating errors on attempts to change or delete locked files, but they may also exhibit unwanted behavior, such as creating temporary files that then cannot be deleted or failing to create or save files at all.

These screenshots from the Windows Notepad and Microsoft Word applications show the types of errors that may be encountered when attempting to use a WORM share as working space for an application.

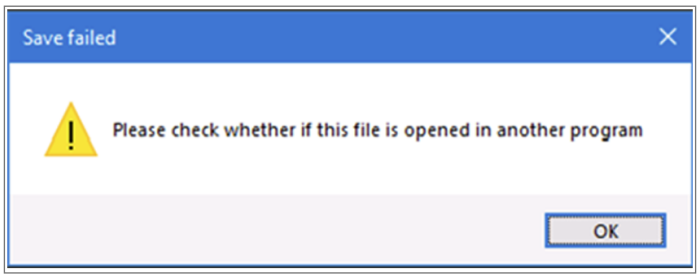

Figure 3. Windows Notepad error when saving to a WORM share

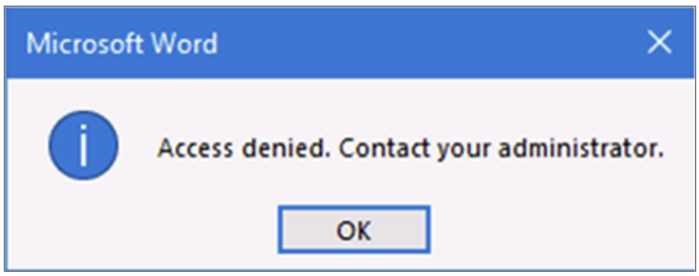

Figure 4. Microsoft Word error when saving to a WORM share

Application issues with WORM storage can be overcome using Hammerspace objectives or by adjusting the workflow. For example, configuring the WORM objective to only take effect once files have been closed for a specified duration enables applications to create files normally. Or using regular writeable storage while files are actively being created or updated, then tiering final files to the WORM archive using objectives.

Understand the workflow: What is the workflow for the files that need WORM protection? Will it need to change to implement WORM? Once files are protected, how are they accessed when they are needed again? If used only for reference, files may be accessed directly from the WORM repository. If files need to be updated or used as the basis for new work, a copy might need to be made to a writable share as a first step.

Make backups: WORM is "data protection" in the sense that it can prevent files from being deleted or changed. But it is not by itself "backup." As with any data storage, if the storage system managed by Hammerspace is destroyed, the data residing on it will be lost. Use Hammerspace objectives to ensure backup best practices are met. This should include keeping multiple copies, on different media types, and at least one off-site copy for DR recovery and one off-line copy for ransomware protection.

Follow security best practices: Because WORM in Hammerspace is software-based, it is important to limit access to the Hammerspace administrative console to authorized individuals. Similarly, access to the storage resources used with WORM objectives must be limited to Hammerspace. Secure audit logs should be enabled for traceability of admin actions and system activity. Role-based access control (RBAC) principles and other standard security best practices should always be followed.

5.2. Hammerspace WORM Best Practices

Use the appropriate objectives: Use the `worm-operation` objective or `deny-write`, `deny-delete` objective pair depending on the need, as discussed earlier. Remember that `worm-operation` relies on either the `WORM_EXPIRE_DATE` or `LEGAL_HOLD_EXPIRE_DATE` attributes being set to function.

Apply WORM at the correct level: WORM objectives can be applied at the file, directory, or share level. Applying a WORM objective at the share level is the most secure, provided the ALWAYS applicability option is used. When an objective is applied ALWAYS it cannot be reversed further down the tree, as is the case when using applicability of TRUE.

There are use cases where it makes sense to apply WORM at the directory or file level, but be aware WORM protections at this level are more easily reversed. Only a Hammerspace administrator may remove WORM protections on a share. Any user with "Data Owner" privileges on Hammerspace and a copy of the Hammerspace Toolkit installed may change or remove directory and file-level WORM protections on files they have access to.

Use appropriate objectives with WORM targets: Hammerspace can place files and objects on targets that support WORM, including object stores. This is useful when Compliance WORM is required for some or all data. Keep in mind, however, that Hammerspace has no innate knowledge of the fact that files written to the target cannot be changed or deleted. To align the Hammerspace configuration with the characteristics of a WORM target, `deny-write`, `deny-delete`, and `confine-to` objectives must be applied to all files or objects that are placed on such a target.

Understand multi-site configurations: One of the powerful features of Hammerspace is the ability to create global namespaces that span multiple sites and clouds. Hammerspace objectives are configured separately for each site, and only affect files at the site where they are configured. When using WORM with multiple sites and global file shares, ensure that the WORM objectives are applied at every site in exactly the same way so that protection is not compromised.

It's also important with any multi-site configuration — not just those involving WORM — to consider the desired behavior of the system when one site is down or temporarily unreachable. Refer to the Hammerspace documentation (available to customers in the Hammerspace Support Hub) and the Hammerspace Objectives Solution Guide for more information regarding multi-site configurations and global file shares.

Take care with scripting: Because of the way Hammerspace SMB shares interact with client tools such as the Windows command line and PowerShell, deletions of files that are forbidden by active `deny-delete` and `worm-operation` objectives will at first appear to succeed, not returning any error. Refreshing the directory listing will show that the files have in fact not been deleted. This may be confusing to human users and scripts alike.

Scripts that delete files that could be protected by these objectives must take this into account, for example, by checking after a few seconds to see that the file deletion has truly succeeded.

When mounting shares with NFS, attempts to delete protected files will produce a prompt asking if you want to remove the write protected file or files. If the prompt is answered `Y`, the deletion will fail with a "Permission denied" error as shown below.

```
# Attempt to delete WORM-protected file 'file3.txt' over NFS
$ ls
file1.txt  file2.txt  file3.txt
$ rm file3.txt
rm: remove write-protected regular file 'file3.txt'? Y
rm: cannot remove 'file3.txt': Permission denied
$
```

Chapter 6. Configuring Hammerspace WORM

The following instructions walk through the process of configuring a basic WORM-enabled share in Hammerspace, followed by additional examples tailored for specific use cases. Keep in mind when viewing these simple examples that the techniques shown may be combined and used with additional conditional logic to meet nearly any requirement.

All steps shown using the GUI may also be accomplished using the CLI or programmatically.

Examples include:

1. Configuring a WORM share including an expiration date (GUI)
2. Extending the expiration date of a WORM share (GUI)
3. Setting WORM on a directory with `deny-write` and `deny-delete` (GUI)
4. Setting WORM on a directory with `deny-write` and `deny-delete` (CLI)
5. Setting WORM on a directory with `worm-operation`, including setting the `WORM_EXPIRE_DATE` attribute (CLI)
6. Delaying WORM application until files reach a specified age (GUI)
7. Expiring WORM on files once they reach a specified age (GUI)
8. Archiving to a Compliance WORM storage target (GUI)
9. Sending snapshots to a Compliance WORM target (GUI)
10. Setting WORM on a file based on a user-applied label (GUI)

6.1. Configuring a WORM Share Including an Expiration Date

This is a basic WORM configuration, useful for archive use cases where populations of data must be saved for a specified amount of time. For example, retaining details of financial filings for seven years. In this case a share could be created for each year's files, with an expiration date set seven years forward.

Follow these steps after logging into the Hammerspace GUI as an administrator.

1. Navigate to the **Data** tab of the left menu bar and click **Create Share**.

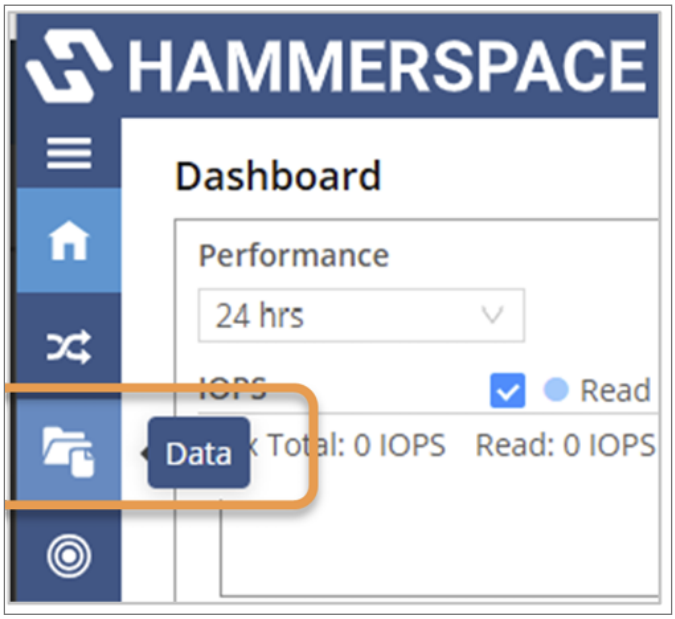

Figure 5. Create a share from the Data tab

2. Enter a name for the share, then click **Advanced**. Check the box to enable WORM, and enter a date in the **Until** field. This defines a date when files in the directory will be unlocked and able to be modified or deleted.

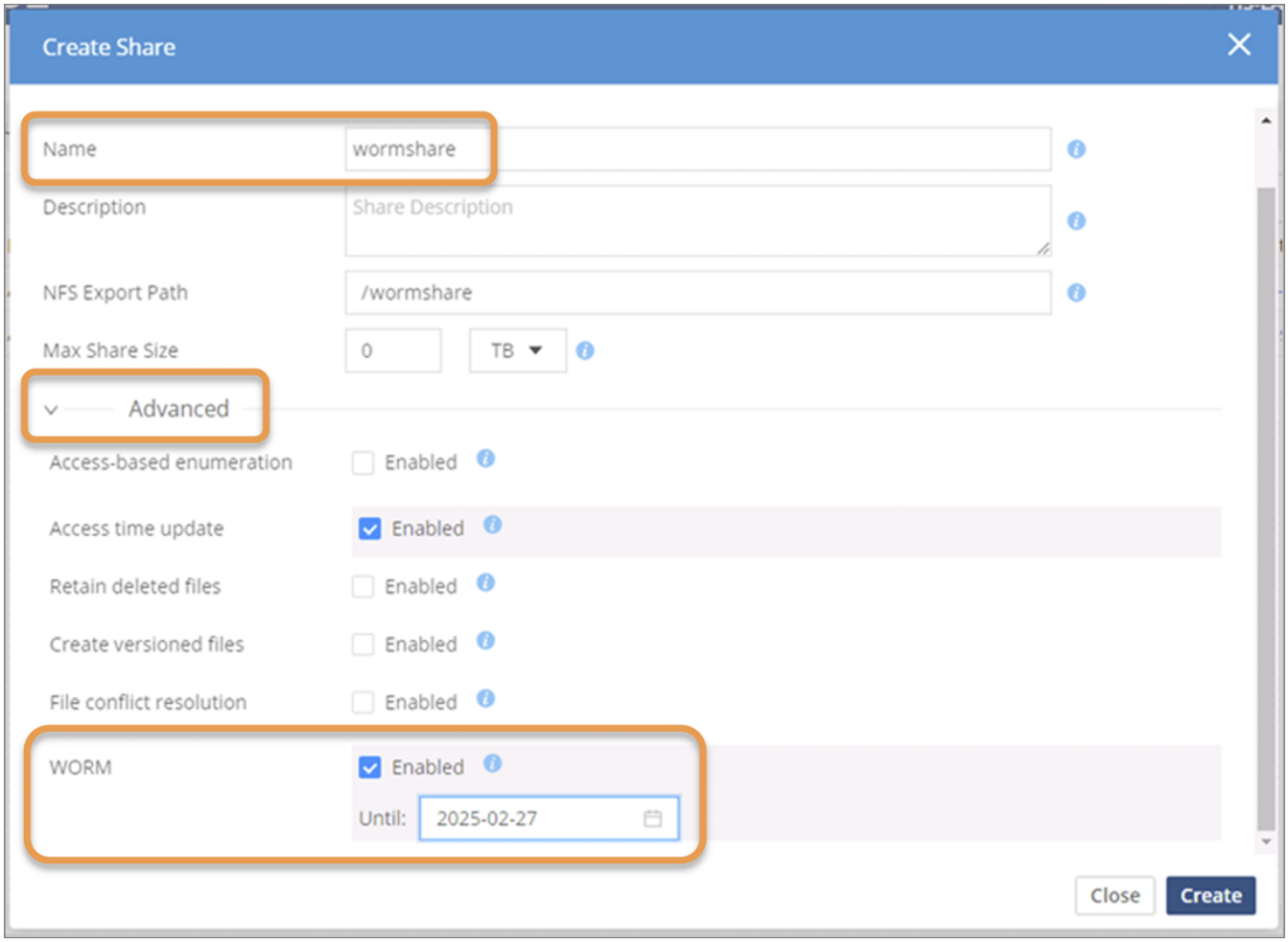

Figure 6. Advanced share settings with the WORM checkbox and Until date

3. Move to the **Objectives** tab. Note that **Applicability** is set to ALWAYS. This ensures that WORM always applies to the entire share and cannot be removed on any files or directories within it.

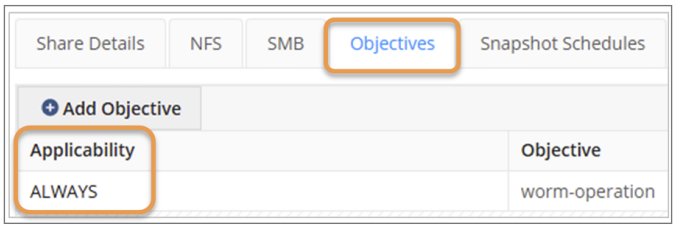

Figure 7. Objectives tab showing Applicability set to ALWAYS

4. Click the **Create** button. After a few seconds, the new share will appear in the list of shares.

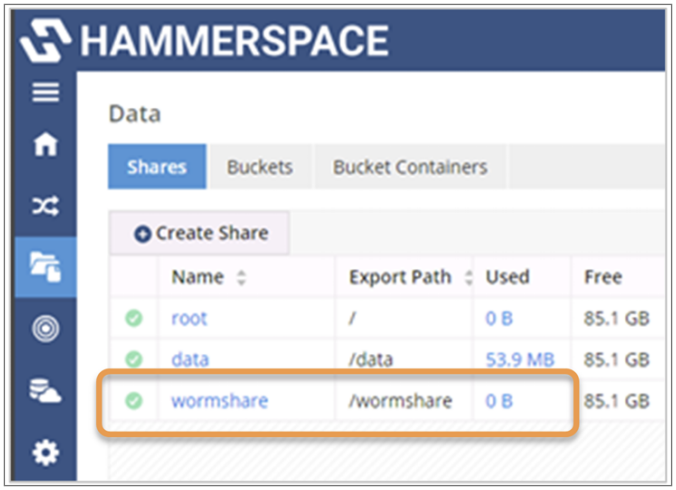

Figure 8. The new WORM share in the list of shares

It is also possible to add the WORM objective to an existing share, simply by modifying the advanced share properties and checking the WORM checkbox as one would when creating a new share. WORM will take effect immediately on any existing and new files.

6.2. Extending the Expiration Date of a WORM Share

Once a share has the WORM objective set, the expiration date (the WORM "Until" date) may be modified, but only in one direction. The duration may be increased, but not decreased.

To extend the expiration date:

1. Log into the Hammerspace administrative GUI and click the **Data** tab on the left menu bar.
2. Click the pencil icon under **Actions** to edit the options for the share to be modified.
3. Open the **Advanced** options on the **Share Details** tab and click in the **WORM Until** field to open the calendar. The current Until date will be highlighted. Note that prior dates are not selectable.

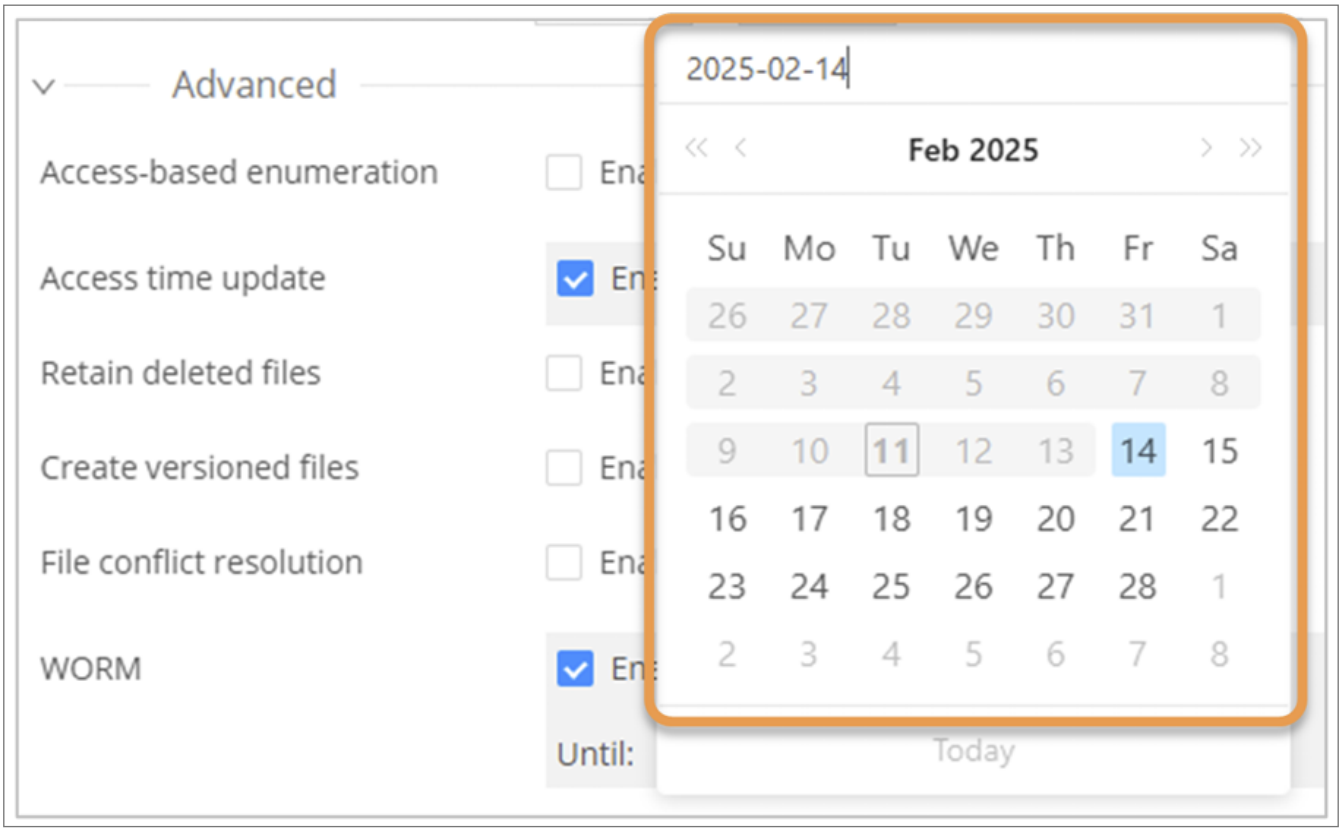

Figure 9. WORM Until calendar with past dates disabled

4. Click a future date to extend the date, then click **Update** to apply the change.

6.3. Setting WORM on a Directory Using the GUI

These steps may also be used to set WORM on individual files.

1. From the **Data** tab in the Hammerspace GUI, click the name of the share (**share1** in this example) to bring up the share dashboard. Click on the **Files** tab. The **Files** tab will display the files and directory listing for the root of the share.

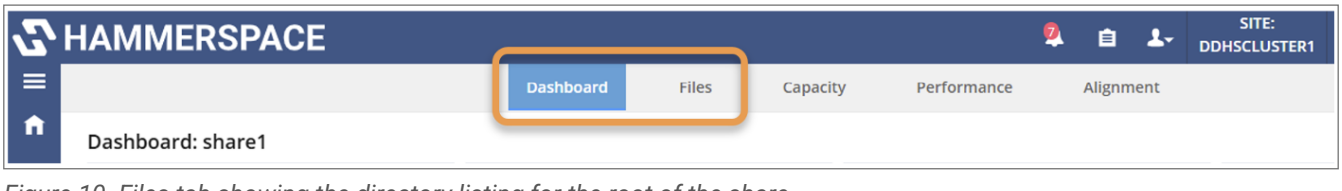

Figure 10. Files tab showing the directory listing for the root of the share

2. Click the pencil icon to edit the objectives that apply to that directory.

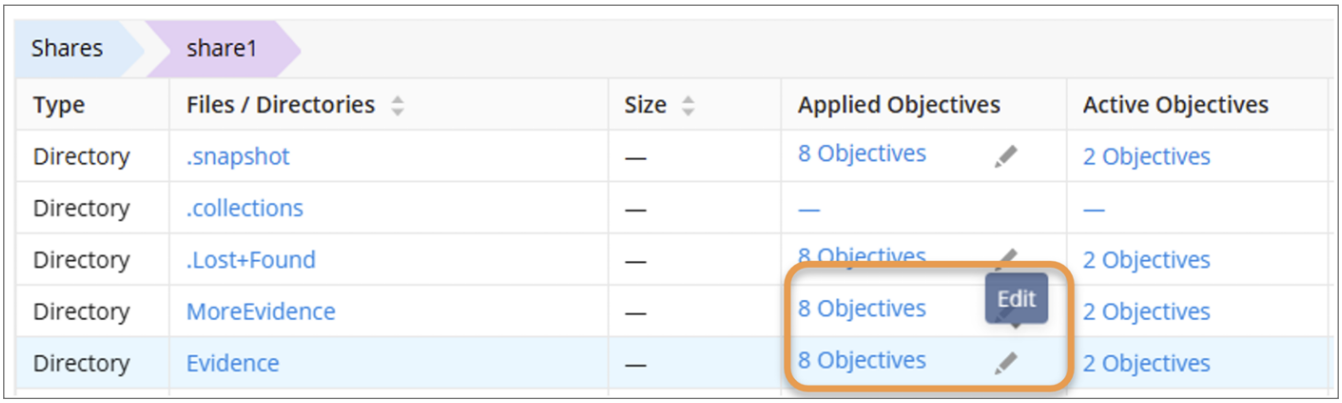

Figure 11. Edit the objectives that apply to the directory

3. Click **Add Objective**.

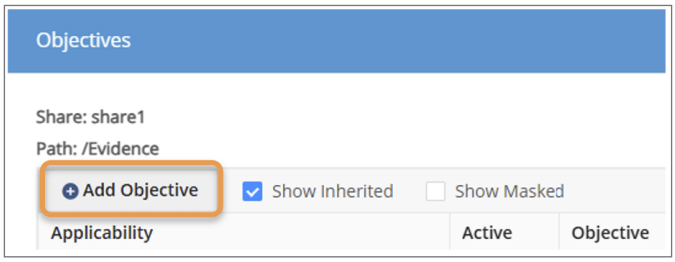

Figure 12. Add an objective to the directory

4. Click the checkboxes for both the `deny-delete` and `deny-write` objectives.

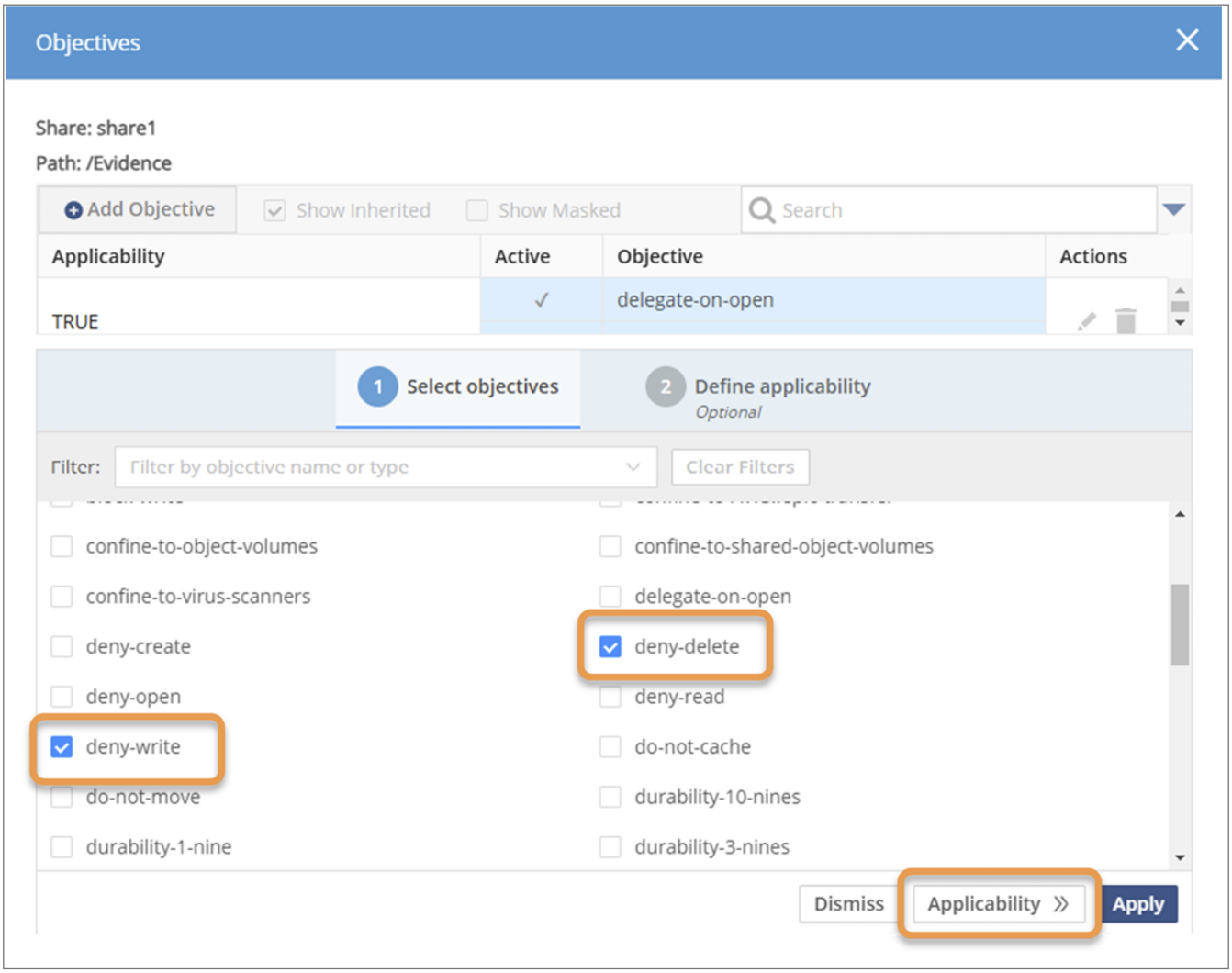

Figure 13. Select the deny-delete and deny-write objectives

5. Click the **Applicability** button and select **ALWAYS** to prevent the objectives from being removed inside the directory, and click **Apply** to apply the change.

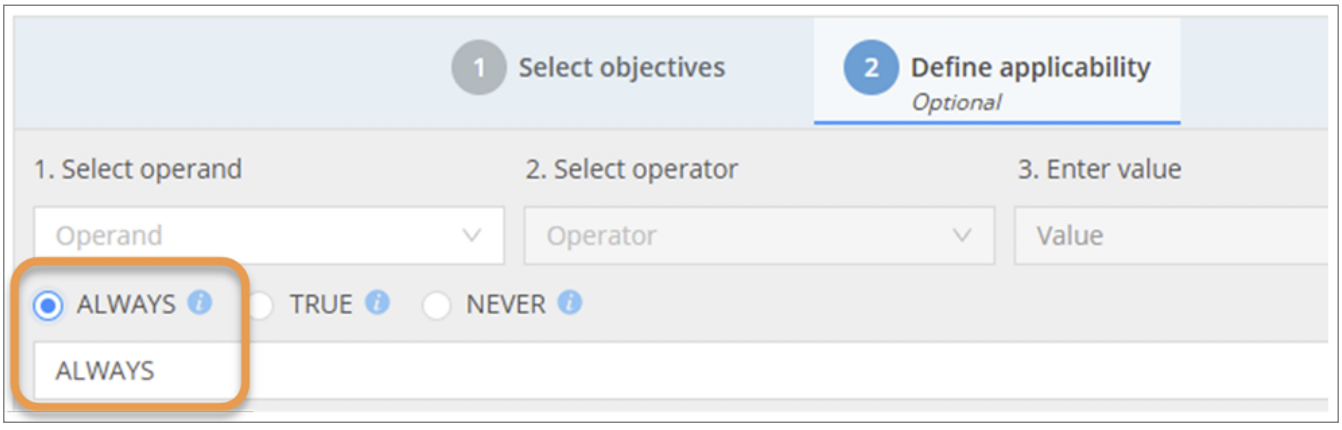

Figure 14. Set Applicability to ALWAYS

6. The new objectives will appear in the list.

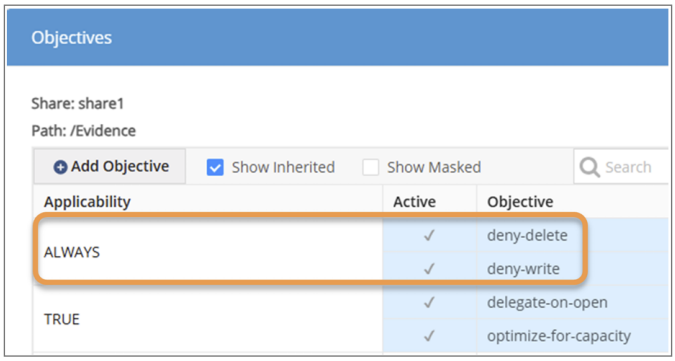

Figure 15. The new objectives in the list

6.4. Setting Deny-Write and Deny-Delete on a Directory Using HSTK

As with the previous example, this process may also be used to set objectives on files. This example uses a Windows system, but the commands are the same regardless of the OS the Hammerspace Toolkit is running on. For Windows, the use of PowerShell is recommended as it parses text input a little differently than the standard command line.

For this example, the `S:` drive is mapped to the share named **share1**, the same one used in the previous example.

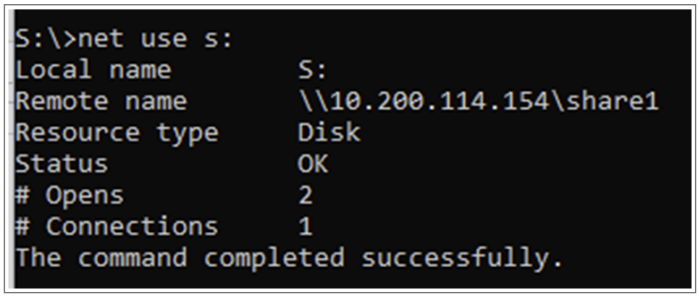

Figure 16. The S: drive mapped to share1

The `hs objective add` command may be used to apply the `deny-write` and `deny-delete` objectives on a directory or file. In this case, they are used on the directory named **Evidence**.

1. First, create the directory with `md`.

```
md Evidence
```

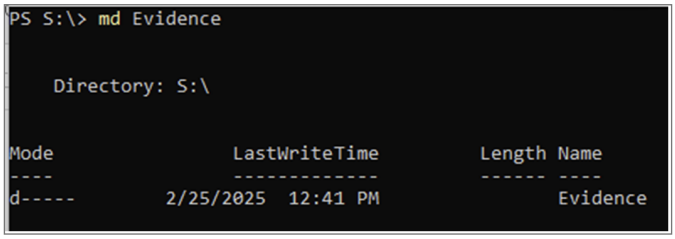

Figure 17. Create the Evidence directory with md

2. Then, apply the objectives with `hs objective add`.

```
hs objective add deny-write Evidence
hs objective add deny-delete Evidence
```

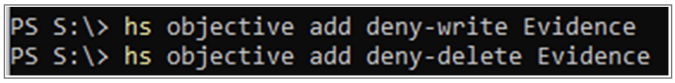

Figure 18. Apply the deny-write and deny-delete objectives

3. Finally, use `hs objective list` to list the objectives on the directory to validate that the new ones were applied. In this case they appear at the top.

```
hs objective list Evidence
```

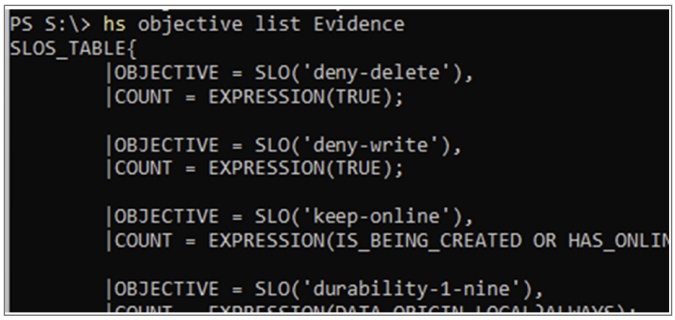

Figure 19. List the objectives on the directory

6.5. Setting Worm-Operation on a Directory Using HSTK

This example is similar to the previous one, but instead of using `deny-delete` and `deny-write`, the `worm-operation` objective is used. This requires setting the `WORM_EXPIRE_DATE` attribute as well.

1. First, create a directory named **CaseArchive** using `md`.

```
md CaseArchive
```

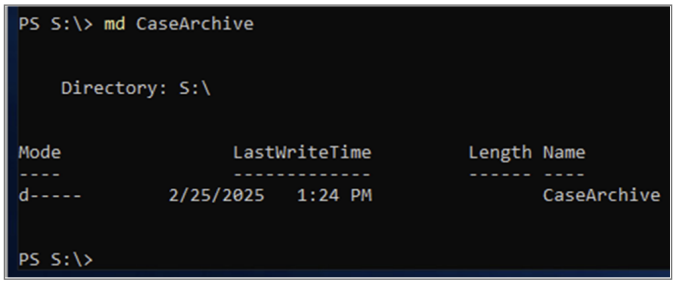

Figure 20. Create the CaseArchive directory with md

2. Next, set the `worm-operation` objective using `hs objective add`, and verify it using `hs objective list`.

```
hs objective add worm-operation CaseArchive
hs objective list CaseArchive
```

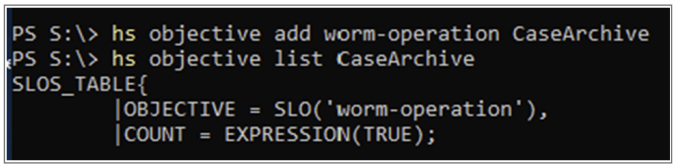

Figure 21. Set and verify the worm-operation objective

3. Finally, remember that for `worm-operation` to work, the attribute `WORM_EXPIRE_DATE` must be set, which is done with `hs attribute set` and verified with `hs attribute get`.

```
hs attribute set WORM_EXPIRE_DATE CaseArchive
hs attribute get WORM_EXPIRE_DATE CaseArchive
```

Figure 22. Set and verify the WORM_EXPIRE_DATE attribute

6.6. Delaying WORM Application Until Files Reach a Specified Age

For some workflows, it may be desirable to not have WORM take effect on files right away. For example, a workflow scanning documents or photos that then require human QA or cleanup before being archived in immutable form. Using WORM storage as working storage avoids the need to move the finished files from one share to another, and delaying the onset of WORM based on file age enables applications to work correctly for the pre-WORM period.

The two figures below show how to configure a share objective that waits until seven days have passed since files were created, at which point the `deny-write` and `deny-delete` objectives are applied. Perform this configuration from the **edit share** dialog, **Objectives** tab.

The objective uses a file-age condition with a "greater than or equal" operator so that the WORM objectives apply only once files reach the specified age:

```
IF ACTUAL_CREATE_AGE>=7*DAYS THEN {SLO('deny-write'),SLO('deny-delete')}
```

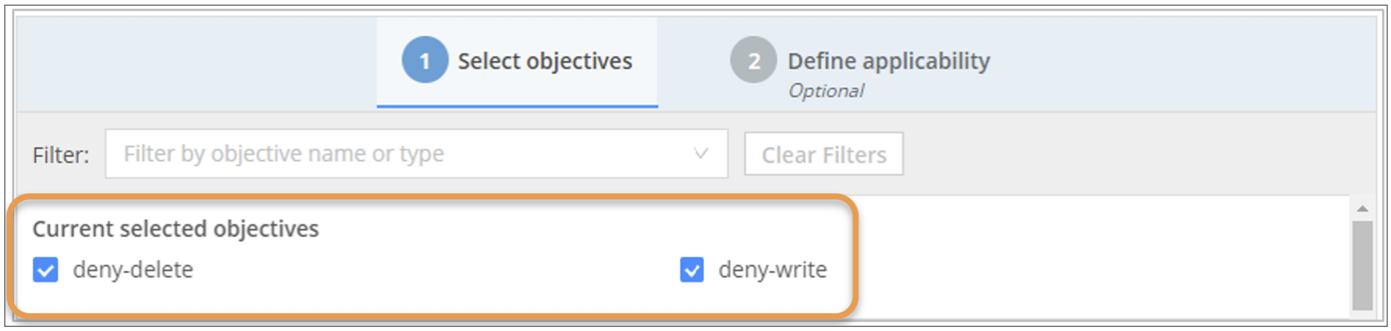

Figure 23. File-age condition that delays WORM until files are seven days old

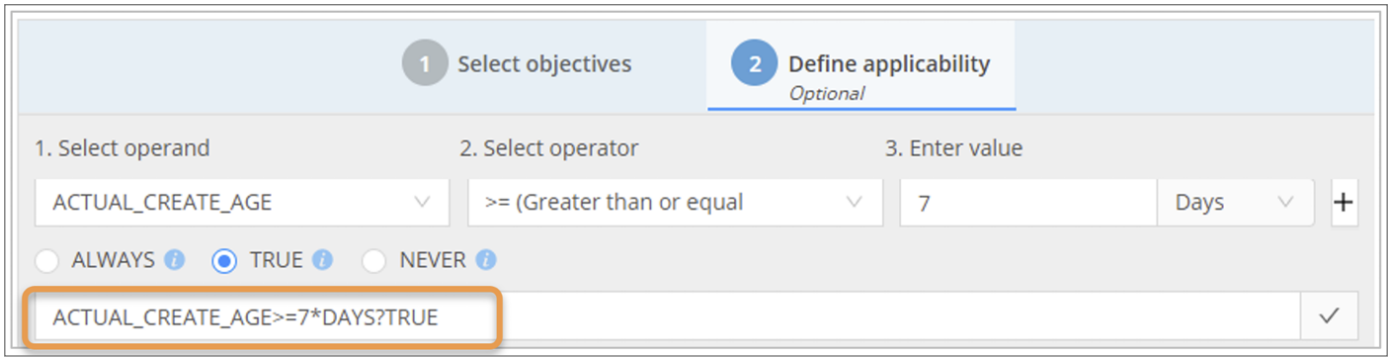

Figure 24. The deny-write and deny-delete objectives applied after the delay

6.7. Expiring WORM Based on File Age

Another common requirement is to make files immutable for some defined period, such as 30 days or seven years. This simply requires setting a WORM objective that operates for the required duration based on the age of the file. After the objective expires, files may be changed or deleted as needed.

Note that this is different from the first example, which sets a general expiration date for WORM on the share without considering the age of individual files.

Some use cases requiring this type of time-limited objective (with typical retention times) include:

- Protecting backup images from ransomware (14–30 days)
- Secure retention of log files (1 year)
- Retaining business records for audit compliance (7 years)

The objective shown below protects all files in the share with WORM for 90 days after they are created. Notice it uses the same operand as in the previous example, but with a "less than or equal" operator vs. "greater than or equal."

```
IF ACTUAL_CREATE_AGE<=90*DAYS THEN {SLO('deny-write'),SLO('deny-delete')}
```

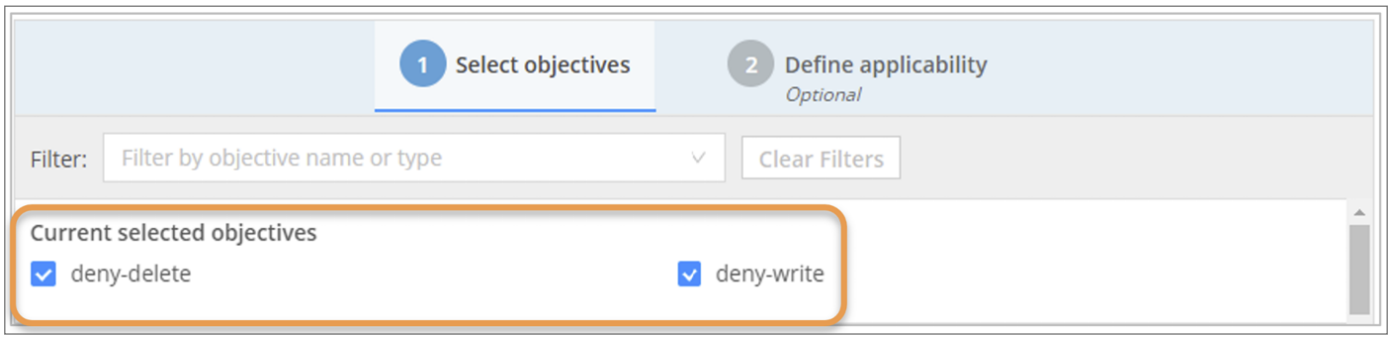

Figure 25. File-age condition that expires WORM 90 days after files are created

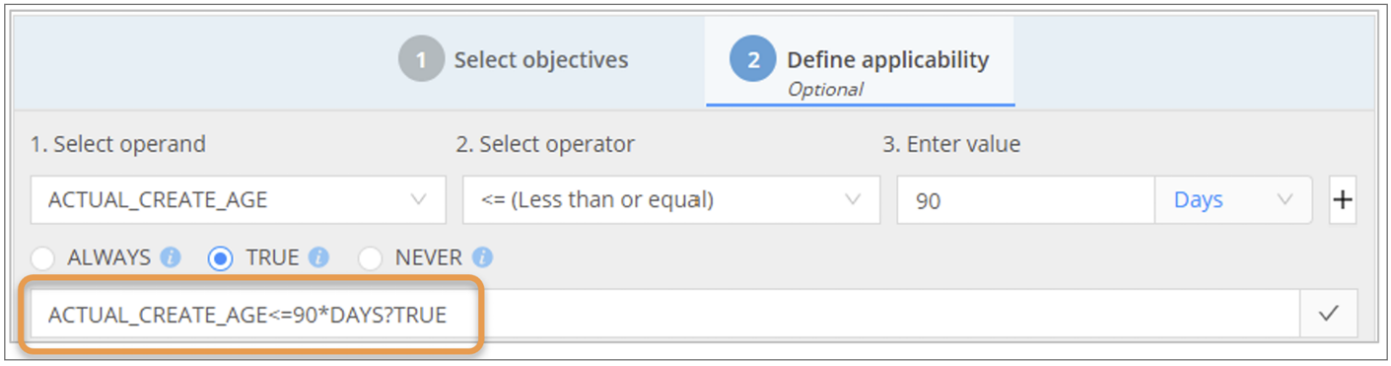

Figure 26. The deny-write and deny-delete objectives applied for the retention period

The workflow for some WORM use cases requires deleting files at the end of their retention time. For example, an organization may need to retain certain records immutably for seven years and then delete them.

For safety reasons, Hammerspace does not include objectives to automatically delete files, but administrators may use Hammerspace CLI commands to generate a list of files eligible for deletion. The file list may be used with standard OS command line or scripting tools to automate the deletion task itself. Details are beyond the scope of this document, but consult the Hammerscript and the Hammerspace Toolkit Guide for ideas, specifically the section entitled "Using Collections for Data Lifecycle Management."

6.8. Archiving to a Compliance WORM Target

Hammerspace can integrate with storage targets that provide Compliance WORM capability, such as immutable file stores or S3 buckets. Many Compliance WORM S3 targets are available, both on premises and in public clouds. Examples include AWS S3 Object Lock, Azure Immutable Blob storage, and even LTO-WORM tape behind a supported S3-to-tape gateway.

As mentioned in the best practices section above, when using Hammerspace with a Compliance WORM target, an objective must be created that sets both the `deny-write` and `deny-delete` objectives on all files or objects on that target.

The following objective configuration confines all files in this share to an AWS bucket that has Object Lock enabled. `deny-write` and `deny-delete` are also set. The objective is configured to apply ALWAYS so that data owners may not remove it.

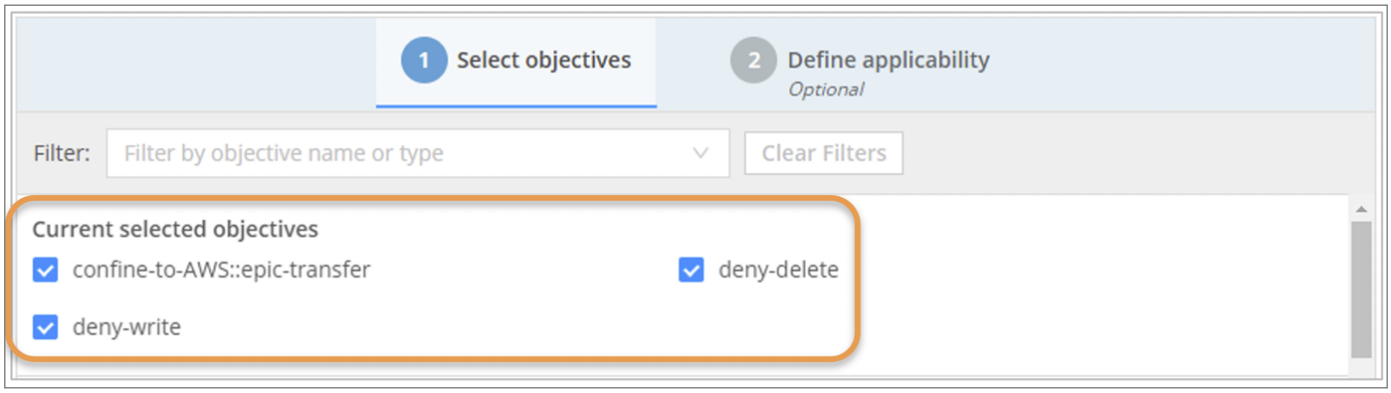

Figure 27. Objective confining all files to an Object Lock-enabled AWS bucket

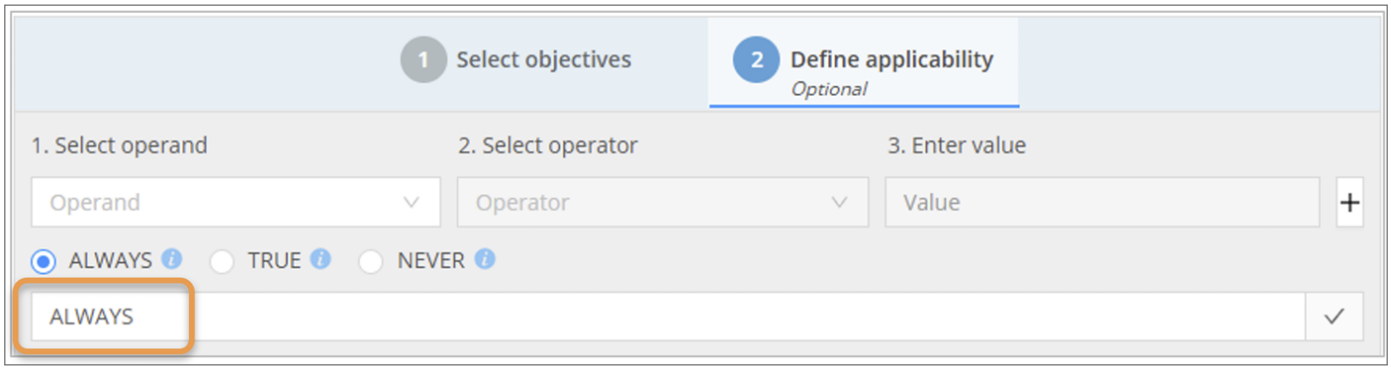

Figure 28. The deny-write and deny-delete objectives applied with Applicability ALWAYS

Any files copied into this share will only reside on the specified immutable AWS bucket.

6.9. Sending Snapshots to a Compliance WORM Target

In order to meet requirements for immutable snapshots, Hammerspace objectives may be configured to copy snapshots to an on-premises or cloud-based WORM storage target.

Hammerspace may be configured with a snapshot retention schedule. If this feature is used, ensure that the WORM target bucket has a retention period configured that is shorter than the Hammerspace snapshot retention schedule. Otherwise, the scheduled deletion of expired snapshots will fail.

In this example two objectives are used to protect a copy of all snapshots to the WORM target. The first places all snapshots older than the current snapshot onto the WORM cloud target.

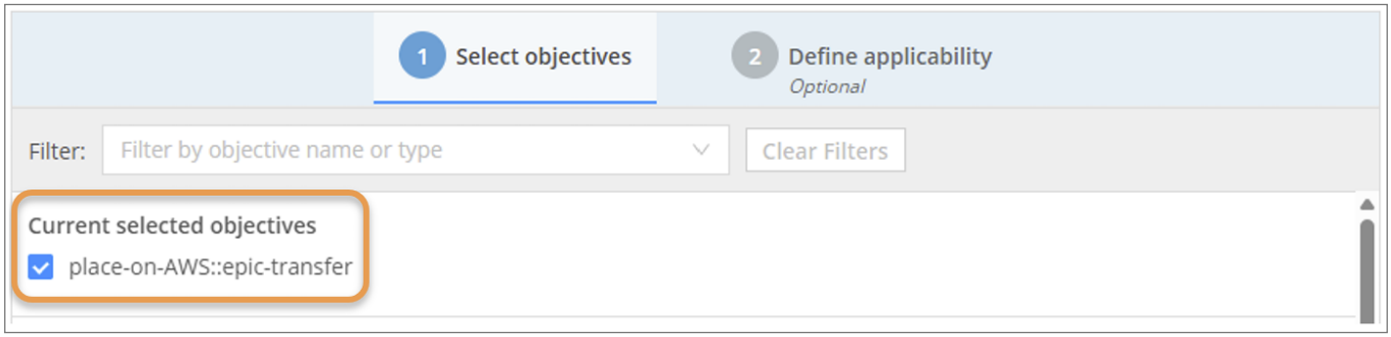

Figure 29. Objective placing snapshots older than the current snapshot onto the WORM cloud target

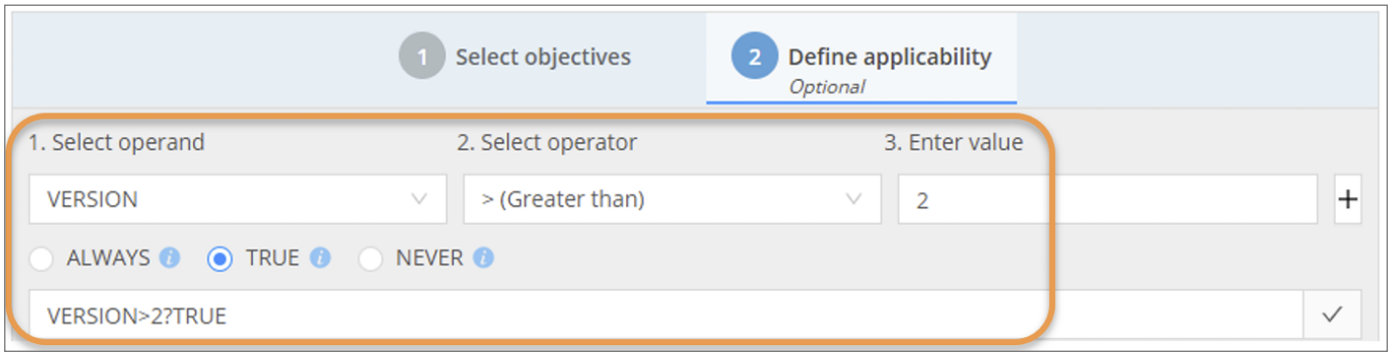

Figure 30. The first objective configured on the share

The second ensures that the current snapshot is always kept in the original location on premises for quick recovery, as well as having a copy on the WORM cloud target.

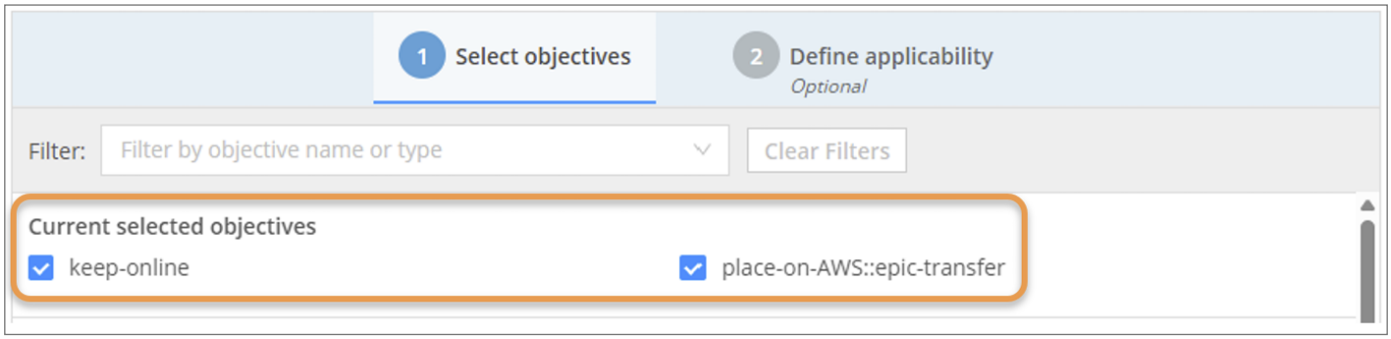

Figure 31. Objective keeping the current snapshot on premises while also copying it to the WORM cloud target

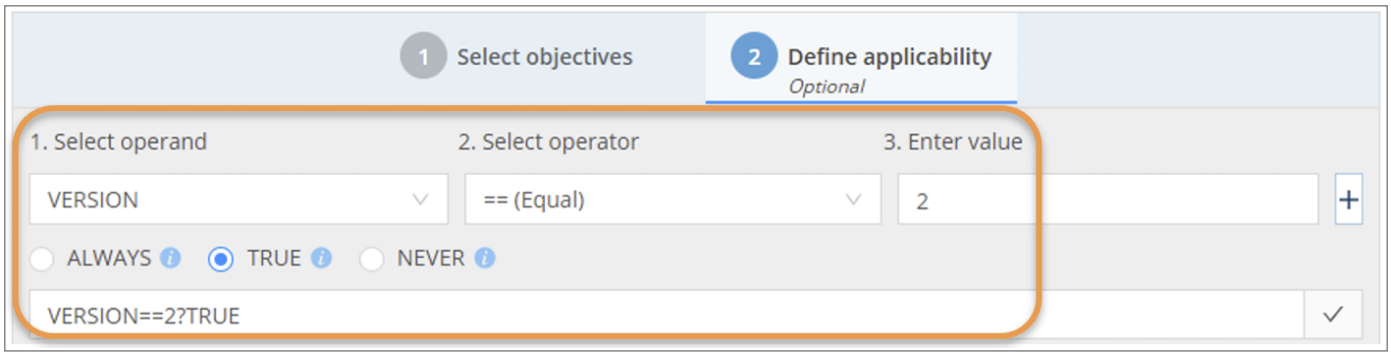

Figure 32. The second objective configured on the share

6.10. Setting WORM on a File Through a User-Applied Label

Hammerspace has very flexible support for user-defined metadata. This includes the ability for end-users to select from a set of labels that have been pre-defined by the administrator. Labels may be applied to files and directories using the HSTK, or via the Hammerspace Metadata plugin for Windows Explorer. Usage of both these tools is covered in detail in the Hammerscript and the Hammerspace Toolkit Guide.

This example configuration enables users to set the `legalhold` label on files. This triggers an objective that makes the file immutable.

If a user has permission to apply a label to a file, they also have permission to remove it. In this case, that means that a user can remove the WORM objective from a file by removing the `legalhold` label. One option to increase security in this scenario is to only provide specific users (such as the legal department) with the tools to manipulate labels. Restricting the use of these tools also restricts other legitimate uses of Hammerspace, however, which may be undesirable.

1. First, a Hammerspace administrator creates the label using the `label-create` command at the admin CLI.

   Create the legalhold label

   Command:

   ```
   label-create --name legalhold
   ```

   Expected output:

   ```
   Name: legalhold
   Internal ID: 1
   ```

2. Next, the administrator configures the `deny-write` and `deny-delete` objectives to be set on all files that have the `legalhold` label. The operand `HAS_LABEL` is not available in the dropdown list, but is typed in the text field below. This field can accept any valid Hammerscript expression. The check mark to the right of the field will appear when the expression is valid. If the check mark is not seen on the right side, the expression as typed is not valid — check the syntax.

   ```
   HAS_LABEL('legalhold')
   ```

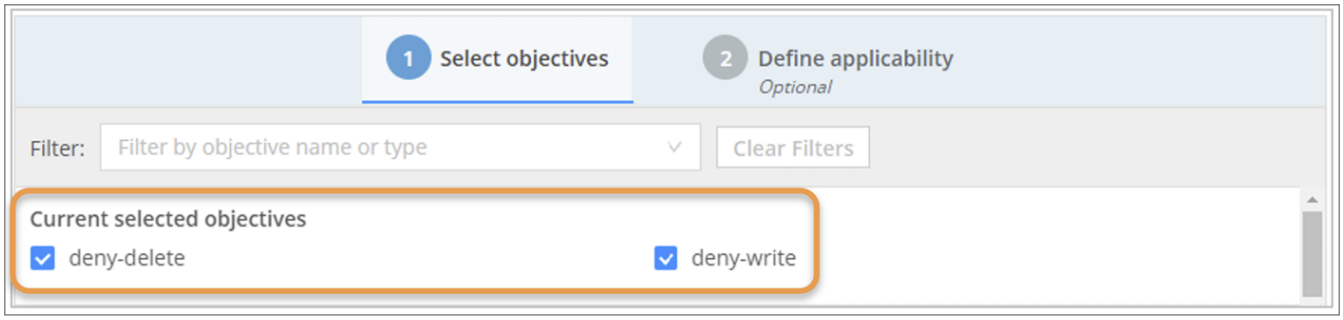

Figure 33. Objective editor with the typed HAS_LABEL expression and the valid-expression check mark

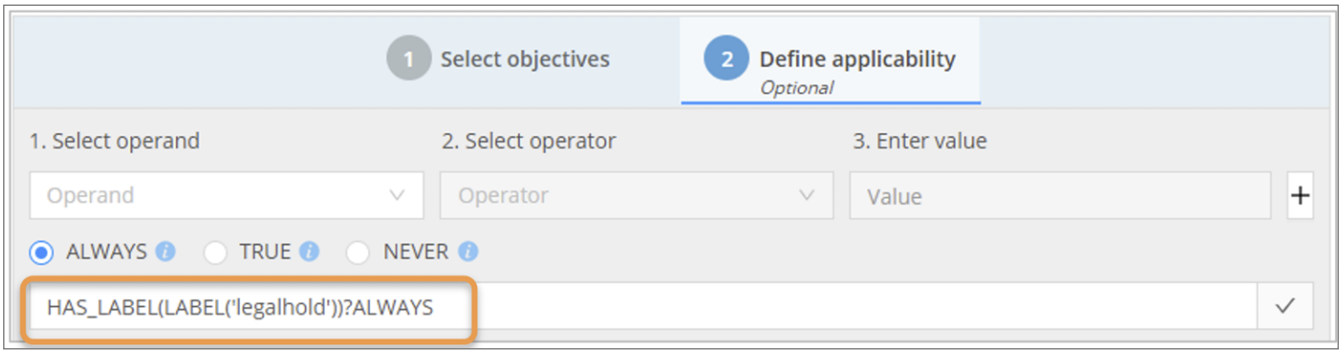

Figure 34. The deny-write and deny-delete objectives set for labeled files

3. At this point, any user on Windows with the Hammerspace Metadata Plugin installed may place the label `legalhold` on a file or directory. The plugin is invoked by right-clicking on any file or directory in Windows Explorer, then selecting **Open Hammerspace** from the context menu.

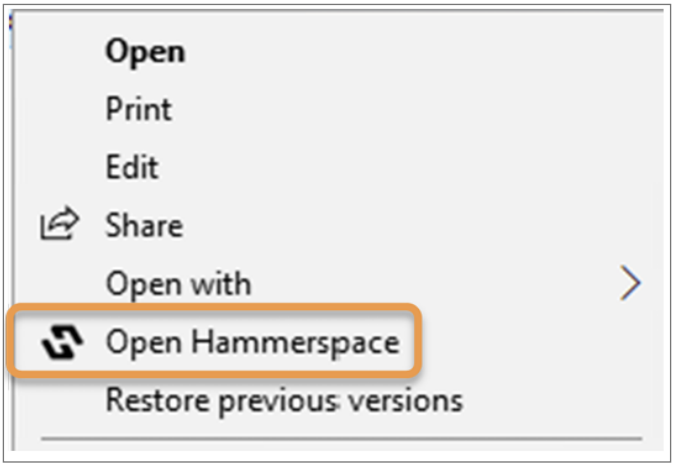

Figure 35. Windows Explorer context menu showing Open Hammerspace

The label may also be applied at the CLI on any Linux, macOS, or Windows machine running the HSTK, or at the system CLI by an administrator.

After clicking the **Select Label** button, the user is presented with a picklist of pre-defined labels, and in this case, selects the `legalhold` label.

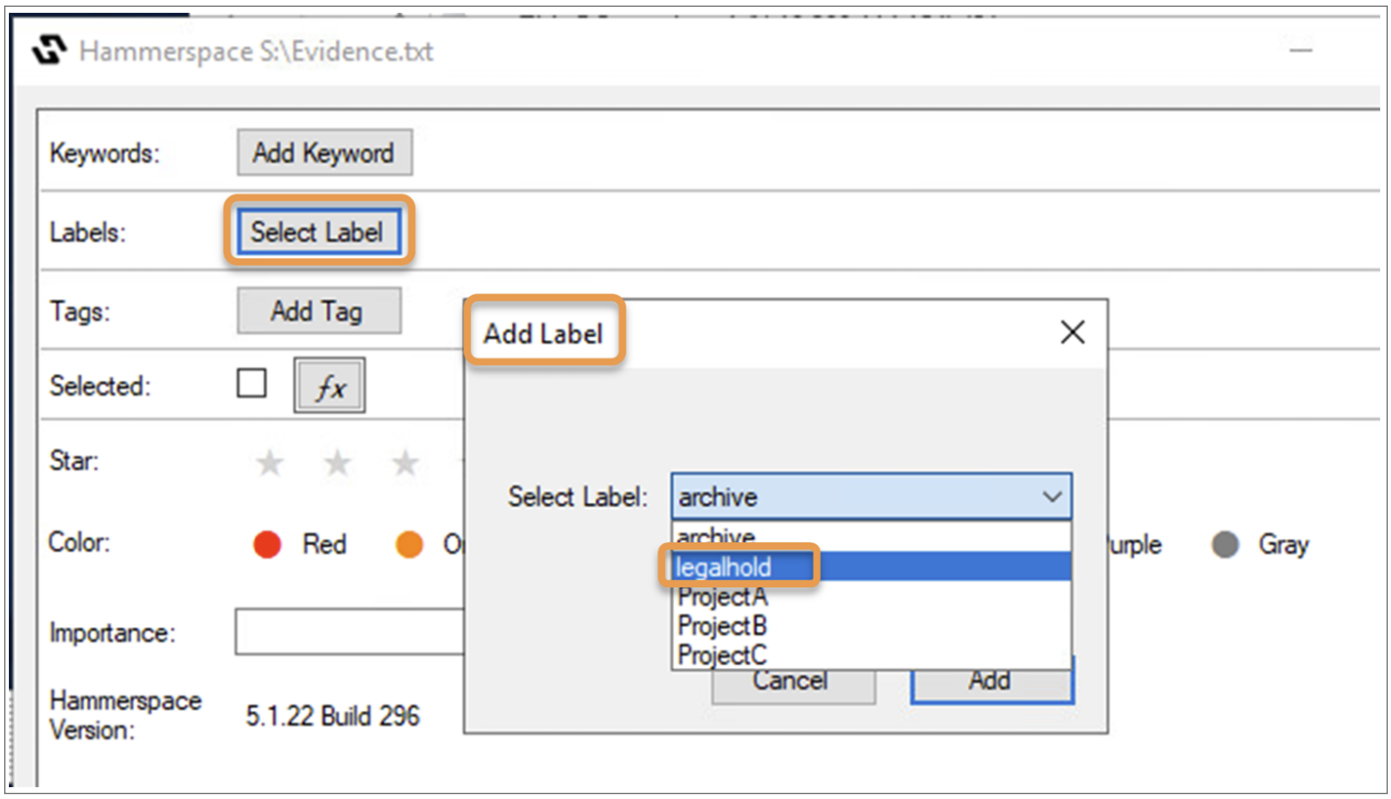

Figure 36. Select Label picklist with the legalhold label selected

Labeling files and directories triggers the pre-defined objective. Affected files will not be able to be modified or deleted until the label is removed or an administrator removes the objective.

Chapter 7. Conclusion

Hammerspace objectives are an extremely capable and flexible tool that may be used to meet data-related business requirements, including many that require immutability. This document detailed a few example objectives for specific use cases, but the possibilities for customization are infinite.

Appendices

Appendix A: Additional Resources

- Configure Hammerspace Objectives
- Hammerspace Support Hub