Swordfish Scalable Storage Management Error Handling Guide

Version 1.2.1

ABSTRACT: The Swordfish Scalable Storage Management Error Handling Guide provides a summary of the preferred handling of errors and error messages in a Swordfish implementation.

Working Draft

Last Updated 18 August 2020

USAGE

DISCLAIMER

Suggestions for revisions should be directed to http://www.snia.org/feedback/.

Revision History

Table 1: Revision history

Date	Revision	Notes
29 May 2020	1.2.0	Initial Release
18 August 2020	1.2.1	Minor editorial corrections

Contact SNIA

SNIA Web Site

Current SNIA practice is to make updates and other information available through the SNIA web site at http://www.snia.org.

FEEDBACK AND INTERPRETATIONS

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent via the SNIA Feedback Portal at http://www.snia.org/feedback/ or by mail to the Storage Networking Industry Association, 4360 ArrowsWest Drive, Colorado Springs, Colorado 80907, U.S.A.

INTENDED AUDIENCE

This document is intended for use by individuals and companies engaged in storage management.

VERSIONING POLICY

This document is versioned material. Versioned material shall have a three-level revision identifier, comprised of a version number 'v', a release number 'r' and an errata number 'e'. Future publications of this document are subject to specific constraints on the scope of change that is permissible from one revision to the next and the degree of interoperability and backward compatibility that should be assumed between products designed to this standard. This versioning policy applies to all SNIA Swordfish versioned materials.

Version Number: Versioned material having version number 'v' shall be backwards compatible with all of revisions of that material that have the same version number 'v'. There is no assurance of interoperability or backward compatibility between revisions of a versioned material with different version numbers.

Release Number: Versioned material with a version number 'v' and release number 'r' shall be backwards compatible with previous revisions of the material with the same version number, and a lower release number. A minor revision represents a technical change to existing content or an adjustment to the scope of the versioned material. Each minor revision causes the release number to be increased by one.

Errata Number: Versioned material having version number 'v', a release number 'r', and an errata number 'e' should be backwards compatible with previous revisions of the material with the same version number and release number ("errata versions"). An errata revision of versioned material is limited to minor corrections or clarifications of existing versioned material. An errata revision may be backwards incompatible, if the incompatibility is necessary for correct operation of implementations of the versioned material.

About SNIA

The Storage Networking Industry Association (SNIA) is a non-profit organization made up of member companies spanning information technology. A globally recognized and trusted authority, SNIA's mission is to lead the storage industry in developing and promoting vendor-neutral architectures, standards and educational services that facilitate the efficient management, movement and security of information.

Acknowledgements

The SNIA Scalable Storage Management Technical Work Group, which developed and reviewed this work in progress, would like to recognize the significant contributions made by the members listed in Table 2.

Table 2: Contributors

Member	Representatives
Broadcom Inc.	Richelle Ahlvers
Cisco Systems, Inc.	Krishnakumar Gowravaram
Hewlett Packard Enterprise	Chris Lionetti
NetApp, Inc.	Don Deel

Table of Contents

1 Introduction

1.1 Audience

This guide is intended to provide a common repository of best practices, common tasks and education for handling error conditions in a Swordfish implementation.

1.2 Documentation structure

This document assumes that the reader has a solid foundation in restful APIs in general and Swordfish in particular. Based on that understanding, this document presents a set of error handling scenarios that capture common situations and best practices. They are intended to promote the complete, correct, and consistent handling of errors and error messaging across Swordfish implementations.

Each error case uses a common template. Table 3 lists each field of the template and its description.

Table 3: Guidelines for the Use Case Template

Name	**Description**
Title	A description of the high-level scope of the error
Summary	A high-level summary of the error
Example	The specific example that will be used to illustrate the error case
Basic Course of Events	A sequence of API requests, including required headers, the body of the request, and the expected reply
Additional Context	Clarifying material, and additional detail intended to clarify subtleties of the error case or to highlight additional response options.

1.3 Base implementation assumptions

This document assumes that some fundamental configuration issues have been properly implemented, and will not need to be addressed in any detail. In particular, this document assumes:

- An appropriate security infrastructure (e.g., TLS 1.2)
- A functional Swordfish/Redfish installation, in either a standalone, aggregator, or distributed configuration
- Any required login credentials

1.4 Knowledge assumptions

The Swordfish API conforms to the standards defined in the Redfish API (http://redfish.dmtf.org). More generally, it is provides a RESTful interface. The reader is assumed to be familiar with common conventions for RESTful APIs. Those readers who are interested in additional background information are encouraged to refer to the following sources:

- For RESTful APIs: Wikipedia (https://en.wikipedia.org/wiki/Representational_state_transfer)
- For HTTP standards: Wikipedia (https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
- For Redfish standards: Redfish Specification (http://redfish.dmtf.org)
- For Swordfish standards: Swordfish Specification (http://www.snia.org/swordfish)
- For Swordfish API tutorials: Swordfish Tutorials (http://www.snia.org/swordfish)

2 HTTP status codes

2.1 Overview

The HTTP status codes are defined by RFC2616 by W3.org, and are intended to address a broad range of HTTP implementations. Both the Redfish specification and the Swordfish specification provide information about usage for a subset of HTTP status codes. In addition, the server can return extended status information as a simple JSON object to further clarify the handling and outcome of a particular API request; guidance on when to use extended status and error information is also specified in the Redfish and Swordfish Specifications.

While Swordfish clients may receive any of the standard HTTP status codes, the Redfish and Swordfish Specifications include an explicit list that must be supported. In addition, as this subset of HTTP codes provides a detailed mapping from generic HTTP status codes to domain-specific situations and probable causes, they should be the most common and as the only required status codes, implementations should target their use exclusively as much as possible. This enables clients to implement with little to no vendor-specific instrumentation.

2.2 Related information

For more information, see:

- The Swordfish Specification (http://snia.org/swordfish)
- The Redfish Specification (http://redfish.dmtf.org)
- The HTTP Protocol definition of HTTP status codes (https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html).

3 Error Types

3.1 General Errors

This group of error cases summarized in Table 3 deals with general error cases.

Errors in this group tend to be simple, non-specific mistakes or general notification that are handled through an error message, and do not reflect any problem with the storage system or its configuration.

Table 3: General Errors

Error Case	Common Error Cause(s)
NoOperation	The request will neither result in a change in the service, nor a change to a resource.

3.2 Action Errors

The group of error cases summarizd in Table 4 deals with errors arising from actions.

Errors in this group tend to indicate invalid parameters, or a resource/parameter/action mismatch.

Table 4: Action errors

Error Message	Common Error Cause(s)
ActionNotSupported	The requested action is not supported by the selected resource.
ActionParameterDuplicate	The body of the request contains duplicate parameter settings.
ActionParameterMissing	The requested action requires a parameter that was not supplied.
ActionParameterNotSupported	The parameter supplied for the action is not supported on the resource.
ActionParameterUnknown	The request contains unknown parameter settings.
ActionParameterValueTypeError	A parameter was given the wrong value type, such as when a number is supplied for a parameter that requires a string.

3.3 JSON Errors

The group of error cases summarized in Table 5 deals with mistakes in formatting the JSON required by an API request.

Errors in this group are simple syntactic mistakes or omissions in the API request, and do not reflect any problem with the storage system or its configuration.

5: JSON Errors

Error Message	Common Error Cause(s)
EmptyJSON	The request requires a JSON body, and none was included
MalformedJSON	The JSON body included in the request is malformed, and could not be parsed

3.4 Property Errors

The group of error cases summarized in Table 6 deals with error in the selection or handling of property values.

Errors in this group tend to arise from property-level constraints in the schema or a particular implementation.

Table 6: Property errors

Error Message	Common Error Cause(s)
PropertyMissing	The request does not include all of the properties required to process it.
PropertyNotWritable	The request to change a **single** property, references a property that is read-only.
PropertyValueNotInList	The request uses a valid value type for a given property according to the Swordfish specification, but the implementation does not support that value.

3.5 Resource Errors

The group of error cases summarized in Table 7 deals with general error cases.

Errors in this group tend to be simple, non-specific mistakes or general notification that are handled through an error message, and do not reflect any problem with the storage system or its configuration.

Table 7: Resource errors

Error Message	Common Error Cause(s)
ResourceAlreadyExists	A CREATE the implementation cannot be accepted, because the resource already exists.
ResourceCannotBeDeleted	The named resource cannot be deleted.
ResourceInUse	The requested change cannot be completed because the resource is in use or in transition.
ResourceNotFound	The DELETE request references a resource that cannot be found.

4 Error Cases

4.1 *Error Case: ActionNotSupported*

Summary: When a client sends a request to a Swordfish implementation and the action supplied with the POST operation is defined within the schema, but is not supported by the implementation, the `ActionNotSupported` message shall be returned.

Example: User tries to suspend replication on a Volume.

Basic Course of Events:

1. The user attempts to suspend replication on a Volume.

Request: `POST /redfish/v1/StorageServices/ISC/Volumes/1/Volume.SuspendReplication`

Headers: No additional headers required.

Body:

```
{
  "TargetVolume": "/redfish/v1/Storage/1/Volumes/650973452245"
}
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ActionNotSupported",
    "message": "The action SuspendReplication is not supported by the resource."
  }
}
```

4.2 *Error Case: ActionParameterDuplicate*

Summary: When a client sends a request to a Swordfish implementation and more than one value is provided for a parameter to the action, the `ActionParameterDuplicate` message shall be returned.

Example: User attempts to create a new volume to serve as a target replica for an existing source volume, but provides duplicate or conflicting values for `ReplicaUpdateMode`.

Basic Course of Events:

1. Post (as an Action) the request on the source Volume.

This instructs the service to use the identified Volume as the source Volume for the specified replication relationship. For any additional details required, the service will rely on default values.

Request: `POST /redfish/v1/Storage/1/Volumes/1/Volume.CreateReplicaTarget`

Headers: No additional headers required.

Body:

```
{
 "VolumeName" : "Mirror of Volume 65",
 "ReplicaUpdateMode" : "Synchronous",
 "ReplicaUpdateMode" : "Asynchronous",
 "TargetStoragePool" : "/redfish/v1/Storage/1/StoragePools/PrimaryPool",
 "ReplicaType" : "Mirror"
 }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ActionParameterDuplicate",
    "message": "The action CreateReplicaTarget was submitted with more than one
     value for the parameter ReplicaUpdateMode."
  }
}
```

4.3 *Error Case: ActionParameterMissing*

Summary: When a client sends a request to a Swordfish implementation, but omits one or more of the required parameters for the action, the `ActionParameterMissing` message shall be returned.

Example: User attempts to create a new volume to serve as a target replica for an existing source volume, but fails to specify the TargetStoragePool for the new Volume.

Basic Course of Events:

1. Post (as an Action) the request on the source Volume.

This instructs the service to use the identified Volume as the source Volume for the specified replication relationship. For any additional details required, the service will rely on default values.

Request: `POST /redfish/v1/Storage/1/Volumes/1/Volume.CreateReplicaTarget`

Headers: No additional headers required.

Body:

```
{
 "ReplicaUpdateMode" : "Synchronous",
 "ReplicaType" : "Mirror"
 }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ActionParameterMissing",
    "message": "The action CreateReplicaTarget requires the parameter
     TargetStoragePool to be present in the request body."
  }
}
```

4.4 *Error Case: ActionParameterNotSupported*

Summary: When a client invokes an action against a Resource, but the body of the request include a parameter that is not supported by the current implementation, the `ActionParameterNotSupported` message shall be returned.

Example: The user attempts to remove a replica relationship, and includes the DeleteTargetVolume property, which is not supported in the current implementation.

Basic Course of Events:

1. The user attempts to delete the replication relationship.

Request: `POST /redfish/v1/Storage/Volumes/1/Volume.RemoveReplicaRelationship`

Headers: No additional headers required.

Body:

```
{
 "TargetVolume" : "/redfish/v1/Storage/1/Volumes/42524988",
 "DeleteTargetVolume" : true
 }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ActionParameterNotSupported",
    "message": "The parameter DeleteTargetVolume for the action
     RemoveReplicaRelationship is not supported on the target resource."
  }
}
```

4.5 *Error Case: ActionParameterUnknown*

Summary: When a client sends a request to a Swordfish implementation, but includes an unknown parameter in the action, the `ActionParameterUnknown` message shall be returned.

Example: User attempts to create a new volume to serve as a target replica for an existing source volume, but includes an unknown parameter ("capacity") in the body of the request.

Basic Course of Events:

1. Post (as an Action) the request on the source Volume.

Request: `POST /redfish/v1/Storage/1/Volumes/1/Volume.CreateReplicaTarget`

Headers: No additional headers required.

Body:

```
{
 "Capacity": 100002334153,
 "ReplicaUpdateMode" : "Synchronous",
 "TargetStoragePool" : "/redfish/v1/Storage/1/StoragePools/PrimaryPool",
 "ReplicaType" : "Mirror"
 }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ActionParameterUnknown",
    "message": "The action CreateReplicaTarget was submitted with the invalid
     parameter Capacity."
  }
}
```

4.6 *Error Case: ActionParameterValueTypeError*

Summary: When a client sends a request to a Swordfish implementation, but uses a wrong value type for one or more parameter(s), the `ActionParameterValueTypeError` message shall be returned.

Example: User attempts to create a new volume to serve as a target replica for an existing source volume, but specifies an integer value for the VolumeName.

Basic Course of Events:

1. Post (as an Action) the request on the source Volume.

This instructs the service to use the identified Volume as the source Volume for the specified replication relationship. For any additional details required, the service will rely on default values.

Request: `POST /redfish/v1/Storage/1/Volumes/1/Volume.CreateReplicaTarget`

Headers: No additional headers required.

Body:

```
 {
  "VolumeName": 123456,
  "ReplicaUpdateMode" : "Synchronous",
  "TargetStoragePool" : "/redfish/v1/Storage/1/StoragePools/PrimaryPool",
  "ReplicaType" : "Mirror"
  }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
  {
    "error": {
      "code": "Base.1.6.ActionParameterValueTypeError",
      "message": "The value 123456 for the parameter VolumeName in the action
       CreateReplicaTarget is of a different type than the parameter can accept."
    }
  }
```

4.7 *Error Case: EmptyJSON*

Summary: When a client sends a request to a Swordfish implementation, but fails to include any properties required to process the request when one or more properties are expected, the `EmptyJSON` message shall be returned.

Example: User tries to create a Volume, but omits any properties.

Basic Course of Events:

1. Post the definition of the new volume to the Volumes resource collection with no Body.

Request: `POST /redfish/v1/Storage/1/Volumes`

Headers: No additional headers required.

Body:

```
{


}
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
 {
   "error": {
     "code": "Base.1.6.EmptyJSON",
     "Message": "The request body submitted contained an empty JSON object and the
        service is unable to process it."
   }
 }
```

AdditionalContext: Note: The EmptyJSON case is technically a subset case of the PropertyMissing case. This may end up deprecated from the Redfish Message Registry in time and replaced by pointers to Property Missing instead.

If desired, the implementation can return an ExtendedInfo structure that includes the information about the specific issue (in this case, a pointer to the duplicate CapacityBytes property).

```
   {
     "error": {
         "code": "Base.1.6.EmptyJSON",
         "Message": "The request body submitted contained an empty JSON object and
          the service is unable to process it.",
         "@Message.ExtendedInfo": [
           {
             "@odata.type": "#Message.v1_0_0.Message",
             "MessageId": "Base.1.6.PropertyMissing",
             "RelatedProperties": [
               "#/CapacityBytes"
             ],
             "Message": "The property CapacityBytes is a required property and must
          be included in the request.",
             "MessageArgs": [
               "CapacityBytes"
             ],
             "Severity": "Warning",
             "Resolution": "Ensure that the property is in the request body and has a
          valid value and resubmit the request if the operation failed."
           }
         ]
       }
     }
```

AdditionalContext: None.

4.8 *Error Case: MalformedJSON*

Summary: When a client sends a request to a Swordfish implementation, but the request contains malformed JSON. This could be anything, such as duplicate properties, syntax errors, and the like. In this case, the `MalformedJSON` message shall be returned.

Example: User tries to create a Volume, but omits a required quotation mark.

Basic Course of Events:

1. Post the definition of the new volume to the Volumes resource collection with no Body.

Request: `POST /redfish/v1/Storage/1/Volumes`

Headers: No additional headers required.

Body:

```
{
 "Name" : "MyVolume",
 "RAIDType" : "RAID1",
 "CapacityBytes": 34576345685
 }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
    {
      "error": {
        "code": "Base.1.6.MalformedJSON",
        "Message": "The request body submitted was malformed JSON and could not be
         parsed by the receiving service."
      }
    }
```

AdditionalContext: None.

4.9 *Error Case: NoOperation*

Summary: When a client sends a valid request to a Swordfish implementation, but that request will neither result in a change in the service, nor a change to the resource, the `NoOperation` message should be returned.

Example: User tries to expand an existing Volume, but provides a new value equal to the existing size.

Basic Course of Events:

1. Post the definition of the new volume to the Volumes resource collection with no Body.

Request: `PATCH /redfish/v1/Storage/1/Volumes`

Headers: No additional headers required.

Body:

```
{
 "Name" : "MyVolume",
 "RAIDType" : "RAID1",
 "CapacityBytes": 23049823948
 }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
    {
      "error": {
        "code": "Base.1.6.NoOperation",
        "Message": "The request body submitted contain no data to act upon and no
         changes to the resource took place."
      }
    }
```

4.10 *Error Case: PropertyMissing*

Summary: When a client sends a request to a Swordfish implementation, but fails to include all of the properties required to process the request, the `PropertyMissing` message shall be returned.

This example demonstrates the usage of that message.

Example: User tries to create a Volume, but omits the desired capacity for the volume.

Basic Course of Events:

1. Post the definition of the new volume to the Volumes resource collection but omit the capacity property.

Request: `POST /redfish/v1/Storage/1/Volumes`

Headers: No additional headers required.

Body:

```
{
 "Name" : "MyVolume",
 "RAIDType" : "RAID1"
 }
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
    {
      "error": {
        "code": "Base.1.6.PropertyMissing",
        "Message": "The property CapacityBytes is a required property and must be
         included in the request."
      }
    }
```

AdditionalContext: For this example, we have used “capacity”; there are several caveats to note here: First, the implementation could use one of two different properties to specify capacity. Either the CapacityBytes, or the Capacity.Data.AllocatedBytes property could be used to specify the desired/required capacity for the new volume.

Additionally, this event code could be used to specify a missing Redfish.required property (such as Name or Id). In this example, we have chosen to specify a “required in context” property. This requirement is not explicitly noted in the Volume schema; it is instead noted in the general requirements for implementations, in the user’s guide examples, and in profile definitions.

4.11 *Error Case: PropertyNotWritable*

Summary: When a client sends a request to a Swordfish implementation to change a **single** property, but the requested property is read-only, the `PropertyNotWritable` message shall be returned.

Example: User tries to set the Name property on a Volume.

Basic Course of Events:

1. The user attempts to PATCH the Volume name.

Request: `PATCH /redfish/v1/StorageServices/ISC/Volumes/1`

Headers: No additional headers required.

Body:

```
{
  "@Redfish.Copyright": "Copyright 2015-2019 SNIA. All rights reserved.",
  "@odata.context": "/redfish/v1/$metadata#Volume.Volume",
  "@odata.id": "/redfish/v1/StorageServices/ISC/Volumes/1",
  "@odata.type": "#Volume.v1_2_1.Volume",
  "Id": "1",
  "Name": "Danny's Volume"
}
```

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
    {
      "error": {
        "code": "Base.1.6.PropertyValueNotWritable",
        "message": "The property Name is a read only property and cannot be assigned
         a value."
      }
    }
```

4.12 *Error Case: PropertyValueNotInList*

Summary: When a client sends a request to a Swordfish implementation using a correct value type for a given property, but the implementation does not support the selected value, the `PropertyValueNotInList` message shall be returned.

Example: User requests an unsupported replication type on AssignReplicaTarget

Inputs:

- URL for target volume: `/redfish/v1/Storage/1/Volumes/650973452245`
- Requested replica type: `TokenizedClone`
- ReplicaUpdateMode: `Synchronous`

Basic Course of Events:

1. Post (as an Action) the request on the source Volume.

This instructs the service to use the identified Volume as the source Volume for the specified replication relationship. For any additional details required, the service will rely on default values.

Request: `POST /redfish/v1/Storage/1/Volumes/1/Volume.AssignReplicaTarget`

Headers: No additional headers required.

Body:

```
{
"ReplicaUpdateMode": "Synchronous",
"TargetVolume": "/redfish/v1/Storage/1/Volumes/650973452245",
"ReplicaType": "`TokenizedClone`"
}
```

HTTP Status Code Returned: 501 Not Implemented

Headers: None

Body:

```
{
  "error": {
    "code": "Base.1.6.PropertyValueNotInList",
    "message": "The value TokenizedClone for the property ReplicaType is not in the
         list of acceptable values."
    }
}
```

4.13 *Error Case: ResourceAlreadyExists*

Summary: When a client requests a create operation on a resource, but the implementation will not accept the request because the resource already exists and returns `ResourceAlreadyExists` error.

Example: User tries to add a volume with the same ID as an existing volume.

Basic Course of Events:

1. POST the volume to the Volumes collection.

Request: `POST /redfish/v1/Systems/1/Storage/1/Volumes`

Headers: No additional headers required.

Body:

```
{
 "ID":"1",
 "CapacityBytes": "9284327497"
}
```

HTTP Status Code Returned: 409

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ResourceAlreadyExists",
    "Message": "The requested resource of type Volume with the property ID with
     the value 1 already exists."
  }
}
```

Additional Context: This message also technically covers a PATCH case (“change or”) in the Redfish Base Message Registry (v1.6.1). At this time, we do not have any specific examples to cover this case.

4.14 *Error Case: ResourceCannotBeDeleted*

Summary: When a client requests a delete operation on a resource that cannot be deleted, the implementation will return a `ResourceCannotBeDeleted` error.

Example: User tries to remove a storage controller.

Basic Course of Events:

1. Delete the storage controller from the Storage object.

Request: `DELETE /redfish/v1/Systems/1/Storage/1#/StorageController/0`

Headers: No additional headers required.

Body: None.

HTTP Status Code Returned: 405

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ResourceCannotBeDeleted",
    "Message": "The delete request failed because the resource requested cannot
     be deleted."
  }
}
```

Additional Context: This use case covers deletions that are prevented by schema notation (i.e., `deletable = false`). It also covers deletions that are allowed by the schema but are prohibited by the implementation.

4.15 *Error Case: ResourceInUse*

Summary: When a client requests a change to a resource, but the change is rejected by the implementation due to the resource being in use or a transitional state, returns the `ResourceInUse` error.

Example: User tries to delete a volume marked with a VolumeUsage of "InUse".

Basic Course of Events:

1. DELETE the volume.

Request: `DELETE /redfish/v1/Systems/1/Storage/1/Volumes/3`

Headers: No additional headers required.

Body:

None.

HTTP Status Code Returned: 400

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ResourceInUse",
    "Message": "The change to the requested resource failed because the resource
     is in use or in transition."
  }
}
```

Additional Context: None.

4.16 *Error Case: ResourceNotFound*

Summary: When a client requests a delete operation on a resource that cannot be found, the implementation will accept the request but then return a `ResourceNotFound` error.

Example: User tries to remove a volume that has already been deleted by another client or some other task.

Basic Course of Events:

1. Delete the volume from the Volumes collection.

Request: `DELETE /redfish/v1/Systems/1/Storage/1/Volumes/1`

Headers: No additional headers required.

Body: None.

HTTP Status Code Returned: 404

Headers: No additional headers required.

Body:

```
{
  "error": {
    "code": "Base.1.6.ResourceNotFound",
    "Message": "The requested resource of type Volume named 1 was not found."
  }
}
```

Additional Context: While the Redfish specification allows implementations to return a status code of 200 for this case, Swordfish recommends a 404 return code, to clarify a successful deletion from the detection of a prior deletion.