Stratisd D-Bus API Reference Manual*
Stratisd Version 2.0.0

Anne Mulhern

Version 0.11.0
Last modified: 11/08/2019

Contents

Asking Questions and Making Changes to this Document

This document can be foundin the stratis-docs repo, and is written using LyX 2.3.0. Please ask any questions by opening an issue, and propose changes as pull requests.

1 Introduction

This document describes the D-Bus API for the Stratis daemon. The D-Bus API constitutes the only public API of the Stratis daemon at the time of this writing. The public methods of the daemon's engine do not constitute part of the public API. The D-Bus API is a thin layer which receives messages on the D-Bus, processes them, transmits them to the Stratis engine, receives the results from the engine, and transmits a response on the D-Bus.

2 Overview

2.1 Technical Notes

2.1.1 Emulating an Option type

The D-Bus specification reserves a signature code for a "maybe" or "option" type, but this type is not available[1]. Nonetheless, it is desirable to have a way of representing a "don't care" condition when invoking D-Bus methods. Certain methods accept a pair argument that mimics such a type. The first item is a boolean. If the value is false, then the pair represents None. If the value is true, then the pair represents Some(x) where x is the second item of the pair.

3 Standard Interfaces and Methods

3.1 org.freedesktop.dbus.ObjectManager interface

The top level D-Bus object implements the org.freedesktop.dbus.ObjectManager interface. This interface defines the GetManagedObjects() method, which returns a view of the objects that the D-Bus layer has in its tree. This view constitutes a summary of the state of the pools, devices, and filesystems that Stratis manages. The objects are identified primarily by their D-Bus object paths. However, depending on the interface which an object supports it may also support certain identifying properties. For example, all objects that represent pools support an interface which implements a Name and a Uuid property which may be used by the client to identify a pool.

In invoking methods, or obtaining the values of properties, the client must identify existing objects by means of their object paths, either implicitly by invoking a method on an object constructed from an object path, or explicitly, by passing object paths as arguments to a method. For example, the API's DestroyPool() method requires two object paths:

- the object path of the Manager object, on which to invoke the DestroyPool method
- the object path of the pool object, which is passed as an argument to the DestroyPool method

It is expected that the client will identify the object path of the pool object by locating it using its Name or Uuid property.

3.2 org.freedesktop.dbus.Introspectable

All objects support the org.freedesktop.dbus.Introspectable interface. This interface has one method, Introspect(), which returns an XML description of the object's interfaces, methods, and properties.

4 Stratisd Interfaces and Methods

Each kind of object implements an identifying interface, i.e., pools implement a pool interface, filesystems implement a filesystem interface, etc. There is a top level manager interface that implements management tasks. The following is a description of the interfaces and their methods and properties. Some more detailed information can be obtained from the introspection information that can be queried for each object, including the signatures of all methods and properties. All methods return a return code and an accompanying message, with type signature "qs", and may also return data. If a method does return data, then the data is the first element in a triple, followed by the return code and message. The data may be a container type such as a struct or an array. Otherwise, if the method returns no data, the returned value is just a pair of the return code and message.

[1] `https://dbus.freedesktop.org/doc/dbus-specification.html`

It is intended that all operations that can be requested via the D-Bus API be idempotent. By this is meant that if an operation is requested repeatedly, and there are no intervening actions, then the operation should succeed every time, or it should fail every time. Since the return values may encode information about actions taken these may differ between different invocations. However if an error is returned that should always be the same error. Lack of idempotency as defined here constitutes a bug.

4.1 Manager interface

The Manager interface is the top level interface. It manages the creation and destruction of pools and also exports various global properties.

Methods

ConfigureSimulator This method is solely used to configure the simulator engine. Invoking it on the real engine is a no-op.

Arguments:

denominator the denominator of the fraction that determines the frequency of unusual occurences, generally failures. Signature: u.

Returns: Signature: qs.

CreatePool This method creates a single pool with the specified name and blockdevs.

Arguments:

name the name of the pool. Signature: s.

redundancy the redundancy specification for the pool. Signature: (bq). Note that redundancy is an Option argument.

devices device nodes of devices to form the pool. Signature: as.

Returns: Signature: (b(oao))qs.

1. True if a pool was created, otherwise false. Signature: b.
2. The result of the creation action, default values if no pool was created. Signature: (oao).
   (a) The object path of the created pool. Signature: o.
   (b) The object paths of all the block devices in the pool. Signature: ao.

DestroyPool This method destroys the specified pool.

Arguments:

pool_object_path the object path of the pool to destroy. Signature: o.

Returns: Signature: (bs)qs.

1. True if the pool was destroyed, otherwise false. Signature: b.
2. The UUID of the pool destroyed, or a default value if no action was taken. Signature: s.

Propeties

Version The current stratisd version. Signature: sqs.

4.2 pool interface

The pool interface manages the devices and filesystems within a pool.

Methods

AddCacheDevs This method allows additional cache devices to be added to the pool after it is created.

Arguments:

devices this argument has the same meaning as the device argument of CreatePool. Signature: as.

Returns: Signature: (bao)qs.

1. True if any cache devices were added, otherwise false. Signature: b.
2. The object paths of the newly added cache devices. Signature: ao.

AddDataDevs This method allows additional data devices to be added to the pool after it is created.

Arguments:

devices this argument has the same meaning as the device argument of CreatePool. Signature: as.

Returns: Signature: (bao)qs.

1. True if any data devices were added, otherwise false. Signature: b.
2. The object paths of the newly added data devices. Signature: ao.

CreateFilesystems This method allows creating multiple filesystems, specified by name, on a pool.

Arguments:

specs The specification for each filesystem to be created, currently just a name. Signature: as.

Returns Signature: (ba(os))qs.

1. True if any filesystem was added, otherwise false. Signature: b.
2. An array of object path/name pairs for each filesystem created. Signature: a(os).

DestroyFilesystems This method allows destroying multiple filesystems.

Arguments:

filesystems the object paths of the filesystems to be destroyed. Signature: ao.

Returns: Signature: (bas)qs.

1. True if any filesystems were destroyed, otherwise False. Signature: b.
2. An array of the UUIDs of the filesystems destroyed. Signature: as.

SetName This allows setting the name of the pool.

Arguments:

new_name The name to set. Signature: s.

Returns: Signature: (bs)qs.

1. True if an action was taken, otherwise false. If the new name is the same as the old name, then no action is considered to have been taken. Signature: b.
2. The UUID of the filesystem or a default UUID if no change was made. Signature: s.

SnapshotFilesystem This method allows a snapshot to be created from an existing filesystem

Arguments:

origin the object path of the source filesystem of the snapshot. Signature: o.

snapshot_name the name of the newly created snapshot. Signature: s.

Returns: Signature: (bo)qs.

1. True if a new snapshot was created, otherwise false. Signature: b.
2. The object path of the newly created snapshot. Signature: o.

Properties

Name The name of the pool. Signature: s.

Uuid The UUID of the pool. This property is constant. Signature: s.

4.3 filesystem interface

The fileystem interface manages the properties of individual filesystems.

Methods

SetName This allows setting the name of the filesystem.

Arguments:

name The name to set. Signature: s.

Returns: Signature: (bs)qs.

1. True if an action was taken, otherwise false. If the new name is the same as the old name, then no action is considered to have been taken. Signature: b.
2. The UUID of the filesystem, or a default UUID if no change was made. Signature: s.

Properties

Devnode The device node of the filesystem's corresponding thin device. This property is constant. Signature: s.

Name The name of the filesystem. Signature: s.

Pool The object path of the parent pool. This property is constant. Signature: o.

Uuid The UUID of the filesystem. This property is constant. Signature: s.

4.4 blockdev interface

The blockdev interface manages the properties of individual blockdevs.

Methods

SetUserInfo This allows setting identifying information for the blockdev by the user.

Arguments:

id The free-form information to set. Signature: (bs).

Returns: Signature: (bs)qs.

1. True if an action was taken, otherwise false. If the new information is the same as the old information, then no action is considered to have been taken. Signature: b.
2. The UUID of the blockdev, or a default UUID if no change was made. Signature: s.

Properties

Devnode The device node of the blockdev. This property is constant. Signature: s.

HardwareInfo The hardware-derived ID for this blockdev. May be empty. This property is constant. Signature: (bs).

InitializationTime Time that Stratis initialized this blockdev, in ISO 8601 text format (UTC). Signature: t.

Pool The object path of the parent pool. This property is constant. Signature: o.

Tier The tier the blockdev is in, either Data(0) or Cache (1). This property is constant. Signature: q.

UserInfo The user-defined string associated with this blockdev. May be empty. Signature: (bs).

Uuid The UUID of the blockdev. This property is constant. Signature: s.

4.5 FetchProperties interface

The FetchProperties interface is a generic interface supported by filesystems, pools, and blockdevs. It supplies two methods, which have the same return signature: a{s(bv)}. The return value is a table mapping string keys to properties of a filesystem, pool, or blockdev. Each value is a tuple of a boolean and a variant. The boolean indicates whether the second element of the pair represents the value requested or not. If the first element of the tuple is false, the second element is a string containing an error message indicating the cause of the failure to obtain the value.

Methods

GetAllProperties This method allows getting all properties of the particular pool, filesystem or blockdev.

Arguments: None.

GetProperties This method allows getting properties specified by a list of keys passed as an argument. If a key is unrecognized, there is no corresponding entry in the returned table.

Arguments:

properties: The list of string keys specifying the names of the properties requested. Signature: as.

Pools, filesystem, and blockdevs each support a different set of properties. The properties supported for each are:

blockdev

TotalPhysicalSize The total physical size of the blockdev in bytes. Signature: s.

filesystem

Used The bytes used by this filesystem. Signature: s.

pool

TotalPhysicalSize The total physical size of the pool in bytes. Signature: s.

TotalPhysicalUsed The total amount of physical space already used in the pool. Signature: s

5 Stratis interface versioning

5.1 Versioning scheme

stratisd implements a versioned D-Bus interface.

5.1.1 Interface major versions

The versioning scheme is currently included in the bus name (for example, org.storage.stratis2) and all of the interface names (org.storage.stratis2.pool, org.storage.stratis2.filesystem, and so on). The interface version number corresponds to the major version number of stratisd.

5.1.2 Interface minor versions

The proposed scheme also leaves room for supporting two interfaces at once. This would arise if enhancements need to be made in a minor release version to support new arguments in a method call or a new method altogether. For accessing minor release versions of the interface, two things are required. First, the minor version of stratisd must be at least that specified in the interface. Second, the interface must be accessed by appending the minor version number after a period to the end of the interface name. For example, a change to the filesystem interface at version 2.4.0 would be accessed through org.storage.stratis2.filesystem.4. This interface would provide the new feature or change to the interface, maintaining backwards compatibility through org.storage.stratis2.filesystem.

5.2 Interface breaking changes

When upgrading stratisd to a version with breaking changes, this should work automatically when installing from packages. If installing from source, there are some considerations when updating. Because stratisd uses the system bus in D-Bus, D-Bus will only allow stratis to bind to the new interface name if the D-Bus policy file allowing this is updated. When using packaging, this is typically installed at the location /usr/share/dbus-1/system.d/stratisd.conf on your filesystem or another path supported by D-Bus when searching for policy files. When stratis releases version 3.0.0, all instances of org.storage.stratis2 should be replaced with org.storage.stratis3 or else stratis will fail to communicate using the CLI or the programmatic API over D-Bus.