Stratisd D-Bus API Reference Manual[*]
Stratisd Version 1.0.0

Anne Mulhern

Version 0.9.0
Last modified: 09/27/2018

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. The engine can also generate D-Bus signals, when important properties change in value.

2 This API is ***NOT STABLE***

While this document covers the API as implemented in Stratis 1.0, the Stratis Team has not yet declared this a stable API. Other software should not rely on this API until the Stratis Team updates this document and declares the API stable and supported.

[*]This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

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`

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: nothing

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: the object path of the constructed pool and the object paths of the blockdevs added to the pool. Signature: (oa(o))qs.

DestroyPool This method destroys the specified pool.

Arguments:

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

Returns: true if it was necessary to take an action, otherwise false. If the pool has already been destroyed, no action is required. Signature: bqs.

Properties

Version The current stratisd version. Signature: s.

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: a list of object paths of the blockdevs added to the pool. Signature: a(o)qs.

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: a list of object paths of the blockdevs added to the pool. Signature: a(o)qs.

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 a list of pairs of object paths and names of filesystems created. Signature: a(os)qs.

DestroyFilesystems This method allows destroying multiple filesystems.

Arguments:

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

Returns: the names of the filesystems destroyed. Signature: asqs.

SetName This allows setting the name of the pool.

Arguments:

new_name The name to set. Signature: s.

Returns: true if the name was actually changed, otherwise false. Signature: bqs.

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: the object path of the newly created snapshot. Signature: oqs.

Properties

Name The name of the pool. Signature: s.

TotalPhysicalSize The total physical size of the pool in sectors. These sectors may be used by the pool for many purposes, so there are always fewer sectors than this that can be used to store user data. This is the largest physical size that can be meaningfully associated with a pool. Signature: s.

TotalPhysicalUsed All the sectors in use by the pool for any purpose whatsoever. These purposes include storage of user data as well as recording of pool metadata and other pool administration. This value may never exceed TotalPhysicalSize. Signature: s.

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

State The state of the pool. Possible values are documented in the design doc. Signature: q.

ExtendState The extend state of the pool. Possible values are documented in the design doc. Signature: q.

SpaceState The space state of the pool. Possible values are documented in the design doc. Signature: q.

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:** true if the name was actually changed, otherwise false. Signature: bqs.

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: s.
>
> **Returns:** true if the Id was actually changed, otherwise false. Signature: bqs.

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: s.

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

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

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

State The current state of the blockdev. These are defined in section 10.2.1 ("Layer 0: Blockdevs") of the Design Doc. Examples include: "Missing", "In-use", "Not-in-use", and "Bad". Signature: q.

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: s.

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