NAME

org.freedesktop.import1 − The D−Bus interface of systemd−importd

INTRODUCTION

systemd-importd.service(8) is a system service which may be used to import, export and download additional system images. These images can be used by tools such as **systemd-nspawn**(1) to run local containers. The service is used as the backend for **machinectl pull−raw**, **machinectl pull−tar** and related commands. This page describes the D−Bus interface.

Note that **systemd-importd.service**(8) is mostly a small companion service for **systemd-machined.service**(8). Many operations to manipulate local container and VM images are hence available via the **systemd−machined** D−Bus API, c.f. **org.freedesktop.machine1**(5).

THE MANAGER OBJECT

The service exposes the following interfaces on the Manager object on the bus:

```
node /org/freedesktop/import1 {
  interface org.freedesktop.import1.Manager {
    methods:
      ImportTar(in  h fd,
                in  s local_name,
                in  b force,
                in  b read_only,
                out u transfer_id,
                out o transfer_path);
      ImportRaw(in  h fd,
                in  s local_name,
                in  b force,
                in  b read_only,
                out u transfer_id,
                out o transfer_path);
      ImportFileSystem(in  h fd,
                       in  s local_name,
                       in  b force,
                       in  b read_only,
                       out u transfer_id,
                       out o transfer_path);
      ExportTar(in  s local_name,
                in  h fd,
                in  s format,
                out u transfer_id,
                out o transfer_path);
      ExportRaw(in  s local_name,
                in  h fd,
                in  s format,
                out u transfer_id,
                out o transfer_path);
      PullTar(in  s url,
              in  s local_name,
              in  s verify_mode,
              in  b force,
              out u transfer_id,
              out o transfer_path);
      PullRaw(in  s url,
              in  s local_name,
              in  s verify_mode,
                in  b force,
                out u transfer_id,
                out o transfer_path);
      ListTransfers(out a(usssdo) transfers);
      CancelTransfer(in  u transfer_id);
    signals:
      TransferNew(u transfer_id,
                  o transfer_path);
      TransferRemoved(u transfer_id,
                      o transfer_path,
                      s result);
  };
  interface org.freedesktop.DBus.Peer { ... };
  interface org.freedesktop.DBus.Introspectable { ... };
  interface org.freedesktop.DBus.Properties { ... };
};
```

Methods

ImportTar() and **ImportRaw()** import a system image and place it into /var/lib/machines/. The first argument should be a file descriptor (opened for reading) referring to the tar or raw file to import. It should reference a file on disk, a pipe or a socket. When **ImportTar()** is used the file descriptor should refer to a tar file, optionally compressed with **gzip**(1), **bzip2**(1), or **xz**(1). **systemd−importd** will detect the used compression scheme (if any) automatically. When **ImportRaw()** is used the file descriptor should refer to a raw or qcow2 disk image containing an MBR or GPT disk label, also optionally compressed with gzip, bzip2 or xz. In either case, if the file is specified as a file descriptor on disk, progress information is generated for the import operation (as in that case we know the total size on disk). If a socket or pipe is specified, progress information is not available. The file descriptor argument is followed by a local name for the image. This should be a name suitable as a hostname and will be used to name the imported image below /var/lib/machines/. A tar import is placed as a directory tree or a **btrfs**(8) subvolume below /var/lib/machines/ under the specified name with no suffix appended. A raw import is placed as a file in /var/lib/machines/ with the .raw suffix appended. If the **force** argument is true, any pre−existing image with the same name is removed before starting the operation. Otherwise, the operation fails if an image with the same name already exists. Finally, the **read_only** argument controls whether to create a writable or read−only image. Both methods return immediately after starting the import, with the import transfer ongoing. They return a pair of transfer identifier and object path, which may be used to retrieve progress information about the transfer or to cancel it. The transfer identifier is a simple numeric identifier, the object path references an org.freedesktop.import1.Transfer object, see below. Listen for a **TransferRemoved** signal for the transfer ID in order to detect when a transfer is complete. The returned transfer object is useful to determine the current progress or log output of the ongoing import operation.

ExportTar() and **ExportRaw()** implement the reverse operation, and may be used to export a system image in order to place it in a tar or raw image. They take the machine name to export as their first parameter, followed by a file descriptor (opened for writing) where the tar or raw file will be written. It may either reference a file on disk or a pipe/socket. The third argument specifies in which compression format to write the image. It takes one of "uncompressed", "xz", "bzip2" or "gzip", depending on which compression scheme is required. The image written to the specified file descriptor will be a tar file in case of **ExportTar()** or a raw disk image in case of **ExportRaw()**. Note that currently raw disk images may not be exported as tar files, and vice versa. This restriction might be lifted eventually. The method returns a transfer identifier and object path for cancelling or tracking the export operation, similar to **ImportTar()** or **ImportRaw()** as described above.

PullTar() and **PullRaw()** may be used to download, verify and import a system image from a URL. They take an URL argument which should point to a tar or raw file on the "http://" or "https://" protocols, possibly compressed with xz, bzip2 or gzip. The second argument is a local name for the image. It should be suitable as a hostname, similar to the matching argument of the **ImportTar()** and **ImportRaw()** methods above. The third argument indicates the verification mode for the image. It may be one of "no", "checksum", "signature". "no" turns off any kind of verification of the image; "checksum" looks for a SHA256SUM file next to the downloaded image and verifies any SHA256 hash value in that file against the image; "signature" does the same but also tries to authenticate the SHA256SUM file via **gpg**(8) first. The last argument indicates whether to replace a possibly pre−existing image with the same local name (if "true"), or whether to fail (if "false"). Like the import and export calls above, these calls return a pair of transfer identifier and object path for the ongoing download.

ListTransfers() returns a list of ongoing import, export or download operations as created with the six calls described above. It returns an array of structures which consist of the numeric transfer identifier, a string indicating the operation (one of "import−tar", "import−raw", "export−tar", "export−raw", "pull−tar" or "pull−raw"), a string describing the remote file (in case of download operations this is the source URL, in case of import/export operations this is a short string describing the file descriptor passed in), a string with the local machine image name, a progress value between 0.0 (for 0%) and 1.0 (for 100%), as well as the transfer object path.

CancelTransfer() may be used to cancel an ongoing import, export or download operation. Simply specify the transfer identifier to cancel the ongoing operation.

Signals

The **TransferNew** signal is generated each time a new transfer is started with the import, export or download calls described above. It carries the transfer ID and object path that have just been created.

The **TransferRemoved** signal is sent each time a transfer finishes, is canceled or fails. It also carries the transfer ID and object path, followed by a string indicating the result of the operation, which is one of "done" (on success), "canceled" or "failed".

THE TRANSFER OBJECT

```
node /org/freedesktop/import1/transfer/_1 {
  interface org.freedesktop.import1.Transfer {
    methods:
      Cancel();
    signals:
      LogMessage(u priority,
                 s line);
    properties:
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly u Id = ...;
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Local = '...';
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Remote = '...';
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Type = '...';
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Verify = '...';
      @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
      readonly d Progress = ...;
  };
  interface org.freedesktop.DBus.Peer { ... };
  interface org.freedesktop.DBus.Introspectable { ... };
  interface org.freedesktop.DBus.Properties { ... };
};
```

Methods

The **Cancel()** method may be used to cancel the transfer. It takes no parameters. This method is pretty much equivalent to the **CancelTransfer()** method on the Manager interface (see above), but is exposed on the Transfer object itself instead of taking a transfer ID.

Properties

The *Id* property exposes the numeric transfer ID of the transfer object.

The *Local*, *Remote* and *Type* properties expose the local container name of this transfer, the remote source (in case of download: the URL, in case of import/export: a string describing the file descriptor passed in), and the type of operation (see the Manager's **ListTransfer()** method above for an explanation of the possible values).

The *Verify* property exposes the selected verification setting and is only defined for download operations (see above).

The *Progress* property exposes the current progress of the transfer as a value between 0.0 and 1.0. To show a progress bar on screen we recommend to query this value in regular intervals, for example every 500 ms or so.

EXAMPLES

Example 1. Introspect org.freedesktop.import1.Manager on the bus

```
$ gdbus introspect −−system \
  −−dest org.freedesktop.import1 \
  −−object−path /org/freedesktop/import1
```

Example 2. Introspect org.freedesktop.import1.Transfer on the bus

```
$ gdbus introspect −−system \
  −−dest org.freedesktop.import1 \
  −−object−path /org/freedesktop/import1/transfer/_1
```

VERSIONING

These D−Bus interfaces follow **the usual interface versioning guidelines**[1].

NOTES

1. the usual interface versioning guidelines
   http://0pointer.de/blog/projects/versioning-dbus.html