

Datamover 10.11

Table of Contents

Introduction and workflow	1
Installation	3
Configuration	4
Configuration options	4
Robustness with respect to program restarts	8
Robustness with respect to clock mismatch	8
Copy engine	8
Dealing with failures when retrying does not help	9
Notifications	9
"High water" mark protection against disk capacity being exceeded	10
Ssh tunneling mode	10
Restricted remote target environment using <code>rsync</code>	12
Hybrid rsync server / ssh tunneling mode	12
A simple rsync server setup	13
Datamover setup without authorization.....	13
A rsync server setup with basic authorization.....	13
Datamover setup with authorization	14
Special features	14
Prefixing incoming data sets.....	14
Handshake (Data completion check).....	14
Local File Cleansing.....	15
Manual Intervention Handling	15
Local Data Transformation	16
Currently available transformers	16
TIFF Compressor	16
Extra local copy	17
Obtaining status	17
Obtaining outgoing target	18
Shutdown mode	18
Timeouts in checking for last modification time of incoming target.....	19

Introduction and workflow

✔ Words written in `monospace` are configuration directives.

Datamover is a program that takes care of moving (typically large to huge) file-based data produced by a data producer (e.g. a measurement device) to a (remote) central storage. Running in the background, it checks for new, incoming data periodically. Since the central storage is remote (i.e. requires a network to be in good working order), the copy process can run into trouble, that is: get

terminated or stuck. Datamover will take care of these problems to the extent possible and keep you informed if the problem persists.

Datamover utilizes 3 "targets" (where "target" denotes a directory on this or a remote host):

1. The `incoming-target` can be either local or on a remote file system where the data producer dumps its data. If it is local, it needs to be on the same file system as `buffer-dir`.
2. The `buffer-dir` is a directory on the local file system of the host that runs Datamover where data from `incoming-target` get moved to when they are found to be ready for moving.
3. The `outgoing-target` is supposed to be always on the remote file system where the data eventually get moved to. Only data from the `buffer-dir` are moved to the `outgoing-target` and only data that have been copied over successfully are then removed from `buffer-dir`.

The **workflow** is as follows:

1. The data producer writes data to `incoming-target`.
2. The *Mover of Incoming Data* monitors `incoming-target` for items (files or directories) that have not seen any updates in a given period of time (the `quiet-period`), are moved from `incoming-target` to `buffer-dir`. This can either be an operation moving one inode (if `treat-incoming-as-remote` is `false`) or a copy / delete cycle employing a copy engine. Optionally a script can be specified that is able to check by any custom criterion that may apply to your `incoming-target` whether the item is ready to be moved (see section [Handshake \(Data completion check\)](#) for details).
1. The *Local Processor* does all local operations on data in `buffer-dir`, like cleansing, data transformation or making an extra copy. It uses internal directory structure inside `buffer-dir` to do that.
2. The *Final Destination Mover* copies files or directories processed by *Local Processor* to `outgoing-target`, using a *copy engine*.
3. If an item has been successfully copied to the `outgoing-target`, it will be removed from the `buffer-dir`. A *mark file* `.MARKER_is_finished_<itemname>` will be created in the `outgoing-target` in order to signal that the copy process has been successfully finished.
4. If the copy operation failed, the *Final Destination Mover* sleeps for a while (the `failure-interval`) and retries the operation. The operation will be retried at a maximum `max-retries` times.

If the `incoming-target` directory does not reside on the same machine as the `buffer-dir` directory, you need to set `treat-incoming-as-remote` option. This will ensure that a regular data copy / delete cycle is used to move data from the incoming directory to the buffer directory instead of a simple in-filesystem move. It should be noted that one important reason for buffering is to avoid on overflow on the disk capacity of the data producer, keeping the data producer from getting stuck. Thus a reasonable setup needs to make sure that if data producer and Datamover run on different hosts, the network connection between them is very

reliable, preferably just a cross-over cable or a room network.

Installation

Datamover 10.11 has been tested on:

- Redhat Enterprise Linux 5.5 (x64)



SELinux

We have seen problems with Redhat Enterprise Linux 5 when *SELinux* was enabled: `rsync` was not able to perform copy operations but terminated with error messages about not being able to `stat` paths. We recommend you disable *SELinux* on hosts that are supposed to run Datamover. To check whether SELinux is disabled, use the `getenforce` command.

- Apple MacOS 10.5 (x64)
- Microsoft Windows 7 Professional (x64 with 32bit JRE)

Earlier versions of the software have been tested on:

- Redhat Enterprise Linux 5.2 (x64), openSUSE Linux 10.2 and 10.3 (x86)
- Sun OpenSolaris 2008.05 (x64)
- Apple MacOS 10.4 (x86)
- FreeBSD 6.2-STABLE (x86)
- Microsoft Windows XP Professional (x86)

Actually it should run on any Posix compliant operating system that has a Java Runtime Environment 5.0 or later and `rsync`, `ssh` and `ln` binaries on it.

As a pre-requisite you need to have installed a Java Runtime Environment 5.0 or later (look for example at the [Oracle](#) website). **Note:** In the following we have to distinguish between Unix and Windows in some places. Linux and MacOS X qualify as Unix.

- Download the distribution zip file (for link see below in the download section) and unzip it at a place convenient to you.
- Look at `etc/log.xml` and see whether the email settings suit your needs (it will work out of the box only on Unix machines with running SMTP server where email to root is forwarded to an admin account). Try it out by calling `datamover.sh test-notify`.
- Edit the `etc/service.properties` file and put in parameters that work for you. You can also specify the parameters in the command line. Note that you probably have to change the `incoming-target`, the `buffer-dir` and the `outgoing-target` settings since the default values are supposed to work only for testing and demonstration. It is safe to leave the other settings unchanged.
- Have a look at `etc/datamover.conf` and see whether that fits your configuration.
- Start the Datamover by calling either `datamover.sh` (Unix) or `datamover.bat` (Windows). On Unix, the program detaches from the console, on Windows, you will have to keep the command shell window open to keep the application running. If you want to use Datamover as a service on Windows, you may

have want to have a look at [Running Java Applications as a Windows Service](#), though please note that we didn't test this.

Configuration

All settings of the Datamover can be specified either on the command line (for a list of all options, see below) or by a line in the file `service.properties`, which is located in the `etc` subdirectory of the distribution. Each line of the configuration file should have the "name = value" format, e.g. "outgoing-target = data/destination". If the same option is specified in two places, the setting on the command line will always take precedence.

There are two environment configurations that cannot be specified this way, which are `JAVA_HOME`, the home directory of the JRE, and `JAVA_OPTS`, the additional parameters provided to the JRE. For Unix/Linux these can be specified in `etc/datamover.conf`. For Windows, they have to be added to `datamover.bat` directly.

Additional parameters of the optional data transformation can be specified only in `service.properties` file.

Configuration options

Options in `service.properties`

```
#
# Incoming target
#

# The directory where the data producer writes data items to.
# Syntax: incoming-target = [[<user-name>@]<host-name>:[<rsync-module>:]]<dir-path>
# * If you set a <host-name> and a <dir-path> it will be assumed that the target is a
#   directory on a
#   remote host that has an accessible ssh server and that this host is allowed to
#   connect to.
# * If you set a <host-name>, an <rsync-module> and a <dir-path>, it will be assumed
#   that that the
#   target is a directory on a remote host that has an accessible ssh server and an
#   accessible rsync
#   server that this host is allowed to connect to.
# Note that setting the <rsync-module> still required an ssh connections for some
# operations, so
# setting this parameter just means that the bulk transfer is using the rsync server.
incoming-target = data/incoming

# The string prepended to incoming data sets. '%t' will be replaced with the current
# time.
prefix-for-incoming = %t_

# If set to true, the initial test for accessibility of the incoming store will be
# skipped.
skip-accessibility-test-on-incoming = false

# When set to <true>, then the incoming directory will be treated as a mounted remote
# directory.
# This is only relevant when the incoming-target does not contain a <host-name>
# (which makes it
```

```

# explicit that the incoming target is remote).
#treat-incoming-as-remote = <true or false>

#
# Buffer
#

# The local directory to store the paths to be transfered temporarily
buffer-dir = data/buffer

# If free disk space goes below value defined here, a notification email will be
sent.
# Value must be specified in kilobytes (1048576 = 1024 * 1024 = 1GB).
# Comment this out or set it to a negative value in order to disable the high-water
mark feature
# for the buffer.
buffer-dir-highwater-mark = 1048576

#
# Outgoing target
#

# The remote target to move the data to.
# Syntax: outgoing-target = [[[<user-name>@]<host-name>:[<rsync-module>:]]<dir-path>
# * If you set a <host-name> and a <dir-path> it will be assumed that the target is a
directory on a
# remote host that has an accessible ssh server and that this host is allowed to
connect to.
# * If you set a <host-name>, an <rsync-module> and a <dir-path>, it will be assumed
that that the
# target is a directory on a remote host that has an accessible ssh server and an
accessible rsync
# server that this host is allowed to connect to.
# Note that setting the <rsync-module> still required an ssh connections for some
operations, so
# setting this parameter just means that the bulk transfer is using the rsync server.
outgoing-target = data/outgoing

# If free disk space goes below value defined here, a notification email will be
sent.
# Value must be specified in kilobytes (1048576 = 1024 * 1024 = 1GB).
# Comment this out or set it to a negative value in order to disable the high-water
mark feature
# for the outgoing directory.
outgoing-target-highwater-mark = 1048576

# If set to true, the initial test for accessibility of the outgoing store will be
skipped.
skip-accessibility-test-on-outgoing = false

#
# Optional feature: handshake on incoming data
#

# Path to the script file that will be executed to check whether an incoming data
item is already
# complete or not.
#data-completed-script = <path to script>

# Timeout (in seconds) for the data-completed-script. If the script exceeds this
timeout, it will
# be killed and an error is reported.
#data-completed-script-timeout = <timeout in seconds>

#
# Optional feature: manual intervention handling

```

```

#

# The local directory to store paths that need manual intervention (mandatory, but
only used when
# manual-intervention-regex is set
manual-intervention-dir = data/manual_intervention

# Regular expression of paths that need manual intervention, default prefix-for-
incoming
# corresponds to regex '[0-9]{14}_'
# Set this to enable manual intervention checking.
#manual-intervention-regex = <regex of paths that need manual intervention, default
prefix-for-incoming corresponds to '[0-9]{14}_'>

#
# Optional feature: The script which should be called when a file/directory has been
# successfully transfered to the outgoing directory.
# The script will be called with one parameter - the transfered
item name.
#
#transfer-finished-executable = <path to the script which will be invoked on
successful transfer completion>

#
# Optional feature: data cleansing
#

# The regular expression of paths that should be removed before moving an item to
outgoing
#cleansing-regex = <regex>

#
# Optional feature: creation of an extra (immutable) copy on the Datamover server for
processing
#

# The (local) directory in which an extra copy of each incoming data item will be
created.
# The copy needs to be treated immutable, i.e. it may be read and deleted, but not
changed!
#extra-copy-dir = <path>

#
# Optional feature: data transformation
#

# The name of the class (together with the list of packages this class belongs to)
# with implementation of data transformation that will be performed in the buffer.
#transformator.class = <class name>

# Additional transformator properties:
#transformator.<property 1> = <property value>
#transformator.<property 2> = <property value>
#...

#
# Timing parameters
#

# The time period (in seconds) that an incoming data item needs to be 'quiet' (i.e.
no write
# access is sensed on it) before moving it to the buffer will start.
#quiet-period = <time period in seconds>

# Time interval (in seconds) between two checks for incoming data.
#check-interval = <time interval in seconds>

```

```

# Time interval (in seconds) between two checks on the buffer directory.
# (You will probably not want to change this.)
#check-interval-internal = <time interval in seconds>

# Time period (in seconds) without any write activity on the target before a copy
process is
# considered stalled.
#inactivity-period = <time period before a copy process is considered stalled in
seconds>

# Time period (in seconds) to wait after a failure has occurred before the operation
is re-tried.
#failure-interval = <time period in seconds>

# Maximal number of re-tries of a failed operation before giving up on it.
#max-retries = <maximal number of retries>

#
# Explicitly set executables (leave blank to let Datamover find them itself)
#

# The path to the rsync executable. Only required if the first occurrence of rsync in
the PATH is
# not what you want to use for the Datamover.
#rsync-executable = <path to rsync>

# If set to true, rsync is called in such a way that target files that already exist
are
# overwritten rather than appended to.
#rsync-overwrite = <true or false, default is false>

# May be used to explicitly add parameters to the rsync command line.
#extra-rsync-params = <coma-separated list of additional params, e.g. --no-owner,--
no-group>

# The path to the rsync executable on the incoming host.
# Only used when ssh tunneling mode is used for the incoming target.
# Only required if the first occurrence of rsync in the PATH on the incoming host is
not what you
# want to use for the Datamover.
#incoming-host-rsync-executable = <path to rsync>

# The path to the rsync executable on the outgoing host.
# Only used when ssh tunneling mode is used for the outgoing target.
# Only required if the first occurrence of rsync in the PATH on the outgoing host is
not what you
# want to use for the Datamover.
#outgoing-host-rsync-executable = <path to rsync>

# Path to the 'lastchanged' executable of Datamover on the remote incoming host
# Specify only when using an ssh tunnel or an rsync server for copying the incoming
data.
#incoming-host-lastchanged-executable = <path of 'lastchanged' executable>

# Path to the GNU find executable on the remote incoming host.
# Specify only when using an ssh tunnel or an rsync server for copying the incoming
data.
#incoming-host-find-executable = <path of 'find' executable>
#
# Path to the 'lastchanged' executable of Datamover on the remote outgoing host
# Specify only when using an ssh tunnel or an rsync server for copying the outgoing
data.
#outgoing-host-lastchanged-executable = <path of 'lastchanged' executable>

# Path to the GNU find executable on the remote outgoing host.

```

```
# Specify only when using an ssh tunnel or an rsync server for copying the outgoing
data.
#outgoing-host-find-executable = <path of 'find' executable>

# The path to the ln executable (for hard link creation). Only required if the first
occurrence of
# ln in the PATH is not what you want to Datamover to use.
#ln-executable = <path to ln>

# The path to the ssh executable (for SSH tunnels). Only required if the first
occurrence of
# ssh in the PATH is not what you want to Datamover to use.
#ssh-executable = <path to ssh>
```

Robustness with respect to program restarts

The directory-based communication has been preferred over a memory-based one, because it is more robust with respect to restarting the program. This is because with the directory-based approach all state is kept on the file system instead in memory. Thus restarting the program, or even the server, will restart an operation where it was terminated. Special care was taken to ensure, that after restarting the program it recovers properly, finishing all operations that were stopped in the middle. This is called a *recovery cycle* which is run automatically after program start.

In case an environment triggered exceptional condition occurs during the processing, a recovery cycle can be triggered without a restart by calling `datamover.sh recover`.

Robustness with respect to clock mismatch

When the `incoming-target` is located on a different host than the Datamover, there is the potential problem that the clocks of the two hosts may be not synchronized. In order to avoid this problem, Datamover is using an algorithm that ensures that this condition does not lead to premature transfer and deletion processes (which might even lead to data loss). To this end, the Datamover never compares times from the data producer and from the Datamover directly. Instead, the last modification time of an "item" (which may be a file or a directory) is compared to the last modification time of the same item at an earlier time, where the time difference that decides on when to compare last modification times is determined from the Datamover clock.

The same robust mechanism is used also for `ssh` tunneling mode and hybrid `rsync` server / `ssh` tunneling mode (see below). There is no requirement of clocks synchronization between the Datamover machine and the remote machine from which or to which data are moved.

Copy engine

The Datamover uses `rsync` as its copy engine. For Microsoft Windows, version

3.0.7 from Cygwin is packaged. For Unix/Linux, `rsync` needs to be installed in the system. The system requires version 2.6.0 or newer, but we recommend version 2.6.7 or newer. Various versions from 2.6.5 to 3.0.7 have been given a cursory test. However, most experience has been gathered with versions 2.6.8 and 3.0.6. Note that the executable of `rsync` to use can be specified using the `rsync-executable` configuration parameter.

Append vs. Overwrite

Note that by default Datamover uses the append mode for `rsync` v2.6.7 or newer. In this mode `rsync` first tries to append to an already existing file. If the assumption turns out to be wrong that the already existing part of the file on the destination was identical to the source, this will be detected during the final checksum calculation and the whole file will be retransmitted.

This behavior can be changed by providing the option `rsync-overwrite` either on the command line or in `service.properties`. If specified, an already existing file will be deleted and re-transmitted anew. This is the only mode available for `rsync` v2.6.6 or older.

Dealing with failures when retrying does not help

When an item can not be successfully copied to the `outgoing` even after the maximal number of retries permitted, its name will be put in the `.faulty_paths` file and a notification entry will be logged (which, depending on the log configuration, will be sent to an administrator per email).

This situation requires manual intervention by an administrator. When the problem is fixed, removing an item from the `.faulty_paths` file (or deleting the file altogether) will make the Datamover retry copying the item(s) to the remote site.

Note that the `.faulty_paths` file is deleted when the data mover starts up. Thus restarting the data mover has the same effect as deleting the `.faulty_paths` file.

Notifications

All log messages of category `NOTIFY` are meant to be sent out to an administrator in one way or another, because they need manual intervention due to a failure that doesn't go away by retrying the operation. The notification is configured in the file `etc/log.xml` by means of the `EMAIL` `log4j` appender. By default, the mail will be sent out to `root@localhost`, which is fine if you are on a Unix/Linux system with running SMTP server bound to port 25 and someone regularly looking at the email of `root` or if the email of `root` is forwarded to a regular user who acts as system administrator of the box. Otherwise (especially if it is a Windows box), the SMTP settings in `etc/log.xml` need to be adapted to your environment. It is recommended that you check the settings by triggering a `NOTIFY` log message. This

can be done by calling `datamover.sh test-notify`. Note that this will trigger a `NOTIFY` log message of level `INFO`, so you must not change the log level above `INFO` in `etc/log.xml` in order to obtain an email.

"High water" mark protection against disk capacity being exceeded

For the `buffer-dir` resp. `outgoing-target`, it is possible to specify a so called *high-water mark*. A *high-water mark* is the lowest level of free disk space reached by a given directory. Once the *high-water mark* is reached (the available free disk space lies below the specified *high-water mark*), the administrator is notified via email and Datamover stops moving files, waiting until sufficient disk space is available again (in this case the administrator is notified as well).

The *high-water mark* is specified in *kilobytes*. Negative values are not considered, meaning that the system is not watching free disk space.

Example for specifying a *high-water mark* in the `service.properties`:

```
buffer-dir = targets/buffer
# Value is specified in kilobytes (1048576 = 1024 * 1024 = 1GB).
buffer-dir-highwater-mark = 1048576
outgoing-target = targets/outgoing
outgoing-target-highwater-mark = 1048576
```

Ssh tunneling mode

The `incoming` and `outgoing` directories can be accessed by the Datamover via an `ssh` tunnel (using the according mechanism of `rsync`). This can be useful when using remote shares (NFS or CIFS) is not an option, e.g. when moving data via the internet. You can switch on `ssh` tunneling mode by specifying the host name of the `ssh` server in the `outgoing-target` just before the (remote) directory using ':' as separator. The same applies to the `incoming-target`. Note that it is possible to move data from one remote machine to the other using `ssh` tunneling for both ends. However, this may not be optimal with regard to performance.

By default, the Datamover will search in the `PATH` for `ssh`. If you need to use a special version of SSH you can provide the path of the executable via `ssh-executable` (as with `rsync-executable` for `rsync`). For Windows, a version of SSH from Cygwin is packaged with the Datamover.

It is important to note that the authentication needs to work *without* password or passphrase. This means that authorization needs to be done using an unencrypted private key or using mechanisms like OpenSSH's `ssh-agent` or Gentoo's `keychain`. For OpenSSH the simplest way is to create an SSH key pair without a passphrase, add the public key to the `~/.ssh/authorized_keys` file of the `ssh` server and write an appropriate section in the `ssh` client's (that is the machine running the Datamover) `~/.ssh/config` file that makes `ssh` use this key when talking to the `ssh` server hosting the outgoing target.

In the default setup, it is required that the machine accessed through ssh provides:

- The `bash` shell
- The `df` Unix utility
- Only for an incoming store: The `rm` Unix utility
- Either the `lastchanged` utility
- Or the combination of the following tools:
 - The GNU version of `find`, which is part of the GNU `findutils`. By default it is checked if the `find` executable is accessible on the remote machine as either `find` or `gfind`. Be careful, because the command executed through the `ssh` tunnel does not have the usual `PATH` variable of the remote machine set. Instead it sees the `PATH` configured by `ssh`. In case the `find` executable can not be found automatically, the path can be configured explicitly using the Datamover `incoming-host-find-executable` or `outgoing-host-find-executable` configuration options.
 - The Unix `sort` utility
 - The Unix `head` utility

If you configure `skip-accessibility-test-on-xxx = true`, (`xxx`: incoming or outgoing) then you *must* configure the remote path to either `lastchanged` or `find`.

The `lastchanged` utility

The C source code of the `lastchanged` binary can be found in the Datamover distribution in `datamover/src`. It has been tested on Linux, Solaris and Mac OS X, but should work on any Posix compliant system using a C99 compatible C compiler. For the most common platforms, you will find the binaries in `bin/lastchanged` of the distribution.

It needs to be available on the **remote host** that you want to use as either incoming or outgoing target with Datamover in an `ssh` tunneling mode. If you need it on a platform where the binary is not available, compile it on the target host with a command similar to

```
# gcc -O3 -Wall lastchanged.c -o lastchanged
```

and put it into the path of the remote user that Datamover connects as, e.g. `/usr/bin`.



Establishing an SSH tunnel from a Windows client

As Datamover uses OpenSSH also on Windows, you need to have the private SSH key of the tunnel in OpenSSH format also on the Windows client machine. The private key needs to be copied into the sub-directory `bin/home/<username>/.ssh/id_rsa` relative to the Datamover installation directory. The first time you run `datamover.bat`, this directory will be created for you automatically if it doesn't exist. As a second requirement, you will need an appropriate entry for the target host in `home/<username>/.ssh/known_hosts`. The simplest way to create this entry is by running `"bin\win\ssh.exe user@host"` from `cmd.exe` in the Datamover installation directory. This will also allow you to check whether your SSH key works: If you can get into the target machine without you need to type anything, then you should be set to use the tunnel with Datamover.

Restricted remote target environment using `rsssh`

It is possible to use Datamover in a restricted target environment which only allows download or upload of data with Datamover, but no shell access using the utility `rssh`. To make `rssh` work with Datamover, a patch is needed that can be downloaded from [the Datamover download page](#).

On the server side, `rssh` needs to be used as the shell of the remote user that Datamover logs in as and the `rssh` configuration file (usually `/etc/rssh.conf`) needs to contain the line

```
allowdmover
```

The Datamover configuration `service.properties` needs to contain the following lines (where the paths have to match your system configuration and `outgoing` may have to be replaced with `incoming`):

```
skip-accessibility-test-on-outgoing = true
outgoing-host-lastchanged-executable = /usr/bin/lastchanged
outgoing-host-rsync-executable = /usr/bin/rsync
```

Note that in the restricted environment the paths have to be specified explicitly as automatic detection of paths will not work, and that the `lastchanged` executable needs to be available on the remote host as the combo of `bash`, `find`, `sort` and `head` will not work in this environment.



Binaries for `rssh`

For some platforms (like Redhat Enterprise Linux 5), we provide binaries of `rssh` on [the Datamover download page](#). Note that these packages also contains the `lastchanged` utility.

Hybrid `rsync` server / `ssh` tunneling mode

`ssh` tunneling mode is easy to setup and in general a secure way to move data. However, it requires all data that is moved to be encrypted and decrypted. Depending on the size of the data sets you are moving and the CPU power of the machines that may not be acceptable with regard to performance. For these cases, a hybrid mode can be used where the bulk of the data is transferred using a remote `rsync` server and only for some commands that need to be executed remotely the `ssh` tunnel is used.



Security consideration

Running an `rsync` server with writable `rsync` modules is not recommended in an insecure network environment and is a particularly bad idea over the internet. Use this feature with care and only when you really need it and know what you are doing.

A simple rsync server setup

It is suggested that you first setup and test the `ssh` tunneling mode and then, when this works, add the `rsync` configuration for the bulk transfers. For this to work, you need to have setup an `rsync` server on the standard port (873) on the outgoing machine that accepts connections from the machine running the Datamover. For this to work you usually need to run the `rsync` as root. A simple approach (*without any authorization or host restriction, so be careful*) is:

rsyncd.conf

```
log file = /data/rsyncd.log

[datagrave]
    path = /data/stuff
    use chroot = true
    read only = false
```

The directory `/data/stuff` needs to exist and needs to be writable by user `nobody`.

Datamover setup without authorization

Add the `rsync` module name in the `outgoing-target` (or `incoming-target`) between the server name and the directory on the server, separated by `':'`. So you replace e.g. (using the same directory as in the `rsyncd.conf` above):

```
outgoing-target = datahost:/data/stuff
```

by:

```
outgoing-target = datahost:datagrave:/data/stuff
```

Note that it is important that the directory (in the example `/data/stuff`) refers to the same location on the filesystem of the server than the `rsync` module (which is `datagrave` in the example given). Datamover cannot check that this is the case, but if it isn't, you will find that Datamover will terminate all copy processes because it won't see any write progress of the copied target on the server.

A rsync server setup with basic authorization

The `rsync` server can be configured to require a "secret" (or password) from the client before allowing access to an `rsync` module. An extension of the `rsyncd.conf` configuration file above which supports authorization reads:

rsyncd.conf with authorization

```
log file = /data/rsyncd.log

[datagrave]
    path = /data/stuff
    use chroot = true
    read only = false
    auth users = dmover1,dmover2
```

```
secrets file = rsyncd.passwd
```


where `dmover1` and `dmover2` are accounts on the client side that should be allowed access. The account name is determined by the operating system user that runs Datamover. The file `rsyncd.passwd` must not be readable by any other user than the one running the `rsync` server and is expected to contain lines like

```
dmover1:passwd
```

where `passwd` is the password in clear text.

Datamover setup with authorization

Compared to the setup described in [Datamover setup without authorization](#), there is only one configuration change needed in order to use the Datamover with an `rsync` server that requires a secret: provide that secret (in clear text) in a file `etc/rsync_incoming.passwd` for the incoming target or `etc/rsync_outgoing.passwd` for an outgoing target (both paths relative to the Datamover application directory).

 The password file must not be readable by any other operating system user than the user that is running Datamover. Otherwise `rsync` will refuse to use the file and consequently authorization will fail.

Special features

Prefixing incoming data sets

The option `prefix-for-incoming` allows setting a prefix for all data sets that the Datamover handles. The prefix is actually a prefix template in that the string `%t` will be replaced with the current time stamp in format `yyyyMMddHHmmss`. The default prefix set in the `service.properties` file is `%t_`.

This options serves two purposes:

1. Assume your measurement device from time to time can produce files or directories with the same name (e.g. the same barcode), then prefixing it with `%t` will make it unique.
2. You can have more than one Datamover running that points to the same outgoing directory and still know from which Datamover the data have been handled.

Handshake (Data completion check)


The option `data-completed-script` allows to specify a script which determines if an item in `incoming-target` is complete and ready to be moved. The script is executed after an incoming file or folder has not changed during the specified quiet period (see `quiet-period` option). The script gets one or two arguments:

1. Path to the incoming file or folder. If no remote host is specified it will be the absolute path, otherwise it may be a relative path, depending on what is specified for `incoming-target`.
2. The remote host if `incoming-target` contains a host specification (i.e. if `ssh` tunneling mode is used). Otherwise (`incoming-target` specifies a directory without host), this argument will not be provided.

The exit value of the script determines whether the incoming data is complete (`exit_value = 0`) or not (`exit_value != 0`). That is, the incoming data will not be moved before the script returns 0. The script path is relative to the application directory (i.e. the parent directory of `etc`).

With the option `data-completed-script-timeout` one can specify a time-out (in seconds) for the data completed script. If the script does not finish before the time out it will be killed (leading to a non-zero exit value).

If the `data-competed-script` script returns an `exit_value != 0` for three times in succession, an notification email will be sent out to the administrator, pointing out the problem.

 Using a handshake can increase robustness of the system as it removes the need for Datamover to "guess" when an incoming item is ready to be moved.

Local File Cleansing

Data cleansing describes the feature that the *Local Processor* removes certain files before moving a path item to the `buffer`. The rationale behind this feature is that sometimes you cannot prevent the data producer from creating certain files that you don't need but that would eat up quite some time and network bandwidth when moving the path entry to the central storage and thus you want to get rid of these files before moving the directory that contains them to the remote side.

The files that should be deleted are specified as regular expression with the `cleansing-regex` command line parameter. It is important to note that the regular expression needs to match the complete basename (that is the file name part of the path excluding the directory). Note also that cleansing does not delete directories, not even empty ones.

Example: If you want to remove all files with extension `.PNL` or `.STC`, you can specify as a regular expression: `'.+(\.PNL|\.STC)'`

Manual Intervention Handling

There may be situations that require manual intervention and where this situation should be already diagnosed on the Datamover host. A typical example of this situation is when the barcode reader of a microscope was unable to read the barcode of a screening plate. Then it may be important to keep the information about the order of the plates so that the missing barcode can be constructed afterwards from the order information.

Whenever the situation that requires manual intervention can be diagnosed by inspecting the path name, the manual intervention handling feature of Datamover can be used. It has two options, `manual-intervention-dir` and `manual-intervention-regex` and works like this: whenever a processed path entry matches the `manual-intervention-regex`, processing will be stopped and the entry will be moved to `manual-intervention-dir`.

Note that since manual intervention detection is handled by the *Local Processor* (and thus *after* the *Mover of Incoming Data*), there is a dependency between prefixing incoming data sets and manual intervention handling: The `manual-intervention-regex` needs to contain the `prefix-for-incoming`, where `'%t'` needs to be replaced by `'[0-9]{14}'`.

Every handled path, whether it matches the `manual-intervention-regex` or not, will be logged in `log/manual_intervention.txt`. The format of the file is (assuming the `manual-intervention-regex` is `'[0-9]{14}ttt.+'`):

```
2007-10-15 18:58:12,600: DEFAULT /some/folder/buffer/copy-
complete/20071015185807_normal_1 [created: 2007-10-15 18:58:01]
2007-10-15 18:58:22,609: ATTENTION /some/folder/buffer/copy-
complete/20071015185817_ttt_2 [created: 2007-10-15 18:58:05]
```

Local Data Transformation

If you would like to transform the data on the buffer server before sending them to the central storage (e.g. compress them to reduce the amount of data transferred over the network), you can use an optional step of transformation. Specify the Java class name (together with the list of packages this class belongs to) of an existing transformer using `transformator.class` option. The class(es) required by the specified transformation need to be in the classpath of the Datamover JVM. On Unix, any JAR file put into the `lib/` sub-directory will be picked up and put into the classpath automatically by the startup script. On Windows, you have to add the JAR file containing your transformation manually to `datamover.bat`.

Currently available transformers

TIFF Compressor

Performs compression of *TIFF* images. By default it uses `tiffcp` command line tool to compress images using *LZW* method. The command line tool needs to be in the operating system path.



There is additional requirement introduced to Datamover by recovery mechanism of this transformer - all files inside incoming directory should be grouped in directories. Putting a file directly into incoming directory will result in an error and the file will not be moved to the outgoing directory.

To use this transformer, it is sufficient to specify the name of its class as Datamover parameter:


```
transformator.class =  
ch.systemsx.cisd.datamover.transformation.TiffCompressorTransformator
```

All additional parameters of this transformator are optional with a reasonable default values that can be overridden in `service.properties` file like in this example:

```
...  
#  
# Optional feature: data transformation  
#  
  
# The name of the class (together with the list of packages this  
class belongs to)  
# with implementation of file transformation that will be performed  
in the buffer.  
transformator.class =  
ch.systemsx.cisd.datamover.transformation.TiffCompressorTransformator  
  
# Additional transformator properties:  
  
#- number of threads performing compression per processor  
# * 1 - lowest (default),  
# * depending on machine bigger value may improve performance  
increasing usage of CPU and HDD  
transformator.threads-per-processor = 2  
  
#- compression command used by tiff compressor; possible values:  
TIFFCP (default), CONVERT  
transformator.compression-command = CONVERT  
  
#- compression type used by compression command;  
# * for TIFFCP used as '-c' option argument, e.g.: lzw:2 (default),  
zip  
# * for CONVERT used as '-compress' option argument, e.g.: LZW  
(default), Zip  
transformator.compression-type = Zip  
...
```

Extra local copy

If you need to access the raw data from the buffer server and you do not want to transfer these data from the central storage, you can use the `extra-copy-dir` option and specify a directory on the buffer server, where the copy will be created. You can read the copy and delete it any time you wish. However you must not modify the content of the copied files, because the procedure will use hard links to save disk space, if the file system supports it!

Obtaining status

The status of the running Datamover daemon can be obtained by invoking the following command:

```
datamover.sh status
```

There is a variant of this command that is optimized for easy usage in scripts:

```
datamover.sh mstatus
```

It returns one of the following values:

- **DOWN:** Datamover is not running (exit value 3)
- **STALE:** Datamover is not running but there is still a stale `.pid` file (exit value 4)
- **SHUTDOWN:** Datamover is in the shutdown mode (invoked by `datamover.sh shutdown`, see below) (exit value 2)
- **IDLE:** Datamover is running but does currently nothing (exit value 0)
- **PROCESSING:** Datamover is running and processes currently some files (exit value 0)
- **ERROR:** Datamover is running and (at least) one of the 3 pipelines has an error (like e.g. the high water mark detection holding back moving data to the final destination (exit value 1)

Obtaining outgoing target

With the following command the outgoing target of the running datamover instance can be obtained:

```
datamover.sh target
```

This prints a value to standard output which can be used as a value for the option `-outgoing-target`. It has the following syntax:

```
[<host>: [<rsync_module>:]] <path>
```

Shutdown mode

The *shutdown mode* allows you to exit Datamover in a clean state. In this mode, the program finishes all currently ongoing transfers and processings, but doesn't start any new. When all transfer and processing operations are finished, Datamover exits. This mode can be triggered by calling `datamover.sh shutdown`. Note that `Ctrl+C` does not trigger the shutdown mode.

By calling `datamover.sh stop` you kill the Datamover process which promptly exits without starting any cleaning step or/and waiting for processes to be finished.

Following rules applies to the *shutdown mode*:

- Not all data listed in the `incoming` directory will be processed while shutting down, but only the one that is currently running at the time when *shutdown mode* is triggered. Datamover tries to cleanly exit the `incoming` directory processing as soon as possible.
- Once a `incoming` data has entered the pipeline, Datamover will not shutdown before this data has reached its final destination, the `outgoing` directory.

Timeouts in checking for last modification time of incoming target

If you get timeouts in checking the last modification time of an incoming target, i.e. if you get messages like

```
WARN [Mover of Incoming Data] OPERATION.DirectoryScanningTimerTask - Failed to
filter store items for processing: filter 'StoreItemFilterBank' threw exception
TimeoutException (message: "Call to method 'IFileStore.lastChanged(StoreItem,long)'
timed out (timeout=180000ms).") on item 'XXX'
```

in your logs, try to increase the parameter `check-interval`, as the time for the timeout is 3x the time of `check-interval`.