ARC::DataMove Reference Manual

Generated by Doxygen 1.3.5

Mon Jul 19 00:19:34 2004

Contents

Chapter 1

ARC::DataMove Hierarchical Index

1.1 ARC::DataMove Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

ARC::DataMove Class Index

2.1 ARC::DataMove Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

ARC::DataMove Class Documentation

3.1 DataBufferPar Class Reference

`#include <databufferpar.h>`

Public Member Functions

- operator bool (void)
- DataBufferPar (unsigned int size=65536, int blocks=3)
- DataBufferPar (CheckSum ∗cksum, unsigned int size=65536, int blocks=3)
- ∼DataBufferPar (void)
- bool set (CheckSum ∗cksum=NULL, unsigned int size=65536, int blocks=3)
- char ∗ operator[ ] (int n)
- bool for_read (int &handle, unsigned int &length, bool wait)
- bool is_read (int handle, unsigned int length, unsigned long long int offset)
- bool is_read (char ∗buf, unsigned int length, unsigned long long int offset)
- bool for_write (int &handle, unsigned int &length, unsigned long long int &offset, bool wait)
- bool is_written (int handle)
- bool is_written (char ∗buf)
- bool is_notwritten (int handle)
- bool is_notwritten (char ∗buf)
- void eof_read (bool v)
- void eof_write (bool v)
- void error_read (bool v)
- void error_write (bool v)
- bool eof_read (void)
- bool eof_write (void)
- bool error_read (void)
- bool error_write (void)
- bool error_transfer (void)
- bool error (void)
- bool wait (void)
- bool wait_used (void)
- bool checksum_valid (void)
- const CheckSum ∗ checksum_object (void)

- bool wait_eof_read (void)
- bool wait_eof_write (void)
- bool wait_eof (void)
- unsigned long long int eof_position (void) const
- unsigned int buffer_size (void)

Public Attributes

- DataSpeed speed

3.1.1 Detailed Description

This class represents set of buffers used during data transfer.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DataBufferPar::DataBufferPar (unsigned int *size* = 65536, int *blocks* = 3)

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

3.1.2.2 DataBufferPar::DataBufferPar (CheckSum ∗ *cksum*, unsigned int *size* = 65536, int *blocks* = 3)

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

cksum object which will compute checksum. Should not be destroyed till DataBufferPar itself.

3.1.2.3 DataBufferPar::∼DataBufferPar (void)

Destructor.

3.1.3 Member Function Documentation

3.1.3.1 unsigned int DataBufferPar::buffer_size (void)

Returns size of buffer in object. If not initialized then this number represents size of default buffer.

3.1.3.2 const CheckSum∗ DataBufferPar::checksum_object (void)

Returns CheckSum object specified in constructor.

3.1.3.3 bool DataBufferPar::checksum_valid (void)

Returns true if checksum was successfully computed.

3.1.3.4 unsigned long long int DataBufferPar::eof_position (void) const `[inline]`

Returns offset following last piece of data transfered.

3.1.3.5 bool DataBufferPar::eof_read (void)

Returns true if object was informed about end of transfer on ’read’ side.

3.1.3.6 void DataBufferPar::eof_read (bool *v*)

Informs object if there will be no more request for ’read’ buffers. v true if no more requests.

3.1.3.7 bool DataBufferPar::eof_write (void)

Returns true if object was informed about end of transfer on ’write’ side.

3.1.3.8 void DataBufferPar::eof_write (bool *v*)

Informs object if there will be no more request for ’write’ buffers. v true if no more requests.

3.1.3.9 bool DataBufferPar::error (void)

Returns true if object was informed about error or internal error occured.

3.1.3.10 bool DataBufferPar::error_read (void)

Returns true if object was informed about error on ’read’ side.

3.1.3.11 void DataBufferPar::error_read (bool *v*)

Informs object if error accured on ’read’ side.

Parameters:
v true if error.

3.1.3.12 bool DataBufferPar::error_transfer (void)

Returns true if eror occured inside object.

3.1.3.13 bool DataBufferPar::error_write (void)

Returns true if object was informed about error on ’write’ side.

3.1.3.14 void DataBufferPar::error_write (bool *v*)

Informs object if error accured on ’write’ side.

Parameters:

v true if error.

3.1.3.15 bool DataBufferPar::for_read (int & *handle*, unsigned int & *length*, bool *wait*)

Request buffer for READING INTO it.

Parameters:

handle returns buffer’s number.

length returns size of buffer

wait if true and there are no free buffers, method will wait for one. Returns true on success

3.1.3.16 bool DataBufferPar::for_write (int & *handle*, unsigned int & *length*, unsigned long long int & *offset*, bool *wait*)

Request buffer for WRITING FROM it.

Parameters:

handle returns buffer’s number.

length returns size of buffer

wait if true and there are no free buffers, method will wait for one.

3.1.3.17 bool DataBufferPar::is_notwritten (char ∗ *buf*)

Informs object that data was NOT written from buffer (and releases buffer).

Parameters:

buf - address of buffer

3.1.3.18 bool DataBufferPar::is_notwritten (int *handle*)

Informs object that data was NOT written from buffer (and releases buffer).

Parameters:

handle buffer’s number.

3.1.3.19 bool DataBufferPar::is_read (char ∗ *buf*, unsigned int *length*, unsigned long long int *offset*)

Informs object that data was read into buffer.

Parameters:

buf - address of buffer

length amount of data.

offset offset in stream, file, etc.

3.1.3.20 bool DataBufferPar::is_read (int *handle*, unsigned int *length*, unsigned long long int *offset*)

Informs object that data was read into buffer.

Parameters:

handle buffer's number.

length amount of data.

offset offset in stream, file, etc.

3.1.3.21 bool DataBufferPar::is_written (char ∗ *buf*)

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

3.1.3.22 bool DataBufferPar::is_written (int *handle*)

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

3.1.3.23 DataBufferPar::operator bool (void) `[inline]`

Check if DataBufferPar object is initialized.

3.1.3.24]

char∗ DataBufferPar::operator[ ] (int *n*)

Direct access to buffer by number.

3.1.3.25 bool DataBufferPar::set (CheckSum ∗ *cksum* = NULL, unsigned int *size* = 65536, int *blocks* = 3)

Reinitialize buffers with different parameters.

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

cksum object which will compute checksum. Should not be destroyed till DataBufferPar itself.

3.1.3.26 bool DataBufferPar::wait (void)

Wait (max 60 sec.) till any action happens in object. Returns true if action is eof on any side.

3.1.3.27 bool DataBufferPar::wait_eof (void)

Wait till end of transfer happens on any side.

3.1.3.28 bool DataBufferPar::wait_eof_read (void)

Wait till end of transfer happens on ’read’ side.

3.1.3.29 bool DataBufferPar::wait_eof_write (void)

Wait till end of transfer happens on ’write’ side.

3.1.3.30 bool DataBufferPar::wait_used (void)

Wait till there are no more used buffers left in object.

3.1.4 Member Data Documentation

3.1.4.1 DataSpeed DataBufferPar::speed

This object controls transfer speed.

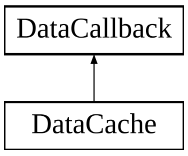
The documentation for this class was generated from the following file:

- databufferpar.h

3.2 DataCache Class Reference

`#include <datacache.h>`

Inheritance diagram for DataCache::



Public Member Functions

- DataCache (void)
- DataCache (const char ∗cache_path, const char ∗cache_data_path, const char ∗cache_link_path, const char ∗id, uid_t cache_uid, gid_t cache_gid)
- DataCache (const DataCache &cache)
- ∼DataCache (void)
- bool start (const char ∗base_url, bool &available)
- const string & file (void) const
- bool stop (bool failure, bool invalidate)
- bool link (const char ∗link_path)
- bool link (const char ∗link_path, uid_t uid, gid_t gid)
- bool clean (unsigned long long int size=1)
- virtual bool cb (unsigned long long int size)
- operator bool (void)
- bool created_available (void)
- void created (time_t val)
- void created_force (time_t val)
- time_t created (void)
- bool validtill_available (void)
- time_t validtill (void)
- void validtill_force (time_t val)
- void validtill (time_t val)

3.2.1 Detailed Description

High level interface to cache operations (same functionality :)) and additional functionality to integrate into grid-manager environment.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 DataCache::DataCache (void)

Default constructor (non-functional cache).

3.2.2.2 DataCache::DataCache (const char $*$ *cache_path*, const char $*$ *cache_data_path*, const char $*$ *cache_link_path*, const char $*$ *id*, uid_t *cache_uid*, gid_t *cache_gid*)

Constructor

Parameters:

cache_path path to directory with cache info files

cache_data_path path to directory with cache data files

cache_link_path path used to create link in case cache_directory is visible under different name during actual usage

id identifier used to claim files in cache

cache_uid owner of cahce (0 for public cache)

cache_gid owner group of cache (0 for public cache)

3.2.2.3 DataCache::DataCache (const DataCache & *cache*)

Copy constructor.

3.2.2.4 DataCache::∼DataCache (void)

and destructor

3.2.3 Member Function Documentation

3.2.3.1 virtual bool DataCache::cb (unsigned long long int *size*) `[virtual]`

Callback implementation to clean at least 1 byte.

Reimplemented from DataCallback.

3.2.3.2 bool DataCache::clean (unsigned long long int *size* = 1)

Remove some amount of oldest information from cache. Returns true on success.

Parameters:

size amount to be removed (bytes)

3.2.3.3 time_t DataCache::created (void) `[inline]`

Get creation time.

3.2.3.4 void DataCache::created (time_t *val*) `[inline]`

Set creation time (if not already set).

Parameters:

val creation time

3.2.3.5 bool DataCache::created_available (void) `[inline]`

Check if there is an information about creation time.

3.2.3.6 void DataCache::created_force (time_t *val*) `[inline]`

Set creation time (even if already set).

Parameters:

val creation time

3.2.3.7 const string& DataCache::file (void) const `[inline]`

Returns path to file which contains/will contain content of assigned url.

3.2.3.8 bool DataCache::link (const char ∗ *link_path*, uid_t *uid*, gid_t *gid*)

Parameters:

uid set owner of soft-link to uid

gid set group of soft-link to gid

3.2.3.9 bool DataCache::link (const char ∗ *link_path*)

Must be called to create soft-link to cache file. All nessary directories will be created. Returns false on error (usually that means soft-link already exists).

Parameters:

link_path path for soft-link.

3.2.3.10 DataCache::operator bool (void) `[inline]`

Returns true of object is useable.

3.2.3.11 bool DataCache::start (const char ∗ *base_url*, bool & *available*)

Prepare cache for downloading file. On success returns true. This function can block for long time if there is another process downloading same url.

Parameters:

base_url url to assign to file in cache (file's identifier)

available contains true on exit if file is already in cache

3.2.3.12 bool DataCache::stop (bool *failure*, bool *invalidate*)

This method must be called after file was downloaded or download failed.

Parameters:

failure true if download failed

3.2.3.13 void DataCache::validtill (time_t *val*) [inline]

Get invalidation time.

3.2.3.14 time_t DataCache::validtill (void) [inline]

Set invalidation time (if not already set).

Parameters:
val validity time

3.2.3.15 bool DataCache::validtill_available (void) [inline]

Check if there is an information about invalidation time.

3.2.3.16 void DataCache::validtill_force (time_t *val*) [inline]

Set invalidation time (even if already set).

Parameters:
val validity time

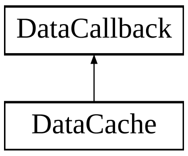
The documentation for this class was generated from the following file:

- datacache.h

3.3 DataCallback Class Reference

`#include <datacallback.h>`

Inheritance diagram for DataCallback::



Public Member Functions

- virtual bool **cb** (int)
- virtual bool **cb** (unsigned int)
- virtual bool **cb** (long long int)
- virtual bool **cb** (unsigned long long int)

3.3.1 Detailed Description

This class is used by DataHandle to report missing space on local filesystem. One of 'cb' functions here will be called if operation initiated by DataHandle::start_reading runs out of disk space.

The documentation for this class was generated from the following file:

- datacallback.h

3.4 DataHandle Class Reference

```
#include <datahandle.h>
```

Public Types

- enum failure_reason_t { **common_failure** = 0, **credentials_expired_failure** = 1 }

Public Member Functions

- DataHandle (DataPoint ∗url_)
- ∼DataHandle (void)
- bool start_reading (DataBufferPar &buffer)
- bool start_writing (DataBufferPar &buffer, DataCallback ∗space_cb=NULL)
- bool stop_reading (void)
- bool stop_writing (void)
- bool analyze (long int ∗bufsize, int ∗bufnum, bool ∗cache, bool ∗local)
- bool check (void)
- bool remove (void)
- bool list_files (list< DataPoint::FileInfo > &files, bool resolve=true)
- bool out_of_order (void)
- void out_of_order (bool v)
- void additional_checks (bool v)
- bool additional_checks (void)
- void secure (bool v)
- bool secure (void)
- void passive (bool v)
- failure_reason_t failure_reason (void)
- void range (unsigned long long int start=0, unsigned long long end=0)

3.4.1 Detailed Description

DataHandle is kind of generalized file handle. Differently from file handle it does not support operations read() and write(). Instead it initiates operation and uses object of class DataBufferPar to pass actual data. It also provides other operations like querying parameters of remote object. It is used by higher-level classes DataMove and DataMovePar to provide data transfer service for application.

3.4.2 Member Enumeration Documentation

3.4.2.1 enum DataHandle::failure_reason_t

Reason of transfer failure.

3.4.3 Constructor & Destructor Documentation

3.4.3.1 DataHandle::DataHandle (DataPoint ∗ *url_*)

Constructor

Parameters:

url_ URL. Should not be destroyed before DataHandle itself.

3.4.3.2 DataHandle::∼DataHandle (void)

Destructor. No comments.

3.4.4 Member Function Documentation

3.4.4.1 bool DataHandle::additional_checks (void) `[inline]`

Check if additional checks before 'reading' and 'writing' will be performed.

3.4.4.2 void DataHandle::additional_checks (bool *v*) `[inline]`

Allow/disallow to make check for existance of remote file (and probably other checks too) before initiating 'reading' and 'writing' operations.

Parameters:

v true if allowed (default is true).

3.4.4.3 bool DataHandle::analyze (long int ∗ *bufsize*, int ∗ *bufnum*, bool ∗ *cache*, bool ∗ *local*)

Analyze url and provide hints.

Parameters:

bufsize returns suggested size of buffers to store data.

bufnum returns suggested number of buffers.

cache returns true if url is allowed to be cached.

local return true if URL is accessed locally (`file://`)

3.4.4.4 bool DataHandle::check (void)

Query remote server or local file system to check if object is accessible.

3.4.4.5 failure_reason_t DataHandle::failure_reason (void) `[inline]`

Returns reason of transfer failure.

3.4.4.6 bool DataHandle::list_files (list< DataPoint::FileInfo > & *files*, bool *resolve* = true)

List files in directory (URL must point to directory/group).

Parameters:

files will contain list of file names and optionally their attributes.

resolve if false no information about attributes will be retrieved.

3.4.4.7 void DataHandle::out_of_order (bool *v*)

Allow/disallow DataHandle to produce scattered data during 'reading' operation.

Parameters:

v true if allowed.

3.4.4.8 bool DataHandle::out_of_order (void)

Returns true if URL can accept scatterd data (like arbitrary access to local file) for 'writing' operation.

3.4.4.9 void DataHandle::passive (bool *v*)

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

3.4.4.10 void DataHandle::range (unsigned long long int *start* = 0, unsigned long long *end* = 0) `[inline]`

Set range of bytes to retrieve. Default values correspond to whole file.

3.4.4.11 bool DataHandle::remove (void)

Remove/delete object at URL.

3.4.4.12 bool DataHandle::secure (void)

Check if heavy security during data transfer is allowed.

3.4.4.13 void DataHandle::secure (bool *v*)

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default is true only for gsiftp://).

3.4.4.14 bool DataHandle::start_reading (DataBufferPar & *buffer*)

Start reading data from URL. Separate thread to transfer data will be created. No other operation can be performed while 'reading' is in progress.

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before stop_-reading was called and returned. Returns true on success.

3.4.4.15 bool DataHandle::start_writing (DataBufferPar & *buffer*, DataCallback ∗ *space_cb* = NULL)

Start writing data to URL. Separate thread to transfer data will be created. No other operation can be performed while 'writing' is in progress.

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned. space_cb callback which is called if there is not enough to space storing data. Currently implemented only for `file:///` URL. Returns true on success.

3.4.4.16 bool DataHandle::stop_reading (void)

Stop reading. It MUST be called after corressponding start_reading method. Either after whole data is transfered or to cancel transfer. Use 'buffer' object to find out when data is transfered.

3.4.4.17 bool DataHandle::stop_writing (void)

Same as stop_reading but for corresponding start_writing.

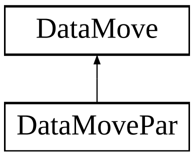
The documentation for this class was generated from the following file:

- datahandle.h

3.5 DataMove Class Reference

`#include <datamove.h>`

Inheritance diagram for DataMove::



Public Types

- typedef void(∗ **callback**)(DataMove ∗, DataMove::result, void ∗)
- enum result {

  success = 0, read_acquire_error = 1, write_acquire_error = 2, read_resolve_error = 3,

  write_resolve_error = 4, preregister_error = 5, read_start_error = 6, write_start_error = 7,

  read_error = 8, write_error = 9, transfer_error = 10, read_stop_error = 11,

  write_stop_error = 12, postregister_error = 13, cache_error = 14, system_error = 15,

  credentials_expired_error = 16, undefined_error = -1 }

Public Member Functions

- DataMove (void)
- ∼DataMove (void)
- result Transfer (DataPoint &source, DataPoint &destination, DataCache &cache, const UrlMap &map, callback cb=NULL, void ∗arg=NULL, const char ∗prefix=NULL)
- result Transfer (DataPoint &source, DataPoint &destination, DataCache &cache, const UrlMap &map, unsigned long long int min_speed, time_t min_speed_time, unsigned long long int min_-average_speed, time_t max_inactivity_time, callback cb=NULL, void ∗arg=NULL, const char ∗prefix=NULL)
- bool verbose (void)
- void verbose (bool)
- void verbose (const string &prefix)
- bool retry (void)
- void retry (bool)
- void secure (bool)
- void passive (bool)
- void force_to_meta (bool)
- bool checks (void)
- void checks (bool v)
- void set_default_min_speed (unsigned long long int min_speed, time_t min_speed_time)
- void set_default_min_average_speed (unsigned long long int min_average_speed)
- void set_default_max_inactivity_time (time_t max_inactivity_time)

3.5.1 Detailed Description

A purpose of this class is to provide service for moves data between 2 locations specified by URLs. It's main action is represented by methods DataMove::Transfer.

3.5.2 Member Enumeration Documentation

3.5.2.1 enum DataMove::result

Error code/failure reason.

Enumeration values:

success Operation completed successfully.

read_acquire_error Source is bad URL or can't be used due to some reason.

write_acquire_error Destination is bad URL or can't be used due to some reason.

read_resolve_error Resolving of meta-URL for source failed.

write_resolve_error Resolving of meta-URL for destination failed.

preregister_error First stage of registration of meta-URL failed.

read_start_error Can't read from source.

write_start_error Can't write to destination.

read_error Failed while reading from source.

write_error Failed while writing to destination.

transfer_error Failed while transfering data (mostly timeout).

read_stop_error Failed while finishing reading from source.

write_stop_error Failed while finishing writing to destination.

postregister_error Last stage of registration of meta-URL failed.

cache_error Error in caching procedure.

system_error Some system function returned unexpected error.

credentials_expired_error Error due to provided credentials are expired.

undefined_error Unknown/undefined error.

3.5.3 Constructor & Destructor Documentation

3.5.3.1 DataMove::DataMove (void)

Constructor.

3.5.3.2 DataMove::~DataMove (void)

Destructor.

3.5.4 Member Function Documentation

3.5.4.1 void DataMove::checks (bool *v*)

Set if to make check for existance of remote file (and probably other checks too) before initiating 'reading' and 'writing' operations.

Parameters:
v true if allowed (default is true).

3.5.4.2 bool DataMove::checks (void)

Check if check for existance of remote file is done before initiating 'reading' and 'writing' operations.

3.5.4.3 void DataMove::force_to_meta (bool)

Set if file should be transfered and registered even if such LFN is already registered and source is not one of registered locations.

3.5.4.4 void DataMove::passive (bool)

Set if passive transfer should be used for FTP-like transfers.

3.5.4.5 void DataMove::retry (bool)

Set if transfer will be retried in case of failure.

3.5.4.6 bool DataMove::retry (void)

Check if transfer will be retried in case of failure.

3.5.4.7 void DataMove::secure (bool)

Set if high level of security (encryption) will be used duirng transfer if available.

3.5.4.8 void DataMove::set_default_max_inactivity_time (time_t *max_inactivity_time*) `[inline]`

Set maximal allowed time for waiting for any data. For more information see description of DataSpeed class.

3.5.4.9 void DataMove::set_default_min_average_speed (unsigned long long int *min_average_speed*) `[inline]`

Set minimal allowed average transfer speed (default is 0 averaged over whole time of transfer. For more information see description of DataSpeed class.

3.5.4.10 void DataMove::set_default_min_speed (unsigned long long int *min_speed*, time_t *min_speed_time*) `[inline]`

Set minimal allowed transfer speed (default is 0) to 'min_speed'. If speed drops below for time longer than 'min_speed_time' error is raised. For more information see description of DataSpeed class.

3.5.4.11 result DataMove::Transfer (DataPoint & *source*, DataPoint & *destination*, DataCache & *cache*, const UrlMap & *map*, unsigned long long int *min_speed*, time_t *min_speed_time*, unsigned long long int *min_average_speed*, time_t *max_inactivity_time*, callback *cb* = NULL, void ∗ *arg* = NULL, const char ∗ *prefix* = NULL)

Initiates transfer from 'source' to 'destination'.

Parameters:

min_speed minimal allowed current speed.

min_speed_time time for which speed should be less than 'min_speed' before transfer fails.

min_average_speed minimal allowed average speed.

max_inactivity_time time for which should be no activity before transfer fails.

3.5.4.12 result DataMove::Transfer (DataPoint & *source*, DataPoint & *destination*, DataCache & *cache*, const UrlMap & *map*, callback *cb* = NULL, void ∗ *arg* = NULL, const char ∗ *prefix* = NULL)

Initiates transfer from 'source' to 'destination'.

Parameters:

source source URL.

destination destination URL.

cache controls caching of downloaded files (if destination url is "file://"). If caching is not needed default constructor DataCache() can be used.

map URL mapping/convertion table (for 'source' URL).

cb ifnot NULL, transfer is done in separate thread and 'cb' is called after transfer completes/fails.

arg passed to 'cb'.

prefix if 'verbose' is activated this information will be printed before each line representing current transfer status.

3.5.4.13 void DataMove::verbose (const string & *prefix*)

Activate printing information about transfer status.

Parameters:

prefix use this string if 'prefix' in DataMove::Transfer is NULL.

3.5.4.14 void DataMove::verbose (bool)

Activate printing information about transfer status.

3.5.4.15 bool DataMove::verbose (void)

Check if printing information about transfer status is activated.

The documentation for this class was generated from the following file:

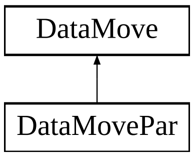
- datamove.h

3.6 DataMovePar Class Reference

Wrapper around DataMove class to handle few transfers at once.

`#include <datamovepar.h>`

Inheritance diagram for DataMovePar::



Public Member Functions

- DataMovePar (void)
- ∼DataMovePar (void)
- bool Add (const char ∗source_url, const char ∗destination_url)
- bool Get (string &source_url, string &destination_url, result &res)
- bool Transfer (int num=5)
- bool Transfer (DataCache cache, const UrlMap &map, int num=5)

3.6.1 Detailed Description

Wrapper around DataMove class to handle few transfers at once.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 DataMovePar::DataMovePar (void)

Constructor.

3.6.2.2 DataMovePar::∼DataMovePar (void)

Destructor. Object can't be destoryed while there is any transfer in progress.

3.6.3 Member Function Documentation

3.6.3.1 bool DataMovePar::Add (const char ∗ *source_url*, const char ∗ *destination_url*)

Add one more source and destination pair to list of handled transfers.

Parameters:

source_url URL (or meta-URL) of source file

destination_url URL (or meta-URL) of destination file

3.6.3.2 bool DataMovePar::Get (string & *source_url*, string & *destination_url*, result & *res*)

Get source and destination pair from list with result of transfer

Parameters:

source_url on exit contains URL (or meta-URL) of source file

destination_url on exit contains URL (or meta-URL) of destination file

res result of operation

3.6.3.3 bool DataMovePar::Transfer (DataCache *cache*, const UrlMap & *map*, int *num* = 5)

Perform transfer

Parameters:

cache to control data caching (use default constructor for no caching)

map to change/map source URLs (use default constructor for no maping)

num number of simultaneous transfers

3.6.3.4 bool DataMovePar::Transfer (int *num* = 5)

Perform transfer

Parameters:

num number of simultaneous transfers

The documentation for this class was generated from the following file:

- datamovepar.h

3.7 DataPoint Class Reference

```
#include <datapoint.h>
```

Public Member Functions

- DataPoint (const char ∗url)
- bool meta_resolve (bool source)
- bool meta_resolve (bool source, const UrlMap &maps)
- bool meta_preregister (bool replication, bool force=false)
- bool meta_postregister (bool replication, bool failure)
- bool meta_preunregister (bool replication)
- bool meta_unregister (bool all)
- bool list_files (list< DataPoint::FileInfo > &files, bool resolve=true)
- bool get_info (DataPoint::FileInfo &fi)
- bool meta_size_available (void)
- void meta_size (unsigned long long int val)
- void meta_size_force (unsigned long long int val)
- unsigned long long int meta_size (void) const
- bool meta_checksum_available (void)
- void meta_checksum (const char ∗val)
- void meta_checksum_force (const char ∗val)
- const char ∗ meta_checksum (void) const
- bool meta_created_available (void)
- void meta_created (time_t val)
- void meta_created_force (time_t val)
- time_t meta_created (void) const
- bool meta_validtill_available (void)
- void meta_validtill (time_t val)
- void meta_validtill_force (time_t val)
- time_t meta_validtill (void) const
- bool meta (void) const
- bool accepts_meta (void)
- bool provides_meta (void)
- void meta (const DataPoint &p)
- bool meta_compare (const DataPoint &p)
- bool meta_stored (void)
- bool local (void) const
- bool map (const UrlMap &maps)
- bool sort (const UrlMap &maps)
- DataPoint & **operator=** (const DataPoint &)
- **operator bool** (void) const
- const string & current_location (void) const
- const string & current_meta_location (void) const
- bool next_location (void)
- bool have_location (void)
- bool have_locations (void)
- bool remove_location (void)
- bool remove_locations (const DataPoint &p)

- int tries (void)
- void tries (int n)
- string base_url (void) const
- string canonic_url (void) const
- const char ∗ lfn (void) const
- bool add_location (const char ∗meta, const char ∗loc)

Public Attributes

- list< Location > **locations**

Friends

- ostream & **operator**<< (ostream &o, const DataPoint &point)

3.7.1 Detailed Description

DataPoint is an abstraction of URL. It can handle URLs of type `file://`, `ftp://`, gsiftp://, `http://`, `https://`, httpg:// (HTTP over GSI), se:// (NG web service over HTTPG) and meta-URLs (URLs of Infexing Services) rc://, rls://. DataPoint provides means to resolve meta-URL into multiple URLs and to loop through them.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 DataPoint::DataPoint (const char ∗ *url*)

Constructor requres URL or meta-URL to be provided.

3.7.3 Member Function Documentation

3.7.3.1 bool DataPoint::accepts_meta (void) `[inline]`

If endpoint can have any use from meta information.

3.7.3.2 bool DataPoint::add_location (const char ∗ *meta*, const char ∗ *loc*)

Add URL to list.

Parameters:

meta meta-name (name of location/service).

loc URL.

3.7.3.3 string DataPoint::base_url (void) const

Returns URL which was passed to constructor.

3.7.3.4 string DataPoint::canonic_url (void) const

Returns URL which was passed to constructor with location names removed, port number added, etc.

3.7.3.5 const string& DataPoint::current_location (void) const `[inline]`

Returns current (resolved) URL.

3.7.3.6 const string& DataPoint::current_meta_location (void) const `[inline]`

Returns meta information used to create curent URL. For RC that is location's name. For RLS that is equal to pfn.

3.7.3.7 bool DataPoint::get_info (DataPoint::FileInfo & *fi*)

Retrieve properties of object pointed by meta-URL of DataPoint object. It works only for meta-URL.

Parameters:

fi contains retrieved information.

3.7.3.8 bool DataPoint::have_location (void)

Returns false if out of retries.

3.7.3.9 bool DataPoint::have_locations (void)

Returns true if number of resolved URLs is not 0.

3.7.3.10 const char* DataPoint::lfn (void) const `[inline]`

Returns name which is given to file in Indexing Service (aka LFN).

3.7.3.11 bool DataPoint::list_files (list< DataPoint::FileInfo > & *files*, bool *resolve* = true)

Obtain information about objects and their properties available under meta-URL of DataPoint object. It works only for meta-URL.

Parameters:

files list of obtained objects.

resolve if false, do not try to obtain propertiers of objects.

3.7.3.12 bool DataPoint::local (void) const `[inline]`

Check if file is local (URL is `file://`).

3.7.3.13 bool DataPoint::map (const UrlMap & *maps*)

Map url (change it) according to table provided in maps.

Parameters:

maps mapping information.

3.7.3.14 void DataPoint::meta (const DataPoint & *p*) `[inline]`

Acquire meta-information from another object. Defined values a not overwritten.

Parameters:

p object from which information is taken.

3.7.3.15 bool DataPoint::meta (void) const `[inline]`

Check if URL is meta-URL.

3.7.3.16 const char∗ DataPoint::meta_checksum (void) const `[inline]`

Get value of meta-information ’checksum’.

3.7.3.17 void DataPoint::meta_checksum (const char ∗ *val*) `[inline]`

Set value of meta-information ’checksum’ if not already set.

3.7.3.18 bool DataPoint::meta_checksum_available (void) `[inline]`

Check if meta-information ’checksum’ is available.

3.7.3.19 void DataPoint::meta_checksum_force (const char ∗ *val*) `[inline]`

Set value of meta-information ’checksum’.

3.7.3.20 bool DataPoint::meta_compare (const DataPoint & *p*) `[inline]`

Compare meta-information form another object. Undefined values are not used for comparison. Default result is ’true’.

Parameters:

p object to which compare.

3.7.3.21 time_t DataPoint::meta_created (void) const `[inline]`

Get value of meta-information ’creation/modification time’.

3.7.3.22 void DataPoint::meta_created (time_t *val*) [inline]

Set value of meta-information 'creation/modification time' if not already set.

3.7.3.23 bool DataPoint::meta_created_available (void) [inline]

Check if meta-information 'creation/modification time' is available.

3.7.3.24 void DataPoint::meta_created_force (time_t *val*) [inline]

Set value of meta-information 'creation/modification time'.

3.7.3.25 bool DataPoint::meta_postregister (bool *replication*, bool *failure*)

Used for same purpose as meta_preregister. Should be called after actual transfer of file successfully finished.

Parameters:

replication if true then file is being replicated between 2 locations registered in Indexing Service under same name.

failure not used.

3.7.3.26 bool DataPoint::meta_preregister (bool *replication*, bool *force* = false)

This function registers physical location of file into Indexing Service. It should be called ∗before∗ actual transfer to that location happens.

Parameters:

replication if true then file is being replicated between 2 locations registered in Indexing Service under same name.

force if true, perform registration of new file even if it already exists. Should be used to fix failures in Indexing Service.

3.7.3.27 bool DataPoint::meta_preunregister (bool *replication*)

Should be called if file transfer failed. It removes changwes made by meta_preregister.

3.7.3.28 bool DataPoint::meta_resolve (bool *source*, const UrlMap & *maps*)

Resolve meta-URL into list of ordinary URLs and obtain meta-information about file. Also sort obtained list so that URLs mentioned in UrlMap object are placed first. This is used during transfer to access local locations first.

Parameters:

maps list of mappings of remote URLs to (potentially) local locations.

3.7.3.29 bool DataPoint::meta_resolve (bool *source*)

Resolve meta-URL into list of ordinary URLs and obtain meta-information about file. Can be called for object representing ordinary URL or already resolved object.

Parameters:

source true if DataPoint object represents source of information

3.7.3.30 unsigned long long int DataPoint::meta_size (void) const `[inline]`

Get value of meta-information 'size'.

3.7.3.31 void DataPoint::meta_size (unsigned long long int *val*) `[inline]`

Set value of meta-information 'size' if not already set.

3.7.3.32 bool DataPoint::meta_size_available (void) `[inline]`

Check if meta-information 'size' is available.

3.7.3.33 void DataPoint::meta_size_force (unsigned long long int *val*) `[inline]`

Set value of meta-information 'size'.

3.7.3.34 bool DataPoint::meta_stored (void) `[inline]`

Check if file is registered in Indexing Service. Proper value is obtinable only after meta-resolve.

3.7.3.35 bool DataPoint::meta_unregister (bool *all*)

Remove information about file registered in Indexing Service.

Parameters:

all if true information about file itself is (LFN) is removed. Otherwise only particular physical instance is unregistered.

3.7.3.36 time_t DataPoint::meta_validtill (void) const `[inline]`

Get value of meta-information 'validity time'.

3.7.3.37 void DataPoint::meta_validtill (time_t *val*) `[inline]`

Set value of meta-information 'validity time' if not already set.

3.7.3.38 bool DataPoint::meta_validtill_available (void) `[inline]`

Check if meta-information 'validity time' is available.

3.7.3.39 void DataPoint::meta_validtill_force (time_t *val*) [inline]

Set value of meta-information ’validity time’.

3.7.3.40 bool DataPoint::next_location (void)

Switch to next location in list of URLs. At last location switch to first if number of allowed retries does s not exceeded. Returns false if no retries left.

3.7.3.41 bool DataPoint::provides_meta (void) [inline]

If endpoint can provide at least some meta information directly.

3.7.3.42 bool DataPoint::remove_location (void)

Remove remove current URL from list.

3.7.3.43 bool DataPoint::remove_locations (const DataPoint & *p*)

Remove locations present in another DataPoint object.

3.7.3.44 bool DataPoint::sort (const UrlMap & *maps*)

Sort list of URLs so that those listed in mapping table are put first.

Parameters:
maps mapping information.

3.7.3.45 void DataPoint::tries (int *n*) [inline]

Set number of retries.

3.7.3.46 int DataPoint::tries (void) [inline]

Returns number of retries left.

The documentation for this class was generated from the following file:

- datapoint.h

3.8 DataPoint::FileInfo Class Reference

```
#include <datapoint.h>
```

Public Types

- enum **Type** { **file_type_unknown** = 0, **file_type_file** = 1, **file_type_dir** = 2 }

Public Member Functions

- FileInfo (const char ∗name_="")
- **operator bool** (void)

Public Attributes

- string **name**
- list< string > **urls**
- unsigned long long int size
- bool size_available
- string checksum
- bool checksum_available
- time_t created
- bool created_available
- time_t valid
- bool valid_available
- Type type

3.8.1 Detailed Description

FileInfo stores information about file (meta-information). Although all members are public it is mot desirable to modify them directly outside DataPoint class.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 DataPoint::FileInfo::FileInfo (const char ∗ *name_* = "") `[inline]`

Fyle type - usually file_type_file - ordinary file.

3.8.3 Member Data Documentation

3.8.3.1 string DataPoint::FileInfo::checksum

If size is known.

3.8.3.2 bool DataPoint::FileInfo::checksum_available

Checksum of file.

3.8.3.3 time_t DataPoint::FileInfo::created

If checksum is known.

3.8.3.4 bool DataPoint::FileInfo::created_available

Creation/modification time.

3.8.3.5 unsigned long long int DataPoint::FileInfo::size

Physical enpoints/URLs at which file can be accessed.

3.8.3.6 bool DataPoint::FileInfo::size_available

Size of filein bytes.

3.8.3.7 Type DataPoint::FileInfo::type

If validity is known.

3.8.3.8 time_t DataPoint::FileInfo::valid

If time is known.

3.8.3.9 bool DataPoint::FileInfo::valid_available

Valid till time.

The documentation for this class was generated from the following file:

- datapoint.h

3.9 DataPoint::Location Class Reference

```
#include <datapoint.h>
```

Public Member Functions

- **Location** (const char ∗url_)
- **Location** (const char ∗meta_, const char ∗url_, bool existing_=true)
- **Location** (const string &url_)
- **Location** (const string &meta_, const string &url_)

Friends

- class DataPoint
- ostream & **operator**<< (ostream &o, const DataPoint &point)

3.9.1 Detailed Description

DataPoint::Location represent physical service at which files are located aka "base URL" inculding it's name (as given in Indexing Service). Currently it is used only internally by DataPoint class and for printing debug information.

The documentation for this class was generated from the following file:

- datapoint.h

3.10 DataSpeed Class Reference

```
#include <dataspeed.h>
```

Public Member Functions

- DataSpeed (time_t base=DATASPEED_AVERAGING_PERIOD)
- DataSpeed (unsigned long long int min_speed, time_t min_speed_time, unsigned long long int min_-average_speed, time_t max_inactivity_time, time_t base=DATASPEED_AVERAGING_PERIOD)
- ∼DataSpeed (void)
- void verbose (bool val)
- void verbose (const string &prefix)
- bool verbose (void)
- void set_min_speed (unsigned long long int min_speed, time_t min_speed_time)
- void set_min_average_speed (unsigned long long int min_average_speed)
- void set_max_inactivity_time (time_t max_inactivity_time)
- void set_base (time_t base_=DATASPEED_AVERAGING_PERIOD)
- void reset (void)
- bool transfer (unsigned long long int n=0)
- void hold (bool disable)
- bool min_speed_failure ()
- bool min_average_speed_failure ()
- bool max_inactivity_time_failure ()
- unsigned long long int transfered_size (void)

3.10.1 Detailed Description

Keeps track of average and instantaneous speed. Also detects data transfer inactivity and other transfer timeouts.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 DataSpeed::DataSpeed (time_t *base* = DATASPEED_AVERAGING_PERIOD)

Constructor

Parameters:

base time period used to average values (default 1 minute).

3.10.2.2 DataSpeed::DataSpeed (unsigned long long int *min_speed*, time_t *min_speed_time*, unsigned long long int *min_average_speed*, time_t *max_inactivity_time*, time_t *base* = DATASPEED_AVERAGING_PERIOD)

Constructor

Parameters:

base time period used to average values (default 1 minute).

min_speed minimal allowed speed (Butes per second). If speed drops and holds below threshold for min_speed_time_ seconds error is triggered.

min_speed_time

min_average_speed_ minimal average speed (Bytes per second) to trigger error. Averaged over whole current transfer time.

max_inactivity_time - if no data is passing for specified amount of time (seconds), error is triggered.

3.10.2.3 DataSpeed::∼DataSpeed (void)

Destructor.

3.10.3 Member Function Documentation

3.10.3.1 void DataSpeed::hold (bool *disable*)

Turn off speed control.

Parameters:

disable true to turn off.

3.10.3.2 bool DataSpeed::max_inactivity_time_failure () `[inline]`

Check if maximal inactivity time error was triggered.

3.10.3.3 bool DataSpeed::min_average_speed_failure () `[inline]`

Check if minimal average speed error was triggered.

3.10.3.4 bool DataSpeed::min_speed_failure () `[inline]`

Check if minimal speed error was triggered.

3.10.3.5 void DataSpeed::reset (void)

Reset all counters and triggers.

3.10.3.6 void DataSpeed::set_base (time_t *base_* = DATASPEED_AVERAGING_PERIOD)

Set averaging time period.

Parameters:

base time period used to average values (default 1 minute).

3.10.3.7 void DataSpeed::set_max_inactivity_time (time_t *max_inactivity_time*)

Set inactivity tiemout.

Parameters:

max_inactivity_time - if no data is passing for specified amount of time (seconds), error is triggered.

3.10.3.8 void DataSpeed::set_min_average_speed (unsigned long long int *min_average_speed*)

Set minmal avaerage speed.

Parameters:

min_average_speed_ minimal average speed (Bytes per second) to trigger error. Averaged over whole current transfer time.

3.10.3.9 void DataSpeed::set_min_speed (unsigned long long int *min_speed*, time_t *min_speed_time*)

Set minimal allowed speed.

Parameters:

min_speed minimal allowed speed (Butes per second). If speed drops and holds below threshold for min_speed_time_ seconds error is triggered.

min_speed_time

3.10.3.10 bool DataSpeed::transfer (unsigned long long int *n* = 0)

Inform object, about amount of data has been transfered. All errors are triggered by this method. To make them work application must call this method periodically even with zero value.

Parameters:

n amount of data transfered (bytes).

3.10.3.11 unsigned long long int DataSpeed::transfered_size (void) `[inline]`

Returns amount of data this object knows about.

3.10.3.12 bool DataSpeed::verbose (void)

Check if speed information is going to be printed.

3.10.3.13 void DataSpeed::verbose (const string & *prefix*)

Print information about current speed and amout of data.

Parameters:

'prefix' add this string at the beginning of every string.

3.10.3.14 void DataSpeed::verbose (bool *val*)

Activate printing information about current time speeds, amount of transfered data.

The documentation for this class was generated from the following file:

- dataspeed.h

Index