ARC Data Library libarcdata

Generated by Doxygen 1.6.1

Wed Jan 30 09:03:07 2013

Contents

Chapter 1

Deprecated List

Global Arc::DataStatus::CacheErrorRetryable

Global Arc::DataStatus::CheckErrorRetryable

Global Arc::DataStatus::CreateDirectoryErrorRetryable

Global Arc::DataStatus::DeleteErrorRetryable

Global Arc::DataStatus::GenericErrorRetryable

Global Arc::DataStatus::ListErrorRetryable

Global Arc::DataStatus::ListNonDirError ListError with errno set to ENOTDIR should be used instead

Global Arc::DataStatus::PostRegisterErrorRetryable

Global Arc::DataStatus::PreRegisterErrorRetryable

Global Arc::DataStatus::ReadAcquireErrorRetryable

Global Arc::DataStatus::ReadErrorRetryable

Global Arc::DataStatus::ReadFinishErrorRetryable

Global Arc::DataStatus::ReadPrepareErrorRetryable

Global Arc::DataStatus::ReadResolveErrorRetryable

Global Arc::DataStatus::ReadStartErrorRetryable

Global Arc::DataStatus::ReadStopErrorRetryable

Global Arc::DataStatus::RenameErrorRetryable

Global Arc::DataStatus::StageErrorRetryable

Global Arc::DataStatus::StatErrorRetryable

Global Arc::DataStatus::StatNotPresentError StatError with errno set to ENOENT should be used instead

Global Arc::DataStatus::TransferErrorRetryable

Global Arc::DataStatus::UnregisterErrorRetryable

Global Arc::DataStatus::WriteAcquireErrorRetryable

Global Arc::DataStatus::WriteErrorRetryable

Global Arc::DataStatus::WriteFinishErrorRetryable

Global Arc::DataStatus::WritePrepareErrorRetryable

Global Arc::DataStatus::WriteResolveErrorRetryable

Global Arc::DataStatus::WriteStartErrorRetryable

Global Arc::DataStatus::WriteStopErrorRetryable

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Class Hierarchy

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

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

Module Documentation

5.1 ARC data library (libarcdata)

Data Structures

- class Arc::DataBuffer

  Represents set of buffers.

- class Arc::DataCallback

  Callbacks to be used when there is not enough space on the local filesystem.

- class Arc::DataHandle

  This class is a wrapper around the DataPoint class.

- class Arc::DataMover

  DataMover provides an interface to transfer data between two DataPoints.

- class Arc::DataPoint

  A DataPoint represents a data resource and is an abstraction of a URL.

- class Arc::DataPointDirect

  DataPointDirect represents "physical" data objects.

- class Arc::DataPointIndex

  DataPointIndex represents "index" data objects, e.g. catalogs.

- class Arc::DataSpeed

  Keeps track of average and instantaneous transfer speed.

- class Arc::DataStatus

  Status code returned by many DataPoint methods.

- struct Arc::CacheParameters

  Contains data on the parameters of a cache.

- class Arc::FileCache

  FileCache provides an interface to all cache operations.

- class Arc::FileCacheHash

  FileCacheHash provides methods to make hashes from strings.

- class Arc::FileInfo

  FileInfo stores information about files (metadata).

- class Arc::URLMap

  URLMap allows mapping certain patterns of URLs to other URLs.

Functions

- std::ostream & Arc::operator<< (std::ostream &o, const DataStatus &d)

5.1.1 Detailed Description

libarcdata is a library for access to data on the Grid. It provides a uniform interface to several types of Grid storage and catalogs using various protocols. The protocols useable on a given system depend on the packages installed. The interface can be used to read, write, list, transfer and delete data to and from storage systems and catalogs.

The library uses ARC's dynamic plugin mechanism to load plugins for specific protocols only when required at runtime. These plugins are called Data Manager Components (DMCs). The DataHandle class takes care of automatically loading the required DMC at runtime to create a DataPoint object representing a resource accessible through a given protocol. DataHandle should always be used instead of DataPoint directly.

DataMover provides a simple high-level interface to copy files. For more fine-grained control over data transfer see the examples in DataHandle.

To create a new DMC for a protocol which is not yet supported see the instruction and examples in the DataPoint class documentation. This documentation also gives a complete overview of the interface.

The following protocols are currently supported in standard distributions of ARC.

ARC (arc://) - Protocol to access the Chelonia storage system developed by ARC.

File (file://) - Regular local file system.

GridFTP (gsiftp://) - GridFTP is essentially the FTP protocol with GSI security. Regular FTP can also be used.

HTTP(S/G) (http://) - Hypertext Transfer Protocol. HTTP over SSL (HTTPS) and HTTP over GSI (HTTPG) are also supported.

LDAP (ldap://) - Lightweight Directory Access Protocol. LDAP is used in grids mainly to store information about grid services or resources rather than to store data itself.

LFC (lfc://) - The LCG File Catalog (LFC) is a replica catalog developed by CERN. It consists of a hierarchical namespace of grid files and each filename can be associated with one or more physical locations.

SRM (srm://) - The Storage Resource Manager (SRM) protocol allows access to data distributed across physical storage through a unified namespace and management interface.

XRootd (root://) - Protocol for data access across large scale storage clusters. More information can be found at http://xrootd.slac.stanford.edu/

5.1.2 Function Documentation

5.1.2.1 std::ostream& Arc::operator<< (std::ostream & *o*, const DataStatus & *d*) [inline]

Write a human-friendly readable string with all error information to o.

Chapter 6

Data Structure Documentation

6.1 Arc::CacheParameters Struct Reference

Contains data on the parameters of a cache.

`#include <arc/data/FileCache.h>`

6.1.1 Detailed Description

Contains data on the parameters of a cache.

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

- FileCache.h

6.2 Arc::DataBuffer Class Reference

Represents set of buffers.

`#include <arc/data/DataBuffer.h>`

Data Structures

- struct **buf_desc**

  internal struct to describe status of every buffer

- class **checksum_desc**

  internal class with pointer to object to compute checksum

Public Member Functions

- operator bool () const
- DataBuffer (unsigned int size=65536, int blocks=3)
- DataBuffer (CheckSum *cksum, unsigned int size=65536, int blocks=3)
- ~DataBuffer ()
- bool set (CheckSum *cksum=NULL, unsigned int size=65536, int blocks=3)
- int add (CheckSum *cksum)
- char * operator[ ] (int n)
- bool for_read (int &handle, unsigned int &length, bool wait)
- bool for_read ()
- 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 for_write ()
- 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 ()
- bool eof_write ()
- bool error_read ()
- bool error_write ()
- bool error_transfer ()
- bool error ()
- bool wait_any ()
- bool wait_used ()
- bool wait_for_read ()
- bool wait_for_write ()
- bool checksum_valid (int index) const
- bool checksum_valid () const

- const CheckSum * checksum_object (int index) const
- const CheckSum * checksum_object () const
- bool wait_eof_read ()
- bool wait_read ()
- bool wait_eof_write ()
- bool wait_write ()
- bool wait_eof ()
- unsigned long long int eof_position () const
- unsigned int buffer_size () const

Data Fields

- DataSpeed speed

6.2.1 Detailed Description

Represents set of buffers. This class is used during data transfer using DataPoint classes.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Arc::DataBuffer::DataBuffer (unsigned int *size* = `65536`, int *blocks* = `3`)

Construct a new DataBuffer object.

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

6.2.2.2 Arc::DataBuffer::DataBuffer (CheckSum * *cksum*, unsigned int *size* = `65536`, int *blocks* = `3`)

Construct a new DataBuffer object with checksum computation.

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

cksum object which will compute checksum. Should not be destroyed until DataBuffer itself.

6.2.3 Member Function Documentation

6.2.3.1 int Arc::DataBuffer::add (CheckSum * *cksum*)

Add a checksum object which will compute checksum of buffer.

Parameters:

cksum object which will compute checksum. Should not be destroyed until DataBuffer itself.

Returns:

integer position in the list of checksum objects.

6.2.3.2 unsigned int Arc::DataBuffer::buffer_size () const

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

6.2.3.3 const CheckSum* Arc::DataBuffer::checksum_object (int *index*) const

Returns CheckSum object at specified index or NULL if index is not in list.

Parameters:

index of the checksum in question.

6.2.3.4 bool Arc::DataBuffer::checksum_valid (int *index*) const

Returns true if the specified checksum was successfully computed.

Parameters:

index of the checksum in question.

Returns:

false if index is not in list

6.2.3.5 void Arc::DataBuffer::eof_read (bool *v*)

Informs object if there will be no more request for 'read' buffers.

Parameters:

v true if no more requests.

6.2.3.6 void Arc::DataBuffer::eof_write (bool *v*)

Informs object if there will be no more request for 'write' buffers.

Parameters:

v true if no more requests.

6.2.3.7 void Arc::DataBuffer::error_read (bool *v*)

Informs object if error occurred on 'read' side.

Parameters:

v true if error

6.2.3.8 void Arc::DataBuffer::error_write (bool *v*)

Informs object if error occurred on 'write' side.

Parameters:

v true if error

6.2.3.9 bool Arc::DataBuffer::for_read ()

Check if there are buffers which can be taken by for_read(). This function checks only for buffers and does not take eof and error conditions into account.

Returns:

true if buffers are available

6.2.3.10 bool Arc::DataBuffer::for_read (int & *handle*, unsigned int & *length*, bool *wait*)

Request buffer for READING INTO it. Should be called when data is being read from a source. The calling code should write data into the returned buffer and then call is_read().

Parameters:

handle filled with buffer's number.

length filled with size of buffer

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

Returns:

true on success For python bindings pattern of this method is (bool, handle, length) for_read(wait). Here buffer for reading to be provided by external code and provided to DataBuffer object through is_read() method. Content of buffer must not exceed provided length.

6.2.3.11 bool Arc::DataBuffer::for_write ()

Check if there are buffers which can be taken by for_write(). This function checks only for buffers and does not take eof and error conditions into account.

Returns:

true if buffers are available

6.2.3.12 bool Arc::DataBuffer::for_write (int & *handle*, unsigned int & *length*, unsigned long long int & *offset*, bool *wait*)

Request buffer for WRITING FROM it. Should be called when data is being written to a destination. The calling code should write the data contained in the returned buffer and then call is_written().

Parameters:

handle returns buffer's number.

length returns size of buffer

offset returns buffer offset

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

Returns:

true on success For python bindings pattern of this method is (bool, handle, length, offset, buffer) for_write(wait). Here buffer is string with content of buffer provided by DataBuffer object.

6.2.3.13 bool Arc::DataBuffer::is_notwritten (char * *buf*)

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

Parameters:

buf - address of buffer

Returns:

true if buffer was successfully informed

6.2.3.14 bool Arc::DataBuffer::is_notwritten (int *handle*)

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

Parameters:

handle buffer's number.

Returns:

true if buffer was successfully informed

6.2.3.15 bool Arc::DataBuffer::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.

Returns:

true if buffer was successfully informed

6.2.3.16 bool Arc::DataBuffer::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.

Returns:

true if buffer was successfully informed For python bindings pattern of that method is bool is_-read(handle,buffer,offset). Here buffer is string containing content of buffer to be passed to DataBuffer object.

6.2.3.17 bool Arc::DataBuffer::is_written (char * *buf*)

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

Returns:

true if buffer was successfully informed

6.2.3.18 bool Arc::DataBuffer::is_written (int *handle*)

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

Returns:

true if buffer was successfully informed

6.2.3.19 char* Arc::DataBuffer::operator[ ] (int *n*)

Direct access to buffer by number.

Parameters:

n buffer number

Returns:

buffer content

6.2.3.20 bool Arc::DataBuffer::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 until DataBuffer itself.

Returns:

true if buffers were successfully initialized

6.2.3.21 bool Arc::DataBuffer::wait_any ()

Wait (max 60 sec.) till any action happens in object.

Returns:

true if action is eof on any side

6.2.3.22 bool Arc::DataBuffer::wait_for_read ()

Wait till no more buffers taken for "READING INTO" left in object.

Returns:

true if an error occurred while waiting

6.2.3.23 bool Arc::DataBuffer::wait_for_write ()

Wait till no more buffers taken for "WRITING FROM" left in object.

Returns:

true if an error occurred while waiting

6.2.3.24 bool Arc::DataBuffer::wait_used ()

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

Returns:

true if an error occurred while waiting

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

- DataBuffer.h

6.3 Arc::DataCallback Class Reference

Callbacks to be used when there is not enough space on the local filesystem.

`#include <arc/data/DataCallback.h>`

Public Member Functions

- DataCallback ()
- virtual ~DataCallback ()
- virtual bool cb (int)
- virtual bool cb (unsigned int)
- virtual bool cb (long long int)
- virtual bool cb (unsigned long long int)

6.3.1 Detailed Description

Callbacks to be used when there is not enough space on the local filesystem. If DataPoint::StartWriting() tries to pre-allocate disk space but finds that there is not enough to write the whole file, one of the 'cb' functions here will be called with the required space passed as a parameter. Users should define their own subclass of this class depending on how they wish to free up space. Each callback method should return true if the space was freed, false otherwise. This subclass should then be used as a parameter in StartWriting().

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

- DataCallback.h

6.4 Arc::DataHandle Class Reference

This class is a wrapper around the DataPoint class.

`#include <arc/data/DataHandle.h>`

Public Member Functions

- DataHandle (const URL &url, const UserConfig &usercfg)
- ~DataHandle ()
- DataPoint * operator-> ()
- const DataPoint * operator-> () const
- DataPoint & operator* ()
- const DataPoint & operator* () const
- bool operator! () const
- operator bool () const

Static Public Member Functions

- static DataPoint * GetPoint (const URL &url, const UserConfig &usercfg)

6.4.1 Detailed Description

This class is a wrapper around the DataPoint class. It simplifies the construction, use and destruction of DataPoint objects and should be used instead of DataPoint classes directly. The appropriate DataPoint subclass is created automatically and stored internally in DataHandle. A DataHandle instance can be thought of as a pointer to the DataPoint instance and the DataPoint can be accessed through the usual dereference operators. A DataHandle cannot be copied.

This class is the main way to access remote data items and obtain information about them. Below is an example of accessing the last 512 bytes of a file stored on a GridFTP server. To simply copy a whole file DataMover::Transfer() can be used.

```
#include <iostream>
#include <arc/data/DataPoint.h>
#include <arc/data/DataHandle.h>
#include <arc/data/DataBuffer.h>

using namespace Arc;

int main(void) {
  #define DESIRED_SIZE 512
  Arc::UserConfig usercfg;
  URL url("gsiftp://localhost/files/file_test_21");
  DataHandle handle(url,usercfg);
  if(!handle || !(*handle)) {
    std::cerr<<"Unsupported URL protocol or malformed URL"<<std::endl;
    return -1;
  };
  handle->SetSecure(false) // GridFTP servers generally do not have encrypted da
     ta channel
  FileInfo info;
  if(!handle->Stat(info)) {
    std::cerr<<"Failed Stat"<<std::endl;
    return -1;
  };
```

```
  unsigned long long int fsize = handle->GetSize();
  if(fsize == (unsigned long long int)-1) {
    std::cerr<<"file size is not available"<<std::endl;
    return -1;
  };
  if(fsize == 0) {
    std::cerr<<"file is empty"<<std::endl;
    return -1;
  };
  if(fsize > DESIRED_SIZE) {
    handle->Range(fsize-DESIRED_SIZE,fsize-1);
  };
  DataBuffer buffer;
  if(!handle->StartReading(buffer)) {
    std::cerr<<"Failed to start reading"<<std::endl;
    return -1;
  };
  for(;;) {
    int n;
    unsigned int length;
    unsigned long long int offset;
    if(!buffer.for_write(n,length,offset,true)) {
      break;
    };
    std::cout<<"BUFFER: "<<offset<<": "<<length<<" :"<<std::string((const char*)
     (buffer[n]),length)<<std::endl;
    buffer.is_written(n);
  };
  if(buffer.error()) {
    std::cerr<<"Transfer failed"<<std::endl;
  };
  handle->StopReading();
  return 0;
}
```

And the same example in python

```
import arc

desired_size = 512
usercfg = arc.UserConfig()
url = arc.URL("gsiftp://localhost/files/file_test_21")
handle = arc.DataHandle(url,usercfg)
point = handle.__ref__()
point.SetSecure(False) # GridFTP servers generally do not have encrypted data ch
     annel
info = arc.FileInfo("")
point.Stat(info)
print "Name: ", info.GetName()
fsize = info.GetSize()
if fsize > desired_size:
    point.Range(fsize-desired_size,fsize-1)
buffer = arc.DataBuffer()
point.StartReading(buffer)
while True:
    n = 0
    length = 0
    offset = 0
    ( r, n, length, offset, buf) = buffer.for_write(True)
    if not r: break
    print "BUFFER: ", offset, ": ", length, " :", buf
    buffer.is_written(n);
point.StopReading()
```

6.4.2 Member Function Documentation

6.4.2.1 static DataPoint* Arc::DataHandle::GetPoint (const URL & *url*, const UserConfig & *usercfg*) [inline, static]

Returns a pointer to new DataPoint object corresponding to URL. This static method is mostly for bindings to other languages and if available scope of obtained DataPoint is undefined.

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

- DataHandle.h

6.5 Arc::DataMover Class Reference

DataMover provides an interface to transfer data between two DataPoints.

`#include <arc/data/DataMover.h>`

Public Types

- typedef void(∗ callback)(DataMover ∗mover, DataStatus status, void ∗arg)

Public Member Functions

- DataMover ()
- ~DataMover ()
- DataStatus Transfer (DataPoint &source, DataPoint &destination, FileCache &cache, const URLMap &map, callback cb=NULL, void ∗arg=NULL, const char ∗prefix=NULL)
- DataStatus Transfer (DataPoint &source, DataPoint &destination, FileCache &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)
- DataStatus Delete (DataPoint &url, bool errcont=false)
- void Cancel ()
- bool verbose ()
- void verbose (bool)
- void verbose (const std::string &prefix)
- bool retry ()
- void retry (bool)
- void secure (bool)
- void passive (bool)
- void force_to_meta (bool)
- bool checks ()
- 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)
- void set_progress_indicator (DataSpeed::show_progress_t func=NULL)
- void set_preferred_pattern (const std::string &pattern)

6.5.1 Detailed Description

DataMover provides an interface to transfer data between two DataPoints. Its main action is represented by Transfer methods.

6.5.2 Member Typedef Documentation

6.5.2.1 typedef void(∗ Arc::DataMover::callback)(DataMover ∗mover, DataStatus status, void ∗arg)

Callback function which can be passed to Transfer().

Parameters:

mover this DataMover instance

status result of the transfer

arg arguments passed in 'arg' parameter of Transfer()

6.5.3 Member Function Documentation

6.5.3.1 void Arc::DataMover::checks (bool *v*)

Set if extra checks are made before transfer starts. If turned on, extra checks are done before commencing the transfer, such as checking the existence of the source file and verifying consistency of metadata between index service and physical replica.

6.5.3.2 DataStatus Arc::DataMover::Delete (DataPoint & *url*, bool *errcont* = `false`)

Delete the file at url. This method differs from DataPoint::Remove() in that for index services, it deletes all replicas in addition to removing the index entry.

Parameters:

url file to delete

errcont if true then replica information will be deleted from an index service even if deleting the physical replica fails

Returns:

DataStatus object with result of deletion

6.5.3.3 void Arc::DataMover::set_default_max_inactivity_time (time_t *max_inactivity_time*) `[inline]`

Set maximal allowed time for no data transfer. For more information see description of DataSpeed class.

Parameters:

max_inactivity_time maximum time in seconds which is allowed without any data transfer

6.5.3.4 void Arc::DataMover::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.

Parameters:

min_average_speed minimum average transfer rate over the whole transfer in bytes/second

6.5.3.5 void Arc::DataMover::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.

Parameters:

min_speed minimum transfer rate in bytes/second

min_speed_time time in seconds over which min_speed is measured

6.5.3.6 void Arc::DataMover::set_preferred_pattern (const std::string & *pattern*) [inline]

Set a preferred pattern for ordering of replicas. This pattern will be used in the case of an index service URL with multiple physical replicas and allows sorting of those replicas in order of preference. It consists of one or more patterns separated by a pipe character (|) listed in order of preference. If the dollar character ($) is used at the end of a pattern, the pattern will be matched to the end of the hostname of the replica. Example: "srm://myhost.org|.uk$|.ch$"

Parameters:

pattern pattern on which to order replicas

6.5.3.7 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, FileCache & *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'. An optional callback can be provided, in which case this method starts a separate thread for the transfer and returns immediately. The callback is called after the transfer finishes.

Parameters:

source source DataPoint to read from.

destination destination DataPoint to write to.

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

map URL mapping/conversion table (for 'source' URL). If URL mapping is not needed the default constructor URLMap() can be used.

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.

cb if not 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.

Returns:

DataStatus object with transfer result

6.5.3.8 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, FileCache & *cache*, const URLMap & *map*, callback *cb* = NULL, void * *arg* = NULL, const char * *prefix* = NULL)

Initiates transfer from 'source' to 'destination'. An optional callback can be provided, in which case this method starts a separate thread for the transfer and returns immediately. The callback is called after the transfer finishes.

Parameters:

source source DataPoint to read from.

destination destination DataPoint to write to.

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

map URL mapping/conversion table (for 'source' URL). If URL mapping is not needed the default constructor URLMap() can be used.

cb if not 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.

Returns:

DataStatus object with transfer result

6.5.3.9 void Arc::DataMover::verbose (const std::string & *prefix*)

Set output of transfer status information during transfer.

Parameters:

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

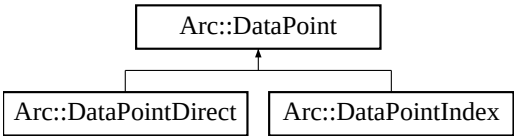
The documentation for this class was generated from the following file:

- DataMover.h

6.6 Arc::DataPoint Class Reference

A DataPoint represents a data resource and is an abstraction of a URL.

`#include <arc/data/DataPoint.h>`Inheritance diagram for Arc::DataPoint::



Public Types

- enum DataPointAccessLatency { ACCESS_LATENCY_ZERO, ACCESS_LATENCY_SMALL, ACCESS_LATENCY_LARGE }
- enum DataPointInfoType {

  INFO_TYPE_MINIMAL = 0, INFO_TYPE_NAME = 1, INFO_TYPE_TYPE = 2, INFO_TYPE_-TIMES = 4,

  INFO_TYPE_CONTENT = 8, INFO_TYPE_ACCESS = 16, INFO_TYPE_STRUCT = 32, INFO_-TYPE_REST = 64,

  INFO_TYPE_ALL = 127 }
- typedef void(∗ Callback3rdParty)(unsigned long long int bytes_transferred)

Public Member Functions

- virtual ~DataPoint ()
- virtual const URL & GetURL () const
- virtual const UserConfig & GetUserConfig () const
- virtual bool SetURL (const URL &url)
- virtual std::string str () const
- virtual operator bool () const
- virtual bool operator! () const
- virtual DataStatus PrepareReading (unsigned int timeout, unsigned int &wait_time)
- virtual DataStatus PrepareWriting (unsigned int timeout, unsigned int &wait_time)
- virtual DataStatus StartReading (DataBuffer &buffer)=0
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback ∗space_cb=NULL)=0
- virtual DataStatus StopReading ()=0
- virtual DataStatus StopWriting ()=0
- virtual DataStatus FinishReading (bool error=false)
- virtual DataStatus FinishWriting (bool error=false)
- virtual DataStatus Check (bool check_meta)=0
- virtual DataStatus Remove ()=0
- virtual DataStatus Stat (FileInfo &file, DataPointInfoType verb=INFO_TYPE_ALL)=0
- virtual DataStatus Stat (std::list< FileInfo > &files, const std::list< DataPoint ∗ > &urls, Data-PointInfoType verb=INFO_TYPE_ALL)=0
- virtual DataStatus List (std::list< FileInfo > &files, DataPointInfoType verb=INFO_TYPE_ALL)=0
- virtual DataStatus CreateDirectory (bool with_parents=false)=0
- virtual DataStatus Rename (const URL &newurl)=0

- virtual void ReadOutOfOrder (bool v)=0
- virtual bool WriteOutOfOrder ()=0
- virtual void SetAdditionalChecks (bool v)=0
- virtual bool GetAdditionalChecks () const =0
- virtual void SetSecure (bool v)=0
- virtual bool GetSecure () const =0
- virtual void Passive (bool v)=0
- virtual DataStatus GetFailureReason (void) const
- virtual void Range (unsigned long long int start=0, unsigned long long int end=0)=0
- virtual DataStatus Resolve (bool source)=0
- virtual DataStatus Resolve (bool source, const std::list< DataPoint * > &urls)=0
- virtual bool Registered () const =0
- virtual DataStatus PreRegister (bool replication, bool force=false)=0
- virtual DataStatus PostRegister (bool replication)=0
- virtual DataStatus PreUnregister (bool replication)=0
- virtual DataStatus Unregister (bool all)=0
- virtual bool CheckSize () const
- virtual void SetSize (const unsigned long long int val)
- virtual unsigned long long int GetSize () const
- virtual bool CheckCheckSum () const
- virtual void SetCheckSum (const std::string &val)
- virtual const std::string & GetCheckSum () const
- virtual const std::string DefaultCheckSum () const
- virtual bool CheckModified () const
- virtual void SetModified (const Time &val)
- virtual const Time & GetModified () const
- virtual bool CheckValid () const
- virtual void SetValid (const Time &val)
- virtual const Time & GetValid () const
- virtual void SetAccessLatency (const DataPointAccessLatency &latency)
- virtual DataPointAccessLatency GetAccessLatency () const
- virtual long long int BufSize () const =0
- virtual int BufNum () const =0
- virtual bool Cache () const
- virtual bool Local () const =0
- virtual bool ReadOnly () const =0
- virtual int GetTries () const
- virtual void SetTries (const int n)
- virtual void NextTry ()
- virtual bool RequiresCredentials () const
- virtual bool IsIndex () const =0
- virtual bool IsStageable () const
- virtual bool AcceptsMeta () const =0
- virtual bool ProvidesMeta () const =0
- virtual void SetMeta (const DataPoint &p)
- virtual bool CompareMeta (const DataPoint &p) const
- virtual std::vector< URL > TransferLocations () const
- virtual const URL & CurrentLocation () const =0
- virtual const std::string & CurrentLocationMetadata () const =0
- virtual DataPoint * CurrentLocationHandle () const =0

- virtual DataStatus CompareLocationMetadata () const =0
- virtual bool NextLocation ()=0
- virtual bool LocationValid () const =0
- virtual bool LastLocation ()=0
- virtual bool HaveLocations () const =0
- virtual DataStatus AddLocation (const URL &url, const std::string &meta)=0
- virtual DataStatus RemoveLocation ()=0
- virtual DataStatus RemoveLocations (const DataPoint &p)=0
- virtual DataStatus ClearLocations ()=0
- virtual int AddCheckSumObject (CheckSum *cksum)=0
- virtual const CheckSum * GetCheckSumObject (int index) const =0
- virtual void SortLocations (const std::string &pattern, const URLMap &url_map)=0
- virtual void AddURLOptions (const std::map< std::string, std::string > &options)

Static Public Member Functions

- static DataStatus Transfer3rdParty (const URL &source, const URL &destination, const UserConfig &usercfg, Callback3rdParty callback=NULL)

Protected Member Functions

- DataPoint (const URL &url, const UserConfig &usercfg, PluginArgument *parg)
- virtual DataStatus Transfer3rdParty (const URL &source, const URL &destination, Callback3rdParty callback=NULL)

Protected Attributes

- URL url
- const UserConfig usercfg
- unsigned long long int size
- std::string checksum
- Time modified
- Time valid
- DataPointAccessLatency access_latency
- int triesleft
- DataStatus failure_code
- bool cache
- bool stageable
- std::set< std::string > valid_url_options

Static Protected Attributes

- static Logger logger

6.6.1 Detailed Description

A DataPoint represents a data resource and is an abstraction of a URL. DataPoint uses ARC's Plugin mechanism to dynamically load the required Data Manager Component (DMC) when necessary. A DMC typically defines a subclass of DataPoint (e.g. DataPointHTTP) and is responsible for a specific protocol (e.g. http). DataPoints should not be used directly, instead the DataHandle wrapper class should be used, which automatically loads the correct DMC. Examples of how to use DataPoint methods are shown in the DataHandle documentation.

DataPoint defines methods for access to the data resource. To transfer data between two DataPoints, DataMover::Transfer() can be used.

There are two subclasses of DataPoint, DataPointDirect and DataPointIndex. None of these three classes can be instantiated directly. DataPointDirect and its subclasses handle "physical" resources through protocols such as file, http and gsiftp. These classes implement methods such as StartReading() and StartWriting(). DataPointIndex and its subclasses handle resources such as indexes and catalogs and implement methods like Resolve() and PreRegister().

When creating a new DMC, a subclass of either DataPointDirect or DataPointIndex should be created, and the appropriate methods implemented. DataPoint itself has no direct external dependencies, but plugins may rely on third-party components. The new DMC must also add itself to the list of available plugins and provide an Instance() method which returns a new instance of itself, if the supplied arguments are valid for the protocol. Here is an example implementation of a new DMC for protocol MyProtocol which represents a physical resource accessible through protocol my://

```
#include <arc/data/DataPointDirect.h>

namespace Arc {

class DataPointMyProtocol : public DataPointDirect {
 public:
  DataPointMyProtocol(const URL& url, const UserConfig& usercfg, PluginArgument*
      parg);
  static Plugin* Instance(PluginArgument *arg);
  virtual DataStatus StartReading(DataBuffer& buffer);
  ...
};

DataPointMyProtocol::DataPointMyProtocol(const URL& url, const UserConfig& userc
     fg, PluginArgument* parg)
 : DataPointDirect(url, usercfg, parg) {
  ...
}

DataPointMyProtocol::StartReading(DataBuffer& buffer) { ... }

...

Plugin* DataPointMyProtocol::Instance(PluginArgument *arg) {
  DataPointPluginArgument *dmcarg = dynamic_cast<DataPointPluginArgument*>(arg);

  if (!dmcarg)
    return NULL;
  if (((const URL &)(*dmcarg)).Protocol() != "my")
    return NULL;
  return new DataPointMyProtocol(*dmcarg, *dmcarg, dmcarg);
}

} // namespace Arc

Arc::PluginDescriptor ARC_PLUGINS_TABLE_NAME[] = {
  { "my", "HED:DMC", "My protocol", 0, &Arc::DataPointMyProtocol::Instance },
  { NULL, NULL, NULL, 0, NULL }
```

```
};
```

6.6.2 Member Typedef Documentation

6.6.2.1 typedef void(∗ Arc::DataPoint::Callback3rdParty)(unsigned long long int bytes_transferred)

Callback for use in 3rd party transfer. Will be called periodically during the transfer with the number of bytes transferred so far.

Parameters:

bytes_transferred the number of bytes transferred so far

6.6.3 Member Enumeration Documentation

6.6.3.1 enum Arc::DataPoint::DataPointAccessLatency

Describes the latency to access this URL. For now this value is one of a small set specified by the enumeration. In the future with more sophisticated protocols or information it could be replaced by a more fine-grained list of possibilities such as an int value.

Enumerator:

ACCESS_LATENCY_ZERO URL can be accessed instantly.

ACCESS_LATENCY_SMALL URL has low (but non-zero) access latency, for example staged from disk.

ACCESS_LATENCY_LARGE URL has a large access latency, for example staged from tape.

6.6.3.2 enum Arc::DataPoint::DataPointInfoType

Describes type of information about URL to request.

Enumerator:

INFO_TYPE_MINIMAL Whatever protocol can get with no additional effort.

INFO_TYPE_NAME Only name of object (relative).

INFO_TYPE_TYPE Type of object - currently file or dir.

INFO_TYPE_TIMES Timestamps associated with object.

INFO_TYPE_CONTENT Metadata describing content, like size, checksum, etc.

INFO_TYPE_ACCESS Access control - ownership, permission, etc.

INFO_TYPE_STRUCT Fine structure - replicas, transfer locations, redirections.

INFO_TYPE_REST All the other parameters.

INFO_TYPE_ALL All the parameters.

6.6.4 Constructor & Destructor Documentation

6.6.4.1 Arc::DataPoint::DataPoint (const URL & *url*, const UserConfig & *usercfg*, PluginArgument * *parg*) [protected]

Constructor. Constructor is protected because DataPoints should not be created directly. Subclasses should however call this in their constructors to set various common attributes.

Parameters:

url The URL representing the DataPoint

usercfg User configuration object

parg plugin argument

6.6.5 Member Function Documentation

6.6.5.1 virtual int Arc::DataPoint::AddCheckSumObject (CheckSum * *cksum*) [pure virtual]

Add a checksum object which will compute checksum during data transfer.

Parameters:

cksum object which will compute checksum. Should not be destroyed until DataPointer itself.

Returns:

integer position in the list of checksum objects.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.2 virtual DataStatus Arc::DataPoint::AddLocation (const URL & *url*, const std::string & *meta*) [pure virtual]

Add URL representing physical replica to list of locations.

Parameters:

url Location URL to add.

meta Location meta information.

Returns:

LocationAlreadyExistsError if location already exists, otherwise success

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.3 virtual void Arc::DataPoint::AddURLOptions (const std::map< std::string, std::string > & *options*) [virtual]

Add URL options to this DataPoint's URL object. Invalid options for the specific DataPoint instance will not be added.

Parameters:

options map of option, value pairs

6.6.5.4 virtual DataStatus Arc::DataPoint::Check (bool *check_meta*) [pure virtual]

Query the DataPoint to check if object is accessible. If check_meta is true this method will also try to provide meta information about the object. Note that for many protocols an access check also provides meta information and so check_meta may have no effect.

Parameters:

check_meta If true then the method will try to retrieve meta data during the check.

Returns:

success if the object is accessible by the caller.

Implemented in Arc::DataPointIndex.

6.6.5.5 virtual DataStatus Arc::DataPoint::CompareLocationMetadata () const [pure virtual]

Compare metadata of DataPoint and current location.

Returns:

inconsistency error or error encountered during operation, or success

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.6 virtual bool Arc::DataPoint::CompareMeta (const DataPoint & *p*) const [virtual]

Compare meta information from another object. Undefined values are not used for comparison.

Parameters:

p object to which to compare.

6.6.5.7 virtual DataStatus Arc::DataPoint::CreateDirectory (bool *with_parents* = false) [pure virtual]

Create a directory. If the protocol supports it, this method creates the last directory in the path to the URL. It assumes the last component of the path is a file-like object and not a directory itself, unless the path ends in a directory separator. If with_parents is true then all missing parent directories in the path will also be created. The access control on the new directories is protocol-specific and may vary depending on protocol.

Parameters:

with_parents If true then all missing directories in the path are created

Returns:

success if the directory was created

6.6.5.8 virtual const std::string& Arc::DataPoint::CurrentLocationMetadata () const [pure virtual]

Returns meta information used to create current URL. Usage differs between different indexing services.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.9 virtual DataStatus Arc::DataPoint::FinishReading (bool *error* = false) [virtual]

Finish reading from the URL. Must be called after transfer of physical file has completed if PrepareReading() was called, to free resources, release requests that were made during preparation etc.

Parameters:

error If true then action is taken depending on the error.

Returns:

success if source was released properly

Reimplemented in Arc::DataPointIndex.

6.6.5.10 virtual DataStatus Arc::DataPoint::FinishWriting (bool *error* = false) [virtual]

Finish writing to the URL. Must be called after transfer of physical file has completed if PrepareWriting() was called, to free resources, release requests that were made during preparation etc.

Parameters:

error if true then action is taken depending on the error, for example cleaning the file from the storage

Returns:

success if destination was released properly

Reimplemented in Arc::DataPointIndex.

6.6.5.11 virtual DataStatus Arc::DataPoint::GetFailureReason (void) const [virtual]

Returns reason of transfer failure, as reported by callbacks. This could be different from the failure returned by the methods themselves.

6.6.5.12 virtual DataStatus Arc::DataPoint::List (std::list< FileInfo > & *files*, DataPointInfoType *verb* = INFO_TYPE_ALL) [pure virtual]

List hierarchical content of this object. If the DataPoint represents a directory or something similar its contents will be listed and put into files. If the DataPoint is file- like an error will be returned.

Parameters:

files will contain list of file names and requested attributes. There may be more attributes than requested. There may be less if object can't provide particular information.

verb defines attribute types which method must try to retrieve. It is not a failure if some attributes could not be retrieved due to limitation of protocol or access control.

Returns:

success if DataPoint is a directory-like object and could be listed.

6.6.5.13 virtual bool Arc::DataPoint::NextLocation () `[pure virtual]`

Switch to next location in list of URLs. At last location switch to first if number of allowed retries is not exceeded.

Returns:

false if no retries left.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.14 virtual void Arc::DataPoint::Passive (bool *v*) `[pure virtual]`

Set passive transfers for FTP-like protocols.

Parameters:

v true if passive should be used.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.15 virtual DataStatus Arc::DataPoint::PostRegister (bool *replication*) `[pure virtual]`

Index service post-registration. Used for same purpose as PreRegister. Should be called after actual transfer of file successfully finished to finalise registration in an index service.

Parameters:

replication if true, the file is being replicated between two locations registered in Indexing Service under the same name.

Returns:

success if post-registration succeeded

Implemented in Arc::DataPointDirect.

6.6.5.16 virtual DataStatus Arc::DataPoint::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*) `[virtual]`

Prepare DataPoint for reading. This method should be implemented by protocols which require preparation or staging of physical files for reading. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a ReadPrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call PrepareReading() again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel it by calling FinishReading(). When file preparation has finished, the physical file(s) to read from can be found from TransferLocations().

Parameters:

timeout If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.

wait_time If timeout is zero (caller would like asynchronous operation) and ReadPrepareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Returns:

Status of the operation

Reimplemented in Arc::DataPointIndex.

6.6.5.17 virtual DataStatus Arc::DataPoint::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*) [virtual]

Prepare DataPoint for writing. This method should be implemented by protocols which require preparation of physical files for writing. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a WritePrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call PrepareWriting() again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel or abort it by calling FinishWriting(true). When file preparation has finished, the physical file(s) to write to can be found from TransferLocations().

Parameters:

timeout If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.

wait_time If timeout is zero (caller would like asynchronous operation) and WritePrepareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Returns:

Status of the operation

Reimplemented in Arc::DataPointIndex.

6.6.5.18 virtual DataStatus Arc::DataPoint::PreRegister (bool *replication*, bool *force* = false) [pure virtual]

Index service pre-registration. This function registers the physical location of a file into an indexing service. It should be called ∗before∗ the actual transfer to that location happens.

Parameters:

replication if true, the file is being replicated between two locations registered in the indexing service under the same name.

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

Returns:

success if pre-registration succeeded

Implemented in Arc::DataPointDirect.

6.6.5.19 virtual DataStatus Arc::DataPoint::PreUnregister (bool *replication*) [pure virtual]

Index service pre-unregistration. Should be called if file transfer failed. It removes changes made by PreRegister().

Parameters:

replication if true, the file is being replicated between two locations registered in Indexing Service under the same name.

Returns:

success if pre-unregistration succeeded

Implemented in Arc::DataPointDirect.

6.6.5.20 virtual void Arc::DataPoint::Range (unsigned long long int *start* = 0, unsigned long long int *end* = 0) [pure virtual]

Set range of bytes to retrieve. Default values correspond to whole file. Both start and end bytes are included in the range, i.e. start - end + 1 bytes will be read.

Parameters:

start byte to start from

end byte to end at

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.21 virtual void Arc::DataPoint::ReadOutOfOrder (bool *v*) [pure virtual]

Allow/disallow DataPoint to read data out of order. If set to true then data may be read from source out of order or in parallel from multiple threads. For a transfer between two DataPoints this should only be set to true if WriteOutOfOrder() returns true for the destination. Only certain protocols support this option.

Parameters:

v true if allowed (default is false).

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.22 virtual DataStatus Arc::DataPoint::Rename (const URL & *newurl*) [pure virtual]

Rename a URL. This method renames the file or directory specified in the constructor to the new name specified in newurl. It only performs namespace operations using the paths of the two URLs and in general ignores any differences in protocol and host between them. It is assumed that checks that the URLs are consistent are done by the caller of this method. This method does not do any data transfer and is only implemented for protocols which support renaming as an atomic namespace operation.

Parameters:

newurl The new name for the URL

Returns:

success if the object was renamed

6.6.5.23 virtual DataStatus Arc::DataPoint::Resolve (bool *source*, const std::list< DataPoint * > & *urls*) [pure virtual]

Resolves several index service URLs. Can use bulk calls if protocol allows. The protocols and hosts of all the DataPoints in urls must be the same and the same as this DataPoint's protocol and host. This method can be called on any of the urls, for example urls.front()->Resolve(true, urls);

Parameters:

source true if DataPoint objects represent source of information

urls List of DataPoints to resolve. Protocols and hosts must match and match this DataPoint's protocol and host.

Returns:

success if any DataPoint was successfully resolved

6.6.5.24 virtual DataStatus Arc::DataPoint::Resolve (bool *source*) [pure virtual]

Resolves index service URL into list of ordinary URLs. Also obtains meta information about the file if possible. Resolve should be called for both source and destination URLs before a transfer. If source is true an error is returned if the file does not exist.

Parameters:

source true if DataPoint object represents source of information.

Returns:

success if DataPoint was successfully resolved

Implemented in Arc::DataPointDirect.

6.6.5.25 virtual void Arc::DataPoint::SetAdditionalChecks (bool *v*) [pure virtual]

Allow/disallow additional checks on a source DataPoint before transfer. If set to true, extra checks will be performed in DataMover::Transfer() before data transfer starts on for example existence of the source file (and probably other checks too).

Parameters:

v true if allowed (default is true).

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.26 virtual void Arc::DataPoint::SetMeta (const DataPoint & *p*) [virtual]

Copy meta information from another object. Already defined values are not overwritten.

Parameters:

p object from which information is taken.

6.6.5.27 virtual void Arc::DataPoint::SetSecure (bool *v*) [pure virtual]

Allow/disallow heavy security (data encryption) during data transfer.

Parameters:

v true if allowed (default depends on protocol).

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.28 virtual bool Arc::DataPoint::SetURL (const URL & *url*) [virtual]

Assigns new URL. The main purpose of this method is to reuse an existing connection for accessing a different object on the same server. The DataPoint implementation does not have to implement this method. If the supplied URL is not suitable or method is not implemented false is returned.

Parameters:

url New URL

Returns:

true if switching to new URL is supported and succeeded

6.6.5.29 virtual void Arc::DataPoint::SortLocations (const std::string & *pattern*, const URLMap & *url_map*) [pure virtual]

Sort locations according to the specified pattern and URLMap. See DataMover::set_preferred_pattern for a more detailed explanation of pattern matching. Locations present in url_map are preferred over others.

Parameters:

pattern a set of strings, separated by |, to match against.

url_map map of URLs to local URLs

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

6.6.5.30 virtual DataStatus Arc::DataPoint::StartReading (DataBuffer & *buffer*) [pure virtual]

Start reading data from URL. A separate thread to transfer data will be created. No other operation can be performed while reading is in progress. Progress of the transfer should be followed using the DataBuffer object.

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned. If StopReading() is not called explicitly to release buffer it will be released in destructor of DataPoint which also usually calls StopReading().

Returns:

success if a thread was successfully started to start reading

Implemented in Arc::DataPointIndex.

6.6.5.31 virtual DataStatus Arc::DataPoint::StartWriting (DataBuffer & *buffer*, DataCallback * *space_cb* = NULL) [pure virtual]

Start writing data to URL. A separate thread to transfer data will be created. No other operation can be performed while writing is in progress. Progress of the transfer should be followed using the DataBuffer object.

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before StopWriting() was called and returned. If StopWriting() is not called explicitly to release buffer it will be released in destructor of DataPoint which also usually calls StopWriting().

space_cb callback which is called if there is not enough space to store data. May not implemented for all protocols.

Returns:

success if a thread was successfully started to start writing

Implemented in Arc::DataPointIndex.

6.6.5.32 virtual DataStatus Arc::DataPoint::Stat (std::list< FileInfo > & *files*, const std::list< DataPoint * > & *urls*, DataPointInfoType *verb* = INFO_TYPE_ALL) [pure virtual]

Retrieve information about several DataPoints. If a DataPoint represents a directory or something similar, information about the object itself and not its contents will be obtained. This method can use bulk operations if the protocol supports it. The protocols and hosts of all the DataPoints in urls must be the same and the same as this DataPoint's protocol and host. This method can be called on any of the urls, for example urls.front()->Stat(files, urls); Calling this method with an empty list of urls returns success if the protocol supports bulk Stat, and an error if it does not and this can be used as a check for bulk support.

Parameters:

files will contain objects' names and requested attributes. There may be more attributes than requested. There may be less if objects can't provide particular information. The order of this list matches the order of urls. If a stat of any url fails then the corresponding FileInfo in this list will evaluate to false.

urls list of DataPoints to stat. Protocols and hosts must match and match this DataPoint's protocol and host.

verb defines attribute types which method must try to retrieve. It is not a failure if some attributes could not be retrieved due to limitation of protocol or access control.

Returns:

success if any information could be retrieved for any DataPoint

6.6.5.33 virtual DataStatus Arc::DataPoint::Stat (FileInfo & *file*, DataPointInfoType *verb* = INFO_TYPE_ALL) [pure virtual]

Retrieve information about this object. If the DataPoint represents a directory or something similar, information about the object itself and not its contents will be obtained.

Parameters:

file will contain object name and requested attributes. There may be more attributes than requested. There may be less if object can't provide particular information.

verb defines attribute types which method must try to retrieve. It is not a failure if some attributes could not be retrieved due to limitation of protocol or access control.

Returns:

success if any information could be retrieved

6.6.5.34 virtual DataStatus Arc::DataPoint::StopReading () [pure virtual]

Stop reading. Must be called after corresponding StartReading() method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred.

Returns:

outcome of stopping reading (not outcome of transfer itself)

Implemented in Arc::DataPointIndex.

6.6.5.35 virtual DataStatus Arc::DataPoint::StopWriting () [pure virtual]

Stop writing. Must be called after corresponding StartWriting() method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred.

Returns:

outcome of stopping writing (not outcome of transfer itself)

Implemented in Arc::DataPointIndex.

6.6.5.36 virtual DataStatus Arc::DataPoint::Transfer3rdParty (const URL & *source*, const URL & *destination*, Callback3rdParty *callback* = NULL) [protected, virtual]

Perform third party transfer. This method is protected because the static version should be used instead to load the correct DMC plugin for third party transfer.

Parameters:

source Source URL to pull data from

destination Destination URL which pulls data to itself

callback Optional monitoring callback

Returns:

outcome of transfer

6.6.5.37 static DataStatus Arc::DataPoint::Transfer3rdParty (const URL & *source*, const URL & *destination*, const UserConfig & *usercfg*, Callback3rdParty *callback* = NULL) [static]

Perform third party transfer. Credentials are delegated to the destination and it pulls data from the source, i.e. data flows directly between source and destination instead of through the client. A callback function can be supplied to monitor progress. This method blocks until the transfer is complete. It is static because third party transfer requires different DMC plugins than those loaded by DataHandle for the same protocol. The third party transfer plugins are loaded internally in this method.

Parameters:

source Source URL to pull data from

destination Destination URL which pulls data to itself

usercfg Configuration information

callback Optional monitoring callback

Returns:

outcome of transfer

6.6.5.38 virtual std::vector<URL> Arc::DataPoint::TransferLocations () const [virtual]

Returns physical file(s) to read/write, if different from CurrentLocation(). To be used with protocols which re-direct to different URLs such as Transport URLs (TURLs). The list is initially filled by PrepareReading and PrepareWriting. If this list is non-empty then real transfer should use a URL from this list. It is up to the caller to choose the best URL and instantiate new DataPoint for handling it. For consistency protocols which do not require redirections return original URL. For protocols which need redirection calling StartReading and StartWriting will use first URL in the list.

Reimplemented in Arc::DataPointIndex.

6.6.5.39 virtual DataStatus Arc::DataPoint::Unregister (bool *all*) [pure virtual]

Index service unregistration. 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 in CurrentLocation() is unregistered.

Returns:

success if unregistration succeeded

Implemented in Arc::DataPointDirect.

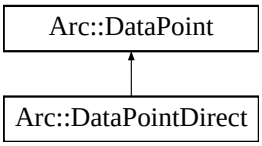
The documentation for this class was generated from the following file:

- DataPoint.h

6.7 Arc::DataPointDirect Class Reference

DataPointDirect represents "physical" data objects.

`#include <arc/data/DataPointDirect.h>`Inheritance diagram for Arc::DataPointDirect::



Public Member Functions

- virtual bool IsIndex () const
- virtual bool IsStageable () const
- virtual long long int BufSize () const
- virtual int BufNum () const
- virtual bool Local () const
- virtual bool ReadOnly () const
- virtual void ReadOutOfOrder (bool v)
- virtual bool WriteOutOfOrder ()
- virtual void SetAdditionalChecks (bool v)
- virtual bool GetAdditionalChecks () const
- virtual void SetSecure (bool v)
- virtual bool GetSecure () const
- virtual void Passive (bool v)
- virtual void Range (unsigned long long int start=0, unsigned long long int end=0)
- virtual int AddCheckSumObject (CheckSum *cksum)
- virtual const CheckSum * GetCheckSumObject (int index) const
- virtual DataStatus Resolve (bool source)
- virtual bool Registered () const
- virtual DataStatus PreRegister (bool replication, bool force=false)
- virtual DataStatus PostRegister (bool replication)
- virtual DataStatus PreUnregister (bool replication)
- virtual DataStatus Unregister (bool all)
- virtual bool AcceptsMeta () const
- virtual bool ProvidesMeta () const
- virtual const URL & CurrentLocation () const
- virtual DataPoint * CurrentLocationHandle () const
- virtual const std::string & CurrentLocationMetadata () const
- virtual DataStatus CompareLocationMetadata () const
- virtual bool NextLocation ()
- virtual bool LocationValid () const
- virtual bool HaveLocations () const
- virtual bool LastLocation ()
- virtual DataStatus AddLocation (const URL &url, const std::string &meta)
- virtual DataStatus RemoveLocation ()
- virtual DataStatus ClearLocations ()
- virtual void SortLocations (const std::string &, const URLMap &)

6.7.1 Detailed Description

DataPointDirect represents "physical" data objects. This class should never be used directly, instead inherit from it to provide a class for a specific access protocol.

6.7.2 Member Function Documentation

6.7.2.1 virtual int Arc::DataPointDirect::AddCheckSumObject (CheckSum * *cksum*) [virtual]

Add a checksum object which will compute checksum during data transfer.

Parameters:

cksum object which will compute checksum. Should not be destroyed until DataPointer itself.

Returns:

integer position in the list of checksum objects.

Implements Arc::DataPoint.

6.7.2.2 virtual DataStatus Arc::DataPointDirect::AddLocation (const URL & *url*, const std::string & *meta*) [virtual]

Add URL representing physical replica to list of locations.

Parameters:

url Location URL to add.

meta Location meta information.

Returns:

LocationAlreadyExistsError if location already exists, otherwise success

Implements Arc::DataPoint.

6.7.2.3 virtual DataStatus Arc::DataPointDirect::CompareLocationMetadata () const [virtual]

Compare metadata of DataPoint and current location.

Returns:

inconsistency error or error encountered during operation, or success

Implements Arc::DataPoint.

6.7.2.4 virtual const std::string& Arc::DataPointDirect::CurrentLocationMetadata () const [virtual]

Returns meta information used to create current URL. Usage differs between different indexing services.

Implements Arc::DataPoint.

6.7.2.5 virtual bool Arc::DataPointDirect::NextLocation () [virtual]

Switch to next location in list of URLs. At last location switch to first if number of allowed retries is not exceeded.

Returns:

false if no retries left.

Implements Arc::DataPoint.

6.7.2.6 virtual void Arc::DataPointDirect::Passive (bool *v*) [virtual]

Set passive transfers for FTP-like protocols.

Parameters:

v true if passive should be used.

Implements Arc::DataPoint.

6.7.2.7 virtual DataStatus Arc::DataPointDirect::PostRegister (bool *replication*) [virtual]

Index service post-registration. Used for same purpose as PreRegister. Should be called after actual transfer of file successfully finished to finalise registration in an index service.

Parameters:

replication if true, the file is being replicated between two locations registered in Indexing Service under the same name.

Returns:

success if post-registration succeeded

Implements Arc::DataPoint.

6.7.2.8 virtual DataStatus Arc::DataPointDirect::PreRegister (bool *replication*, bool *force* = false) [virtual]

Index service pre-registration. This function registers the physical location of a file into an indexing service. It should be called ∗before∗ the actual transfer to that location happens.

Parameters:

replication if true, the file is being replicated between two locations registered in the indexing service under the same name.

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

Returns:

success if pre-registration succeeded

Implements Arc::DataPoint.

6.7.2.9 virtual DataStatus Arc::DataPointDirect::PreUnregister (bool *replication*) [virtual]

Index service pre-unregistration. Should be called if file transfer failed. It removes changes made by PreRegister().

Parameters:

replication if true, the file is being replicated between two locations registered in Indexing Service under the same name.

Returns:

success if pre-unregistration succeeded

Implements Arc::DataPoint.

6.7.2.10 virtual void Arc::DataPointDirect::Range (unsigned long long int *start* = 0, unsigned long long int *end* = 0) [virtual]

Set range of bytes to retrieve. Default values correspond to whole file. Both start and end bytes are included in the range, i.e. start - end + 1 bytes will be read.

Parameters:

start byte to start from

end byte to end at

Implements Arc::DataPoint.

6.7.2.11 virtual void Arc::DataPointDirect::ReadOutOfOrder (bool *v*) [virtual]

Allow/disallow DataPoint to read data out of order. If set to true then data may be read from source out of order or in parallel from multiple threads. For a transfer between two DataPoints this should only be set to true if WriteOutOfOrder() returns true for the destination. Only certain protocols support this option.

Parameters:

v true if allowed (default is false).

Implements Arc::DataPoint.

6.7.2.12 virtual DataStatus Arc::DataPointDirect::Resolve (bool *source*) [virtual]

Resolves index service URL into list of ordinary URLs. Also obtains meta information about the file if possible. Resolve should be called for both source and destination URLs before a transfer. If source is true an error is returned if the file does not exist.

Parameters:

source true if DataPoint object represents source of information.

Returns:

success if DataPoint was successfully resolved

Implements Arc::DataPoint.

6.7.2.13 virtual void Arc::DataPointDirect::SetAdditionalChecks (bool *v*) [virtual]

Allow/disallow additional checks on a source DataPoint before transfer. If set to true, extra checks will be performed in DataMover::Transfer() before data transfer starts on for example existence of the source file (and probably other checks too).

Parameters:

v true if allowed (default is true).

Implements Arc::DataPoint.

6.7.2.14 virtual void Arc::DataPointDirect::SetSecure (bool *v*) [virtual]

Allow/disallow heavy security (data encryption) during data transfer.

Parameters:

v true if allowed (default depends on protocol).

Implements Arc::DataPoint.

6.7.2.15 virtual void Arc::DataPointDirect::SortLocations (const std::string & *pattern*, const URLMap & *url_map*) [inline, virtual]

Sort locations according to the specified pattern and URLMap. See DataMover::set_preferred_pattern for a more detailed explanation of pattern matching. Locations present in url_map are preferred over others.

Parameters:

pattern a set of strings, separated by |, to match against.

url_map map of URLs to local URLs

Implements Arc::DataPoint.

6.7.2.16 virtual DataStatus Arc::DataPointDirect::Unregister (bool *all*) [virtual]

Index service unregistration. 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 in CurrentLocation() is unregistered.

Returns:

success if unregistration succeeded

Implements Arc::DataPoint.

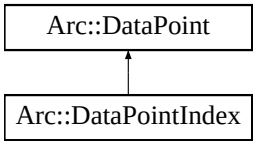
The documentation for this class was generated from the following file:

- DataPointDirect.h

6.8 Arc::DataPointIndex Class Reference

DataPointIndex represents "index" data objects, e.g. catalogs.

`#include <arc/data/DataPointIndex.h>`Inheritance diagram for Arc::DataPointIndex::



Public Member Functions

- virtual const URL & CurrentLocation () const
- virtual const std::string & CurrentLocationMetadata () const
- virtual DataPoint * CurrentLocationHandle () const
- virtual DataStatus CompareLocationMetadata () const
- virtual bool NextLocation ()
- virtual bool LocationValid () const
- virtual bool HaveLocations () const
- virtual bool LastLocation ()
- virtual DataStatus RemoveLocation ()
- virtual DataStatus ClearLocations ()
- virtual DataStatus AddLocation (const URL &url, const std::string &meta)
- virtual void SortLocations (const std::string &pattern, const URLMap &url_map)
- virtual bool IsIndex () const
- virtual bool IsStageable () const
- virtual bool AcceptsMeta () const
- virtual bool ProvidesMeta () const
- virtual void SetCheckSum (const std::string &val)
- virtual void SetSize (const unsigned long long int val)
- virtual bool Registered () const
- virtual void SetTries (const int n)
- virtual long long int BufSize () const
- virtual int BufNum () const
- virtual bool Local () const
- virtual bool ReadOnly () const
- virtual DataStatus PrepareReading (unsigned int timeout, unsigned int &wait_time)
- virtual DataStatus PrepareWriting (unsigned int timeout, unsigned int &wait_time)
- virtual DataStatus StartReading (DataBuffer &buffer)
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback *space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()
- virtual DataStatus FinishReading (bool error=false)
- virtual DataStatus FinishWriting (bool error=false)
- virtual std::vector< URL > TransferLocations () const
- virtual DataStatus Check (bool check_meta)
- virtual DataStatus Remove ()
- virtual void ReadOutOfOrder (bool v)

- virtual bool WriteOutOfOrder ()
- virtual void SetAdditionalChecks (bool v)
- virtual bool GetAdditionalChecks () const
- virtual void SetSecure (bool v)
- virtual bool GetSecure () const
- virtual DataPointAccessLatency GetAccessLatency () const
- virtual void Passive (bool v)
- virtual void Range (unsigned long long int start=0, unsigned long long int end=0)
- virtual int AddCheckSumObject (CheckSum *cksum)
- virtual const CheckSum * GetCheckSumObject (int index) const

6.8.1 Detailed Description

DataPointIndex represents "index" data objects, e.g. catalogs. This class should never be used directly, instead inherit from it to provide a class for a specific indexing service.

6.8.2 Member Function Documentation

6.8.2.1 virtual int Arc::DataPointIndex::AddCheckSumObject (CheckSum * *cksum*) [virtual]

Add a checksum object which will compute checksum during data transfer.

Parameters:

cksum object which will compute checksum. Should not be destroyed until DataPointer itself.

Returns:

integer position in the list of checksum objects.

Implements Arc::DataPoint.

6.8.2.2 virtual DataStatus Arc::DataPointIndex::AddLocation (const URL & *url*, const std::string & *meta*) [virtual]

Add URL representing physical replica to list of locations.

Parameters:

url Location URL to add.

meta Location meta information.

Returns:

LocationAlreadyExistsError if location already exists, otherwise success

Implements Arc::DataPoint.

6.8.2.3 virtual DataStatus Arc::DataPointIndex::Check (bool *check_meta*) [virtual]

Query the DataPoint to check if object is accessible. If check_meta is true this method will also try to provide meta information about the object. Note that for many protocols an access check also provides meta information and so check_meta may have no effect.

Parameters:

check_meta If true then the method will try to retrieve meta data during the check.

Returns:

success if the object is accessible by the caller.

Implements Arc::DataPoint.

6.8.2.4 virtual DataStatus Arc::DataPointIndex::CompareLocationMetadata () const [virtual]

Compare metadata of DataPoint and current location.

Returns:

inconsistency error or error encountered during operation, or success

Implements Arc::DataPoint.

6.8.2.5 virtual const std::string& Arc::DataPointIndex::CurrentLocationMetadata () const [virtual]

Returns meta information used to create current URL. Usage differs between different indexing services.

Implements Arc::DataPoint.

6.8.2.6 virtual DataStatus Arc::DataPointIndex::FinishReading (bool *error* = false) [virtual]

Finish reading from the URL. Must be called after transfer of physical file has completed if PrepareReading() was called, to free resources, release requests that were made during preparation etc.

Parameters:

error If true then action is taken depending on the error.

Returns:

success if source was released properly

Reimplemented from Arc::DataPoint.

6.8.2.7 virtual DataStatus Arc::DataPointIndex::FinishWriting (bool *error* = false) [virtual]

Finish writing to the URL. Must be called after transfer of physical file has completed if PrepareWriting() was called, to free resources, release requests that were made during preparation etc.

Parameters:

error if true then action is taken depending on the error, for example cleaning the file from the storage

Returns:

success if destination was released properly

Reimplemented from Arc::DataPoint.

6.8.2.8 virtual bool Arc::DataPointIndex::NextLocation () [virtual]

Switch to next location in list of URLs. At last location switch to first if number of allowed retries is not exceeded.

Returns:

false if no retries left.

Implements Arc::DataPoint.

6.8.2.9 virtual void Arc::DataPointIndex::Passive (bool *v*) [virtual]

Set passive transfers for FTP-like protocols.

Parameters:

v true if passive should be used.

Implements Arc::DataPoint.

6.8.2.10 virtual DataStatus Arc::DataPointIndex::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*) [virtual]

Prepare DataPoint for reading. This method should be implemented by protocols which require preparation or staging of physical files for reading. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a ReadPrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call PrepareReading() again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel it by calling FinishReading(). When file preparation has finished, the physical file(s) to read from can be found from TransferLocations().

Parameters:

timeout If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.

wait_time If timeout is zero (caller would like asynchronous operation) and ReadPrepareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Returns:

Status of the operation

Reimplemented from Arc::DataPoint.

6.8.2.11 virtual DataStatus Arc::DataPointIndex::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*) [virtual]

Prepare DataPoint for writing. This method should be implemented by protocols which require preparation of physical files for writing. It can act synchronously or asynchronously (if protocol supports it). In the first case the method will block until the file is prepared or the specified timeout has passed. In the second case the method can return with a WritePrepareWait status before the file is prepared. The caller should then wait some time (a hint from the remote service may be given in wait_time) and call PrepareWriting() again to poll for the preparation status, until the file is prepared. In this case it is also up to the caller to decide when the request has taken too long and if so cancel or abort it by calling FinishWriting(true). When file preparation has finished, the physical file(s) to write to can be found from TransferLocations().

Parameters:

timeout If non-zero, this method will block until either the file has been prepared successfully or the timeout has passed. A zero value means that the caller would like to call and poll for status.

wait_time If timeout is zero (caller would like asynchronous operation) and WritePrepareWait is returned, a hint for how long to wait before a subsequent call may be given in wait_time.

Returns:

Status of the operation

Reimplemented from Arc::DataPoint.

6.8.2.12 virtual void Arc::DataPointIndex::Range (unsigned long long int *start* = 0, unsigned long long int *end* = 0) [virtual]

Set range of bytes to retrieve. Default values correspond to whole file. Both start and end bytes are included in the range, i.e. start - end + 1 bytes will be read.

Parameters:

start byte to start from

end byte to end at

Implements Arc::DataPoint.

6.8.2.13 virtual void Arc::DataPointIndex::ReadOutOfOrder (bool *v*) [virtual]

Allow/disallow DataPoint to read data out of order. If set to true then data may be read from source out of order or in parallel from multiple threads. For a transfer between two DataPoints this should only be set to true if WriteOutOfOrder() returns true for the destination. Only certain protocols support this option.

Parameters:

v true if allowed (default is false).

Implements Arc::DataPoint.

6.8.2.14 virtual void Arc::DataPointIndex::SetAdditionalChecks (bool *v*) [virtual]

Allow/disallow additional checks on a source DataPoint before transfer. If set to true, extra checks will be performed in DataMover::Transfer() before data transfer starts on for example existence of the source file (and probably other checks too).

Parameters:

v true if allowed (default is true).

Implements Arc::DataPoint.

6.8.2.15 virtual void Arc::DataPointIndex::SetSecure (bool *v*) [virtual]

Allow/disallow heavy security (data encryption) during data transfer.

Parameters:

v true if allowed (default depends on protocol).

Implements Arc::DataPoint.

6.8.2.16 virtual void Arc::DataPointIndex::SortLocations (const std::string & *pattern*, const URLMap & *url_map*) [virtual]

Sort locations according to the specified pattern and URLMap. See DataMover::set_preferred_pattern for a more detailed explanation of pattern matching. Locations present in url_map are preferred over others.

Parameters:

pattern a set of strings, separated by |, to match against.

url_map map of URLs to local URLs

Implements Arc::DataPoint.

6.8.2.17 virtual DataStatus Arc::DataPointIndex::StartReading (DataBuffer & *buffer*) [virtual]

Start reading data from URL. A separate thread to transfer data will be created. No other operation can be performed while reading is in progress. Progress of the transfer should be followed using the DataBuffer object.

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned. If StopReading() is not called explicitly to release buffer it will be released in destructor of DataPoint which also usually calls StopReading().

Returns:

success if a thread was successfully started to start reading

Implements Arc::DataPoint.

6.8.2.18 virtual DataStatus Arc::DataPointIndex::StartWriting (DataBuffer & *buffer*, DataCallback * *space_cb* = NULL) [virtual]

Start writing data to URL. A separate thread to transfer data will be created. No other operation can be performed while writing is in progress. Progress of the transfer should be followed using the DataBuffer object.

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before StopWriting() was called and returned. If StopWriting() is not called explicitly to release buffer it will be released in destructor of DataPoint which also usually calls StopWriting().

space_cb callback which is called if there is not enough space to store data. May not implemented for all protocols.

Returns:

success if a thread was successfully started to start writing

Implements Arc::DataPoint.

6.8.2.19 virtual DataStatus Arc::DataPointIndex::StopReading () [virtual]

Stop reading. Must be called after corresponding StartReading() method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred.

Returns:

outcome of stopping reading (not outcome of transfer itself)

Implements Arc::DataPoint.

6.8.2.20 virtual DataStatus Arc::DataPointIndex::StopWriting () [virtual]

Stop writing. Must be called after corresponding StartWriting() method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred.

Returns:

outcome of stopping writing (not outcome of transfer itself)

Implements Arc::DataPoint.

6.8.2.21 virtual std::vector<URL> Arc::DataPointIndex::TransferLocations () const [virtual]

Returns physical file(s) to read/write, if different from CurrentLocation(). To be used with protocols which re-direct to different URLs such as Transport URLs (TURLs). The list is initially filled by PrepareReading and PrepareWriting. If this list is non-empty then real transfer should use a URL from this list. It is up to the caller to choose the best URL and instantiate new DataPoint for handling it. For consistency protocols which do not require redirections return original URL. For protocols which need redirection calling StartReading and StartWriting will use first URL in the list.

Reimplemented from Arc::DataPoint.

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

- DataPointIndex.h

6.9 Arc::DataSpeed Class Reference

Keeps track of average and instantaneous transfer speed.

`#include <arc/data/DataSpeed.h>`

Public Types

- typedef void(∗ show_progress_t)(FILE ∗o, const char ∗s, unsigned int t, unsigned long long int all, unsigned long long int max, double instant, double average)

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 verbose (bool val)
- void verbose (const std::string &prefix)
- bool verbose ()
- 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)
- time_t get_max_inactivity_time ()
- void set_base (time_t base_=DATASPEED_AVERAGING_PERIOD)
- void set_max_data (unsigned long long int max=0)
- void set_progress_indicator (show_progress_t func=NULL)
- void reset ()
- 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 transferred_size ()

6.9.1 Detailed Description

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

6.9.2 Member Typedef Documentation

6.9.2.1 typedef void(∗ Arc::DataSpeed::show_progress_t)(FILE ∗o, const char ∗s, unsigned int t, unsigned long long int all, unsigned long long int max, double instant, double average)

Callback for output of transfer status. A function with this signature can be passed to set_progress_indicator() to enable user-defined output of transfer progress.

Parameters:

o FILE object connected to stderr

s prefix set in verbose(const std::string&)

t time in seconds since the start of the transfer

all number of bytes transferred so far

max total amount of bytes to be transferred (set in set_max_data())

instant instantaneous transfer rate in bytes per second

average average transfer rate in bytes per second

6.9.3 Constructor & Destructor Documentation

6.9.3.1 Arc::DataSpeed::DataSpeed (time_t *base* = `DATASPEED_AVERAGING_PERIOD`)

Constructor.

Parameters:

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

6.9.3.2 Arc::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:

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

min_speed_time time over which to calculate min_speed.

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, error is triggered.

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

6.9.4 Member Function Documentation

6.9.4.1 void Arc::DataSpeed::set_max_inactivity_time (time_t *max_inactivity_time*)

Set inactivity timeout.

Parameters:

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

6.9.4.2 void Arc::DataSpeed::set_min_average_speed (unsigned long long int *min_average_speed*)

Set minimal average speed in bytes per second.

Parameters:

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

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

Set minimal allowed speed in bytes per second.

Parameters:

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

min_speed_time time over which to calculate min_speed.

6.9.4.4 void Arc::DataSpeed::set_progress_indicator (show_progress_t *func* = `NULL`)

Specify an external function to print verbose messages. If not specified an internal function is used.

Parameters:

func pointer to function which prints information.

6.9.4.5 bool Arc::DataSpeed::transfer (unsigned long long int *n* = `0`)

Inform object that an amount of data has been transferred. All errors are triggered by this method. To make them work the application must call this method periodically even with zero value.

Parameters:

n amount of data transferred in bytes.

Returns:

false if transfer rate is below limits

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

- DataSpeed.h

6.10 Arc::DataStatus Class Reference

Status code returned by many DataPoint methods.

`#include <arc/data/DataStatus.h>`

Public Types

- enum DataStatusType {

  Success, ReadAcquireError, WriteAcquireError, ReadResolveError,

  WriteResolveError, ReadStartError, WriteStartError, ReadError,

  WriteError, TransferError, ReadStopError, WriteStopError,

  PreRegisterError, PostRegisterError, UnregisterError, CacheError,

  CredentialsExpiredError, DeleteError, NoLocationError, LocationAlreadyExistsError,

  NotSupportedForDirectDataPointsError, UnimplementedError, IsReadingError, IsWritingError,

  CheckError, ListError, ListNonDirError, StatError,

  StatNotPresentError, NotInitializedError, SystemError, StageError,

  InconsistentMetadataError, ReadPrepareError, ReadPrepareWait, WritePrepareError,

  WritePrepareWait, ReadFinishError, WriteFinishError, CreateDirectoryError,

  RenameError, SuccessCached, SuccessCancelled, GenericError,

  UnknownError, ReadAcquireErrorRetryable = DataStatusRetryableBase+ReadAcquireError, WriteAcquireErrorRetryable = DataStatusRetryableBase+WriteAcquireError, ReadResolveErrorRetryable = DataStatusRetryableBase+ReadResolveError,

  WriteResolveErrorRetryable = DataStatusRetryableBase+WriteResolveError, ReadStartErrorRetryable = DataStatusRetryableBase+ReadStartError, WriteStartErrorRetryable = DataStatusRetryableBase+WriteStartError, ReadErrorRetryable = DataStatusRetryableBase+ReadError,

  WriteErrorRetryable = DataStatusRetryableBase+WriteError, TransferErrorRetryable = DataStatusRetryableBase+TransferError, ReadStopErrorRetryable = DataStatusRetryableBase+ReadStopError, WriteStopErrorRetryable = DataStatusRetryableBase+WriteStopError,

  PreRegisterErrorRetryable = DataStatusRetryableBase+PreRegisterError, PostRegisterErrorRetryable = DataStatusRetryableBase+PostRegisterError, UnregisterErrorRetryable = DataStatusRetryableBase+UnregisterError, CacheErrorRetryable = DataStatusRetryableBase+CacheError,

  DeleteErrorRetryable = DataStatusRetryableBase+DeleteError, CheckErrorRetryable = DataStatusRetryableBase+CheckError, ListErrorRetryable = DataStatusRetryableBase+ListError, StatErrorRetryable = DataStatusRetryableBase+StatError,

  StageErrorRetryable = DataStatusRetryableBase+StageError, ReadPrepareErrorRetryable = DataStatusRetryableBase+ReadPrepareError, WritePrepareErrorRetryable = DataStatusRetryableBase+WritePrepareError, ReadFinishErrorRetryable = DataStatusRetryableBase+ReadFinishError,

  WriteFinishErrorRetryable = DataStatusRetryableBase+WriteFinishError, CreateDirectoryErrorRetryable = DataStatusRetryableBase+CreateDirectoryError, RenameErrorRetryable = DataStatusRetryableBase+RenameError, GenericErrorRetryable = DataStatusRetryableBase+GenericError }

Public Member Functions

- DataStatus (const DataStatusType &status, std::string desc="")
- DataStatus (const DataStatusType &status, int error_no, const std::string &desc="")

- DataStatus ()
- bool operator== (const DataStatusType &s)
- bool operator== (const DataStatus &s)
- bool operator!= (const DataStatusType &s)
- bool operator!= (const DataStatus &s)
- DataStatus operator= (const DataStatusType &s)
- bool operator! () const
- operator bool () const
- bool Passed () const
- bool Retryable () const
- void SetErrno (int error_no)
- int GetErrno () const
- std::string GetStrErrno () const
- void SetDesc (const std::string &d)
- std::string GetDesc () const
- operator std::string (void) const

6.10.1 Detailed Description

Status code returned by many DataPoint methods. A class to be used for return types of all major data handling methods. It describes the outcome of the method and contains three fields: DataStatusType describes in which operation the error occurred, Errno describes why the error occurred and desc gives more detail if available. Errno is an integer corresponding to error codes defined in errno.h plus additional ARC-specific error codes defined here.

For those DataPoints which natively support errno, it is safe to use code like

```
DataStatus s = someMethod();
if (!s) {
  logger.msg(ERROR, "someMethod failed: %s", StrError(errno));
  return DataStatus(DataStatus::ReadError, errno);
}
```

since logger.msg() does not call any system calls that modify errno.

6.10.2 Member Enumeration Documentation

6.10.2.1 enum Arc::DataStatus::DataStatusType

Status codes. These codes describe in which operation an error occurred. Retryable error codes are deprecated - the corresponding non-retryable error code should be used with errno set to a retryable value.

Enumerator:

Success Operation completed successfully.

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

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

ReadResolveError Resolving of index service URL for source failed.

WriteResolveError Resolving of index service URL for destination failed.

ReadStartError Can't read from source.

WriteStartError Can't write to destination.

ReadError Failed while reading from source.

WriteError Failed while writing to destination.

TransferError Failed while transfering data (mostly timeout).

ReadStopError Failed while finishing reading from source.

WriteStopError Failed while finishing writing to destination.

PreRegisterError First stage of registration of index service URL failed.

PostRegisterError Last stage of registration of index service URL failed.

UnregisterError Unregistration of index service URL failed.

CacheError Error in caching procedure.

CredentialsExpiredError Error due to provided credentials are expired.

DeleteError Error deleting location or URL.

NoLocationError No valid location available.

LocationAlreadyExistsError No valid location available.

NotSupportedForDirectDataPointsError Operation has no sense for this kind of URL.

UnimplementedError Feature is unimplemented.

IsReadingError DataPoint is already reading.

IsWritingError DataPoint is already writing.

CheckError Access check failed.

ListError Directory listing failed.

Deprecated

ListNonDirError ListError with errno set to ENOTDIR should be used instead

StatError File/dir stating failed.

Deprecated

StatNotPresentError StatError with errno set to ENOENT should be used instead

NotInitializedError Object initialization failed.

SystemError Error in OS.

StageError Staging error.

InconsistentMetadataError Inconsistent metadata.

ReadPrepareError Can't prepare source.

ReadPrepareWait Wait for source to be prepared.

WritePrepareError Can't prepare destination.

WritePrepareWait Wait for destination to be prepared.

ReadFinishError Can't finish source.

WriteFinishError Can't finish destination.

CreateDirectoryError Can't create directory.

RenameError Can't rename URL.

SuccessCached Data was already cached.

SuccessCancelled Operation was cancelled successfully.

GenericError General error which doesn't fit any other error.

UnknownError Undefined.

Deprecated

ReadAcquireErrorRetryable

Deprecated

WriteAcquireErrorRetryable

Deprecated

ReadResolveErrorRetryable

Deprecated

WriteResolveErrorRetryable

Deprecated

ReadStartErrorRetryable

Deprecated

WriteStartErrorRetryable

Deprecated

ReadErrorRetryable

Deprecated

WriteErrorRetryable

Deprecated

TransferErrorRetryable

Deprecated

ReadStopErrorRetryable

Deprecated

WriteStopErrorRetryable

Deprecated

PreRegisterErrorRetryable

Deprecated

PostRegisterErrorRetryable

Deprecated

UnregisterErrorRetryable

Deprecated

CacheErrorRetryable

Deprecated

DeleteErrorRetryable

Deprecated

CheckErrorRetryable

Deprecated

ListErrorRetryable

Deprecated

StatErrorRetryable

Deprecated

StageErrorRetryable

Deprecated

ReadPrepareErrorRetryable

Deprecated

WritePrepareErrorRetryable

Deprecated

ReadFinishErrorRetryable

Deprecated

WriteFinishErrorRetryable

Deprecated

CreateDirectoryErrorRetryable

Deprecated

RenameErrorRetryable

Deprecated

GenericErrorRetryable

6.10.3 Constructor & Destructor Documentation

6.10.3.1 Arc::DataStatus::DataStatus (const DataStatusType & *status*, std::string *desc* = "") [inline]

Constructor to use when errno-like information is not available.

Parameters:

status error location

desc error description

References Passed().

6.10.3.2 Arc::DataStatus::DataStatus (const DataStatusType & *status*, int *error_no*, const std::string & *desc* = "") [inline]

Construct a new DataStatus with errno and optional text description. If the status is an error condition then error_no must be set to a non-zero value.

Parameters:

status error location

error_no errno

desc error description

6.10.4 Member Function Documentation

6.10.4.1 DataStatus Arc::DataStatus::operator= (const DataStatusType & *s*) [inline]

Assignment operator. Sets status type to s and errno to EARCOTHER if s is an error state.

References Passed().

6.10.4.2 bool Arc::DataStatus::Retryable () const

Returns true if the error was temporary and could be retried. Retryable error numbers are EAGAIN, EBUSY, ETIMEDOUT, EARCSVCTMP, EARCTRANSFERTIMEOUT, EARCCHECKSUM and EARCOTHER.

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

- DataStatus.h

6.11 Arc::FileCache Class Reference

FileCache provides an interface to all cache operations.

`#include <arc/data/FileCache.h>`

Public Member Functions

- FileCache (const std::string &cache_path, const std::string &id, uid_t job_uid, gid_t job_gid)
- FileCache (const std::vector< std::string > &caches, const std::string &id, uid_t job_uid, gid_t job_gid)
- FileCache (const std::vector< std::string > &caches, const std::vector< std::string > &remote_caches, const std::vector< std::string > &draining_caches, const std::string &id, uid_t job_uid, gid_t job_gid)
- FileCache ()
- bool Start (const std::string &url, bool &available, bool &is_locked, bool use_remote=true, bool delete_first=false)
- bool Stop (const std::string &url)
- bool StopAndDelete (const std::string &url)
- std::string File (const std::string &url)
- bool Link (const std::string &link_path, const std::string &url, bool copy, bool executable, bool holding_lock, bool &try_again)
- bool Release () const
- bool AddDN (const std::string &url, const std::string &DN, const Time &expiry_time)
- bool CheckDN (const std::string &url, const std::string &DN)
- bool CheckCreated (const std::string &url)
- Time GetCreated (const std::string &url)
- bool CheckValid (const std::string &url)
- Time GetValid (const std::string &url)
- bool SetValid (const std::string &url, const Time &val)
- operator bool ()
- bool operator== (const FileCache &a)

6.11.1 Detailed Description

FileCache provides an interface to all cache operations. When it is decided a file should be downloaded to the cache, Start() should be called, so that the cache file can be prepared and locked if necessary. If the file is already available it is not locked and Link() can be called immediately to create a hard link to a per-job directory in the cache and then soft link, or copy the file directly to the session directory so it can be accessed from the user's job. If the file is not available, Start() will lock it, then after downloading Link() can be called. Stop() must then be called to release the lock. If the transfer failed, StopAndDelete() can be called to clean up the cache file. After a job has finished, Release() should be called to remove the hard links created for that job.

Cache files are locked for writing using the FileLock class, which creates a lock file with the '.lock' suffix next to the cache file. If Start() is called and the cache file is not already available, it creates this lock and Stop() must be called to release it. All processes calling Start() must wait until they successfully obtain the lock before downloading can begin.

The cache directory(ies) and the optional directory to link to when the soft-links are made are set in the constructor. The names of cache files are formed from an SHA-1 hash of the URL to cache. To

ease the load on the file system, the cache files are split into subdirectories based on the first two characters in the hash. For example the file with hash 76f11edda169848038efbd9fa3df5693 is stored in 76/f11edda169848038efbd9fa3df5693. A cache filename can be found by passing the URL to Find(). For more information on the structure of the cache, see the ARC Computing Element System Administrator Guide (NORDUGRID-MANUAL-20).

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Arc::FileCache::FileCache (const std::string & *cache_path*, const std::string & *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new FileCache instance with one cache directory.

Parameters:

cache_path The format is "cache_dir[link_path]". path is the path to the cache directory and the optional link_path is used to create a link in case the cache directory is visible under a different name during actual usage. When linking from the session dir this path is used instead of cache_-path.

id the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

job_uid owner of job. The per-job dir will only be readable by this user

job_gid owner group of job

6.11.2.2 Arc::FileCache::FileCache (const std::vector< std::string > & *caches*, const std::string & *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new FileCache instance with multiple cache dirs.

Parameters:

caches a vector of strings describing caches. The format of each string is "cache_dir[link_path]".

id the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

job_uid owner of job. The per-job dir will only be readable by this user

job_gid owner group of job

6.11.2.3 Arc::FileCache::FileCache (const std::vector< std::string > & *caches*, const std::vector< std::string > & *remote_caches*, const std::vector< std::string > & *draining_caches*, const std::string & *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new FileCache instance with multiple cache dirs, remote caches and draining cache directories.

Parameters:

caches a vector of strings describing caches. The format of each string is "cache_dir[link_path]".

remote_caches Same format as caches. These are the paths to caches which are under the control of other Grid Managers and are read-only for this process.

draining_caches Same format as caches. These are the paths to caches which are to be drained.

id the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

job_uid owner of job. The per-job dir will only be readable by this user

job_gid owner group of job

6.11.3 Member Function Documentation

6.11.3.1 bool Arc::FileCache::AddDN (const std::string & *url*, const std::string & *DN*, const Time & *expiry_time*)

Store a DN in the permissions cache for the given url. Add the given DN to the list of cached DNs with the given expiry time.

Parameters:

url the url corresponding to the cache file to which we want to add a cached DN

DN the DN of the user

expiry_time the expiry time of this DN in the DN cache

Returns:

true if the DN was successfully added

6.11.3.2 bool Arc::FileCache::CheckCreated (const std::string & *url*)

Check if it is possible to obtain the creation time of a cache file.

Parameters:

url the url corresponding to the cache file for which we want to know if the creation date exists

Returns:

true if the file exists in the cache, since the creation time is the creation time of the cache file.

6.11.3.3 bool Arc::FileCache::CheckDN (const std::string & *url*, const std::string & *DN*)

Check if a DN exists in the permission cache and is still valid for the given url. Check if the given DN is cached for authorisation and it is still valid.

Parameters:

url the url corresponding to the cache file for which we want to check the cached DN

DN the DN of the user

Returns:

true if the DN exists and is still valid

6.11.3.4 bool Arc::FileCache::CheckValid (const std::string & *url*)

Check if there is an expiry time of the given url in the cache.

Parameters:

url the url corresponding to the cache file for which we want to know if the expiration time exists

Returns:

true if an expiry time exists

6.11.3.5 std::string Arc::FileCache::File (const std::string & *url*)

Get the cache filename for the given URL.

Parameters:

url the URL to look for in the cache

Returns:

the full pathname of the file in the cache which corresponds to the given url.

6.11.3.6 Time Arc::FileCache::GetCreated (const std::string & *url*)

Get the creation time of a cached file.

Parameters:

url the url corresponding to the cache file for which we want to know the creation date

Returns:

creation time of the file or 0 if the cache file does not exist

6.11.3.7 Time Arc::FileCache::GetValid (const std::string & *url*)

Get expiry time of a cached file.

Parameters:

url the url corresponding to the cache file for which we want to know the expiry time

Returns:

the expiry time or 0 if none is available

6.11.3.8 bool Arc::FileCache::Link (const std::string & *link_path*, const std::string & *url*, bool *copy*, bool *executable*, bool *holding_lock*, bool & *try_again*)

Link a cache file to the place it will be used. Create a hard-link to the per-job dir from the cache dir, and then a soft-link from here to the session directory. This is effectively 'claiming' the file for the job, so even if the original cache file is deleted, eg by some external process, the hard link still exists until it is explicitly released by calling Release().

If cache_link_path is set to "." or copy or executable is true then files will be copied directly to the session directory rather than linked.

After linking or copying, the cache file is checked for the presence of a write lock, and whether the modification time has changed since linking started (in case the file was locked, modified then released during linking). If either of these are true the links created during Link() are deleted, try_again is set to true and Link() returns false. The caller should then go back to Start(). If the caller has obtained a write lock from Start() and then downloaded the file, it should set holding_lock to true, in which case none of the above checks are performed.

The session directory is accessed under the uid and gid passed in the constructor.

Parameters:

link_path path to the session dir for soft-link or new file

url url of file to link to or copy

copy If true the file is copied rather than soft-linked to the session dir

executable If true then file is copied and given execute permissions in the session dir

holding_lock Should be set to true if the caller already holds the lock

try_again If after linking the cache file was found to be locked, deleted or modified, then try_again is set to true

Returns:

true if linking succeeded, false if an error occurred or the file was locked or modified by another process during linking

6.11.3.9 bool Arc::FileCache::Release () const

Release cache files used in this cache. Release claims on input files for the job specified by id. For each cache directory the per-job directory with the hard-links will be deleted.

Returns:

false if any directory fails to be deleted

6.11.3.10 bool Arc::FileCache::SetValid (const std::string & *url*, const Time & *val*)

Set expiry time of a cache file.

Parameters:

url the url corresponding to the cache file for which we want to set the expiry time

val expiry time

Returns:

true if the expiry time was successfully set

6.11.3.11 bool Arc::FileCache::Start (const std::string & *url*, bool & *available*, bool & *is_locked*, bool *use_remote* = `true`, bool *delete_first* = `false`)

Start preparing to cache the file specified by url. Start() returns true if the file was successfully prepared. The available parameter is set to true if the file already exists and in this case Link() can be called immediately. If available is false the caller should write the file and then call Link() followed by Stop(). Start() returns false if it was unable to prepare the cache file for any reason. In this case the is_locked parameter should be checked and if it is true the file is locked by another process and the caller should try again later.

Parameters:

url url that is being downloaded

available true on exit if the file is already in cache

is_locked true on exit if the file is already locked, ie cannot be used by this process

use_remote Whether to look to see if the file exists in a remote cache. Can be set to false if for example a forced download to cache is desired.

delete_first If true then any existing cache file is deleted.

Returns:

true if file is available or ready to be downloaded, false if the file is already locked or preparing the cache failed.

6.11.3.12 bool Arc::FileCache::Stop (const std::string & *url*)

Stop the cache after a file was downloaded. This method (or stopAndDelete()) must be called after file was downloaded or download failed, to release the lock on the cache file. Stop() does not delete the cache file. It returns false if the lock file does not exist, or another pid was found inside the lock file (this means another process took over the lock so this process must go back to Start()), or if it fails to delete the lock file. It must only be called if the caller actually downloaded the file. It must not be called if the file was already available.

Parameters:

url the url of the file that was downloaded

Returns:

true if the lock was successfully released.

6.11.3.13 bool Arc::FileCache::StopAndDelete (const std::string & *url*)

Stop the cache after a file was downloaded and delete the cache file. Release the cache file and delete it, because for example a failed download left an incomplete copy. This method also deletes the meta file which contains the url corresponding to the cache file. The logic of the return value is the same as Stop(). It must only be called if the caller downloaded the file.

Parameters:

url the url corresponding to the cache file that has to be released and deleted

Returns:

true if the cache file and lock were successfully removed.

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

- FileCache.h

6.12 Arc::FileCacheHash Class Reference

FileCacheHash provides methods to make hashes from strings.

`#include <arc/data/FileCacheHash.h>`

Static Public Member Functions

- static std::string getHash (std::string url)
- static int maxLength ()

6.12.1 Detailed Description

FileCacheHash provides methods to make hashes from strings. Currently the SHA-1 hash from the openssl library is used.

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

- FileCacheHash.h

6.13 Arc::FileInfo Class Reference

FileInfo stores information about files (metadata).

`#include <arc/data/FileInfo.h>`

Public Types

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

Public Member Functions

- FileInfo (const std::string &name="")
- const std::string & GetName () const
- std::string GetLastName () const
- void SetName (const std::string &n)
- const std::list< URL > & GetURLs () const
- void AddURL (const URL &u)
- bool CheckSize () const
- unsigned long long int GetSize () const
- void SetSize (const unsigned long long int s)
- bool CheckCheckSum () const
- const std::string & GetCheckSum () const
- void SetCheckSum (const std::string &c)
- bool CheckModified () const
- Time GetModified () const
- void SetModified (const Time &t)
- bool CheckValid () const
- Time GetValid () const
- void SetValid (const Time &t)
- bool CheckType () const
- Type GetType () const
- void SetType (const Type t)
- bool CheckLatency () const
- std::string GetLatency () const
- void SetLatency (const std::string l)
- std::map< std::string, std::string > GetMetaData () const
- void SetMetaData (const std::string att, const std::string val)
- bool operator< (const FileInfo &f) const
- operator bool () const
- bool operator! () const

6.13.1 Detailed Description

FileInfo stores information about files (metadata). Set/Get methods exist for "standard" metadata such as name, size and modification time, and there is a generic key-value map for protocol-specific attributes. The Set methods always set the corresponding entry in the generic map, so there is no need for a caller make two calls, for example SetSize(1) followed by SetMetaData("size", "1").

6.13.2 Member Enumeration Documentation

6.13.2.1 enum Arc::FileInfo::Type

Type of file object.

Enumerator:

file_type_unknown Unknown.

file_type_file File-type.

file_type_dir Directory-type.

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

- FileInfo.h

6.14 Arc::URLMap Class Reference

URLMap allows mapping certain patterns of URLs to other URLs.

`#include <arc/data/URLMap.h>`

Data Structures

- class **map_entry**

Public Member Functions

- URLMap ()
- bool map (URL &url) const
- bool local (const URL &url) const
- void add (const URL &templ, const URL &repl, const URL &accs=URL())
- operator bool () const
- bool operator! () const

6.14.1 Detailed Description

URLMap allows mapping certain patterns of URLs to other URLs. A URLMap can be used if certain URLs can be more efficiently accessed by other means on a certain site. For example a GridFTP storage element may be mounted as a local file system and so a map can be made from a gsiftp:// URL to a local file path.

6.14.2 Member Function Documentation

6.14.2.1 void Arc::URLMap::add (const URL & *templ*, const URL & *repl*, const URL & *accs* = `URL()`)

Add an entry to the URLMap. All URLs matching templ will have the templ part replaced by repl.

Parameters:

templ template to replace, for example gsiftp://se.org/files

repl replacement for template, for example /export/grid/files

accs replacement path if it differs in the place the file will actually be accessed (e.g. on worker nodes), for example /mount/grid/files

6.14.2.2 bool Arc::URLMap::local (const URL & *url*) const

Check if a mapping exists for a URL. Checks to see if a URL will be mapped but does not do the mapping.

Parameters:

url URL to check

Returns:

true if a mapping exists for this URL

6.14.2.3 bool Arc::URLMap::map (URL & *url*) const

Map a URL if possible. If the given URL matches any template it will be changed to the mapped URL. Additionally, if the mapped URL is a local file, a permission check is done by attempting to open the file. If a different access path is specified for this URL the URL will be changed to link://accesspath. To check if a URL will be mapped without changing it local() can be used.

Parameters:

url URL to check

Returns:

true if the URL was mapped to a new URL, false if it was not mapped or an error occurred during mapping

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

- URLMap.h

Index