ARC Data Library libarcdata Reference Manual

Generated by Doxygen 1.4.7

Thu Jul 14 13:38:48 2011

Contents

Chapter 1

Summary of libarcdata

libarcdata is a library for data access. It provides a uniform interface to several types of grid storage and catalogs using various protocols. See the DataPoint inheritance diagram for a list of currently supported protocols. 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). To create a new DMC for a protocol which is not yet supported see the instruction and examples in the DataPoint class documentation. This also gives a complete overview of the interface.

Chapter 2

ARC Data Library libarcdata Hierarchical Index

2.1 ARC Data Library libarcdata Class Hierarchy

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

Chapter 3

ARC Data Library libarcdata Data Structure Index

3.1 ARC Data Library libarcdata Data Structures

Here are the data structures with brief descriptions:

Chapter 4

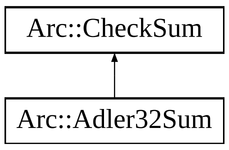
ARC Data Library libarcdata Data Structure Documentation

4.1 Arc::Adler32Sum Class Reference

Implementation of Adler32 checksum.

`#include <CheckSum.h>`

Inheritance diagram for Arc::Adler32Sum::



4.1.1 Detailed Description

Implementation of Adler32 checksum.

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

- CheckSum.h

4.2 Arc::CacheParameters Struct Reference

Contains data on the parameters of a cache.

```
#include <FileCache.h>
```

4.2.1 Detailed Description

Contains data on the parameters of a cache.

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

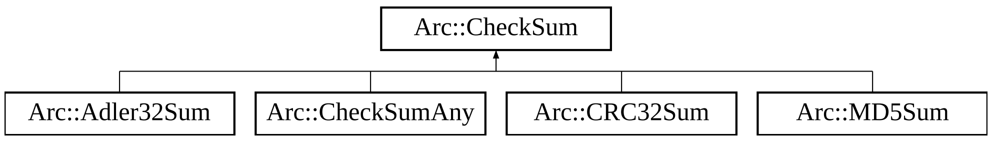
- FileCache.h

4.3 Arc::CheckSum Class Reference

Defines interface for various checksum manipulations.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSum::



4.3.1 Detailed Description

Defines interface for various checksum manipulations.

This class is used during data transfers through DataBuffer class

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

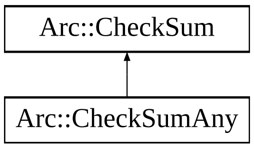
- CheckSum.h

4.4 Arc::CheckSumAny Class Reference

Wrapper for CheckSum class.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSumAny::



4.4.1 Detailed Description

Wrapper for CheckSum class.

To be used for manipulation of any supported checksum type in a transparent way.

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

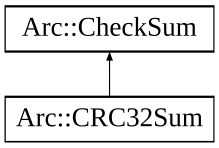
- CheckSum.h

4.5 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

```
#include <CheckSum.h>
```

Inheritance diagram for Arc::CRC32Sum::



4.5.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

4.6 Arc::DataBuffer Class Reference

Represents set of buffers.

`#include <DataBuffer.h>`

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 () 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

Data Structures

- struct **buf_desc**
- class **checksum_desc**

4.6.1 Detailed Description

Represents set of buffers.

This class is used used during data transfer using DataPoint classes.

4.6.2 Constructor & Destructor Documentation

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

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

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

Contructor

Parameters:

size size of every buffer in bytes.

blocks number of buffers.

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

4.6.2.3 Arc::DataBuffer::∼DataBuffer ()

Destructor.

4.6.3 Member Function Documentation

4.6.3.1 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 till DataBuffer itself.

Returns:

integer position in the list of checksum objects.

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

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

4.6.3.3 const CheckSum∗ Arc::DataBuffer::checksum_object () const

Returns CheckSum object specified in constructor, returns NULL if index is not in list.

Parameters:

index of the checksum in question.

4.6.3.4 bool Arc::DataBuffer::checksum_valid () const

Returns true if checksum was successfully computed, returns false if index is not in list.

Parameters:

index of the checksum in question.

4.6.3.5 unsigned long long int Arc::DataBuffer::eof_position () const `[inline]`

Returns offset following last piece of data transferred.

4.6.3.6 bool Arc::DataBuffer::eof_read ()

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

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

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

4.6.3.8 bool Arc::DataBuffer::eof_write ()

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

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

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

4.6.3.10 bool Arc::DataBuffer::error ()

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

4.6.3.11 bool Arc::DataBuffer::error_read ()

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

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

Informs object if error accured on 'read' side.

Parameters:

v true if error.

4.6.3.13 bool Arc::DataBuffer::error_transfer ()

Returns true if eror occured inside object.

4.6.3.14 bool Arc::DataBuffer::error_write ()

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

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

Informs object if error accured on 'write' side.

Parameters:

v true if error.

4.6.3.16 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.

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

Request buffer for READING INTO it.

Parameters:

handle returns buffer's number.

length returns size of buffer

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

Returns:

true on success

4.6.3.18 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.

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

Request buffer for WRITING FROM it.

Parameters:

handle returns buffer's number.

length returns size of buffer

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

4.6.3.20 bool Arc::DataBuffer::is_notwritten (char ∗ *buf*)

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

Parameters:

buf - address of buffer

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

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

Parameters:

handle buffer's number.

4.6.3.22 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.

4.6.3.23 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.

4.6.3.24 bool Arc::DataBuffer::is_written (char ∗ *buf*)

Informs object that data was written from buffer.

Parameters:

buf - address of buffer

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

Informs object that data was written from buffer.

Parameters:

handle buffer's number.

4.6.3.26 Arc::DataBuffer::operator bool (void) const `[inline]`

Check if DataBuffer object is initialized.

4.6.3.27]

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

Direct access to buffer by number.

4.6.3.28 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 till DataBuffer itself.

4.6.3.29 bool Arc::DataBuffer::wait_any ()

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

4.6.3.30 bool Arc::DataBuffer::wait_eof ()

Wait till end of transfer happens on any side.

4.6.3.31 bool Arc::DataBuffer::wait_eof_read ()

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

4.6.3.32 bool Arc::DataBuffer::wait_eof_write ()

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

4.6.3.33 bool Arc::DataBuffer::wait_for_read ()

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

4.6.3.34 bool Arc::DataBuffer::wait_for_write ()

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

4.6.3.35 bool Arc::DataBuffer::wait_read ()

Wait till end of transfer or error happens on 'read' side.

4.6.3.36 bool Arc::DataBuffer::wait_used ()

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

4.6.3.37 bool Arc::DataBuffer::wait_write ()

Wait till end of transfer or error happens on 'write' side.

4.6.4 Field Documentation

4.6.4.1 DataSpeed Arc::DataBuffer::speed

This object controls transfer speed.

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

- DataBuffer.h

4.7 Arc::DataCallback Class Reference

This class is used by DataHandle to report missing space on local filesystem.

```
#include <DataCallback.h>
```

4.7.1 Detailed Description

This class is used by DataHandle to report missing space on local filesystem.

One of 'cb' functions here will be called if operation initiated by DataHandle::StartReading runs out of disk space.

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

- DataCallback.h

4.8 Arc::DataHandle Class Reference

This class is a wrapper around the DataPoint class.

`#include <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

4.8.1 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.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Arc::DataHandle::DataHandle (const URL & *url*, const UserConfig & *usercfg*) `[inline]`

Construct a new DataHandle.

4.8.2.2 Arc::DataHandle::∼DataHandle () `[inline]`

Destructor.

4.8.3 Member Function Documentation

4.8.3.1 const DataPoint& Arc::DataHandle::operator ∗ () const `[inline]`

Returns a const reference to a DataPoint object.

4.8.3.2 DataPoint& Arc::DataHandle::operator ∗ () `[inline]`

Returns a reference to a DataPoint object.

4.8.3.3 Arc::DataHandle::operator bool (void) const `[inline]`

Returns true if the DataHandle is valid.

4.8.3.4 bool Arc::DataHandle::operator! (void) const `[inline]`

Returns true if the DataHandle is not valid.

4.8.3.5 const DataPoint∗ Arc::DataHandle::operator → () const `[inline]`

Returns a const pointer to a DataPoint object.

4.8.3.6 DataPoint∗ Arc::DataHandle::operator → () `[inline]`

Returns a pointer to a DataPoint object.

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

- DataHandle.h

4.9 Arc::DataMover Class Reference

DataMover provides an interface to transfer data between two DataPoints.

```
#include <DataMover.h>
```

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)
- 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)

4.9.1 Detailed Description

DataMover provides an interface to transfer data between two DataPoints.

Its main action is represented by Transfer methods

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Arc::DataMover::DataMover ()

Constructor.

4.9.2.2 Arc::DataMover::∼DataMover ()

Destructor.

4.9.3 Member Function Documentation

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

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

Parameters:

v true if allowed (default is true).

4.9.3.2 bool Arc::DataMover::checks ()

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

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

Delete the file at url.

4.9.3.4 void Arc::DataMover::force_to_meta (bool)

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

4.9.3.5 void Arc::DataMover::passive (bool)

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

4.9.3.6 void Arc::DataMover::retry (bool)

Set if transfer will be retried in case of failure.

4.9.3.7 bool Arc::DataMover::retry ()

Check if transfer will be retried in case of failure.

4.9.3.8 void Arc::DataMover::secure (bool)

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

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

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

4.9.3.10 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.

4.9.3.11 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.

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

Set a preferred pattern for ordering of replicas.

4.9.3.13 void Arc::DataMover::set_progress_indicator (DataSpeed::show_progress_t *func* = `NULL`) `[inline]`

Set function which is called every second during the transfer.

4.9.3.14 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'.

Parameters:

min_speed minimal allowed current speed.

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

min_average_speed minimal allowed average speed.

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

4.9.3.15 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'.

Parameters:

source source URL.

destination destination URL.

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

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

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.

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

Activate printing information about transfer status.

Parameters:

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

4.9.3.17 void Arc::DataMover::verbose (bool)

Activate printing information about transfer status.

4.9.3.18 bool Arc::DataMover::verbose ()

Check if printing information about transfer status is activated.

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

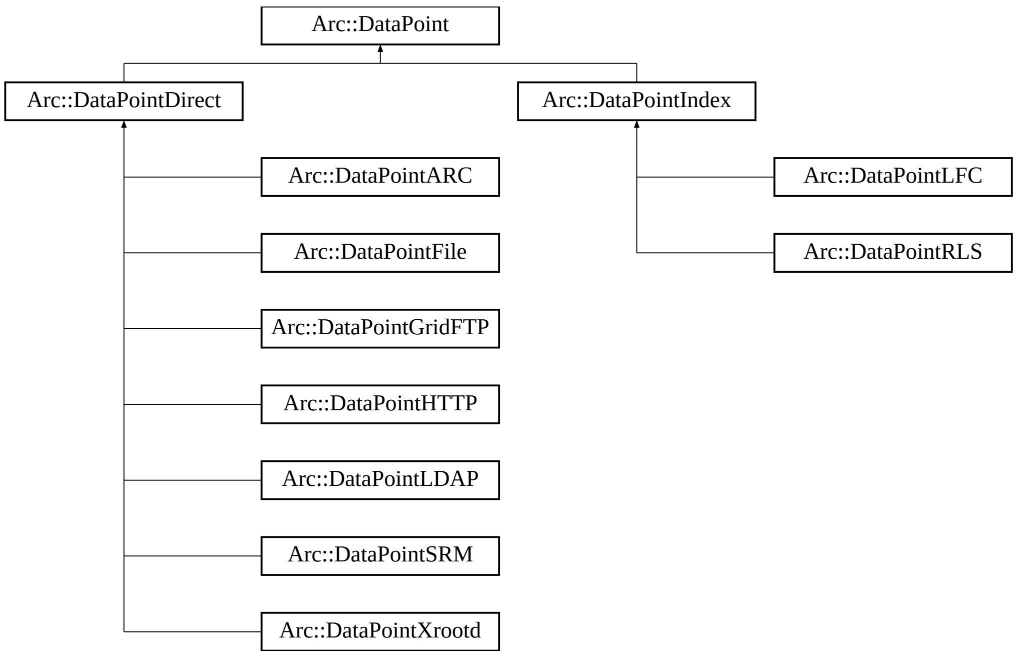
- DataMover.h

4.10 Arc::DataPoint Class Reference

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

`#include <DataPoint.h>`

Inheritance diagram for Arc::DataPoint::



Public Types

- ACCESS_LATENCY_ZERO
- ACCESS_LATENCY_SMALL
- ACCESS_LATENCY_LARGE
- 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
- 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 }

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, const std::list< std::string > &transport_protocols)
- virtual DataStatus PrepareWriting (unsigned int timeout, unsigned int &wait_time, const std::list< std::string > &transport_protocols)
- 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 ()=0
- virtual DataStatus Remove ()=0
- virtual DataStatus Stat (FileInfo &file, DataPointInfoType verb=INFO_TYPE_ALL)=0
- virtual DataStatus List (std::list< FileInfo > &files, DataPointInfoType verb=INFO_TYPE_ALL)=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 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 CheckCreated () const
- virtual void SetCreated (const Time &val)
- virtual const Time & GetCreated () 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 int GetTries () const
- virtual void SetTries (const int n)
- virtual void NextTry (void)
- 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 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

Protected Member Functions

- DataPoint (const URL &url, const UserConfig &usercfg)

Protected Attributes

- std::list< std::string > valid_url_options

4.10.1 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.

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);
  static Plugin* Instance(PluginArgument *arg);
  virtual DataStatus StartReading(DataBuffer& buffer);
  ...
};

DataPointMyProtocol::DataPointMyProtocol(const URL& url, const UserConfig& usercfg) {
  ...
}

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);
}

} // namespace Arc

Arc::PluginDescriptor PLUGINS_TABLE_NAME[] = {
  { "my", "HED:DMC", 0, &Arc::DataPointMyProtocol::Instance },
  { NULL, NULL, 0, NULL }
};
```

4.10.2 Member Enumeration Documentation

4.10.2.1 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.

4.10.2.2 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.

4.10.3 Constructor & Destructor Documentation

4.10.3.1 virtual Arc::DataPoint::∼DataPoint () `[virtual]`

Destructor.

4.10.3.2 Arc::DataPoint::DataPoint (const URL & *url*, const UserConfig & *usercfg*) `[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

4.10.4 Member Function Documentation

4.10.4.1 virtual bool Arc::DataPoint::AcceptsMeta () const `[pure virtual]`

If endpoint can have any use from meta information.

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

4.10.4.2 virtual int Arc::DataPoint::AddCheckSumObject (CheckSum ∗ *cksum*) `[pure virtual]`

Add a checksum object which will compute checksum during transmission.

Parameters:

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

Returns:

integer position in the list of checksum objects.

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

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

Add URL to list.

Parameters:

url Location URL to add.

meta Location meta information.

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

4.10.4.4 virtual int Arc::DataPoint::BufNum () const `[pure virtual]`

Get suggested number of buffers for transfers.

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

4.10.4.5 virtual long long int Arc::DataPoint::BufSize () const `[pure virtual]`

Get suggested buffer size for transfers.

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

4.10.4.6 virtual bool Arc::DataPoint::Cache () const `[virtual]`

Returns true if file is cacheable.

4.10.4.7 virtual DataStatus Arc::DataPoint::Check () `[pure virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implemented in Arc::DataPointIndex, Arc::DataPointARC, Arc::DataPointFile, Arc::DataPointGridFTP, Arc::DataPointHTTP, Arc::DataPointLDAP, Arc::DataPointLFC, Arc::DataPointRLS, Arc::DataPoint-SRM, and Arc::DataPointXrootd.

4.10.4.8 virtual bool Arc::DataPoint::CheckCheckSum () const `[virtual]`

Check if meta-information 'checksum' is available.

4.10.4.9 virtual bool Arc::DataPoint::CheckCreated () const `[virtual]`

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

4.10.4.10 virtual bool Arc::DataPoint::CheckSize () const `[virtual]`

Check if meta-information 'size' is available.

4.10.4.11 virtual bool Arc::DataPoint::CheckValid () const `[virtual]`

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

4.10.4.12 virtual DataStatus Arc::DataPoint::ClearLocations () `[pure virtual]`

Remove all locations.

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

4.10.4.13 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.

4.10.4.14 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.

4.10.4.15 virtual const URL& Arc::DataPoint::CurrentLocation () const `[pure virtual]`

Returns current (resolved) URL.

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

4.10.4.16 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.

4.10.4.17 virtual const std::string Arc::DataPoint::DefaultCheckSum () const `[virtual]`

Default checksum type.

Reimplemented in Arc::DataPointGridFTP, Arc::DataPointLFC, and Arc::DataPointSRM.

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

Finish reading from the URL.

Must be called after transfer of physical file has completed and 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.

Reimplemented in Arc::DataPointIndex, and Arc::DataPointSRM.

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

Finish writing to the URL.

Must be called after transfer of physical file has completed and 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.

Reimplemented in Arc::DataPointIndex, and Arc::DataPointSRM.

4.10.4.20 virtual DataPointAccessLatency Arc::DataPoint::GetAccessLatency () const `[virtual]`

Get value of meta-information 'access latency'.

Reimplemented in Arc::DataPointIndex.

4.10.4.21 virtual bool Arc::DataPoint::GetAdditionalChecks () const `[pure virtual]`

Check if additional checks before transfer will be performed.

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

4.10.4.22 virtual const std::string& Arc::DataPoint::GetCheckSum () const `[virtual]`

Get value of meta-information 'checksum'.

4.10.4.23 virtual const CheckSum∗ Arc::DataPoint::GetCheckSumObject (int *index*) const `[pure virtual]`

Get CheckSum object at given position in list.

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

4.10.4.24 virtual const Time& Arc::DataPoint::GetCreated () const `[virtual]`

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

4.10.4.25 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.

4.10.4.26 virtual bool Arc::DataPoint::GetSecure () const `[pure virtual]`

Check if heavy security during data transfer is allowed.

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

4.10.4.27 virtual unsigned long long int Arc::DataPoint::GetSize () const `[virtual]`

Get value of meta-information 'size'.

4.10.4.28 virtual int Arc::DataPoint::GetTries () const `[virtual]`

Returns number of retries left.

4.10.4.29 virtual const URL& Arc::DataPoint::GetURL () const `[virtual]`

Returns the URL that was passed to the constructor.

4.10.4.30 virtual const UserConfig& Arc::DataPoint::GetUserConfig () const `[virtual]`

Returns the UserConfig that was passed to the constructor.

4.10.4.31 virtual const Time& Arc::DataPoint::GetValid () const `[virtual]`

Get value of meta-information 'validity time'.

4.10.4.32 virtual bool Arc::DataPoint::HaveLocations () const `[pure virtual]`

Returns true if number of resolved URLs is not 0.

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

4.10.4.33 virtual bool Arc::DataPoint::IsIndex () const `[pure virtual]`

Check if URL is an Indexing Service.

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

4.10.4.34 virtual bool Arc::DataPoint::IsStageable () const `[virtual]`

If URL should be staged or queried for Transport URL (TURL).

Reimplemented in Arc::DataPointDirect, Arc::DataPointIndex, and Arc::DataPointSRM.

4.10.4.35 virtual bool Arc::DataPoint::LastLocation () `[pure virtual]`

Returns true if the current location is the last.

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

4.10.4.36 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.

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.

Implemented in Arc::DataPointARC, and Arc::DataPointLDAP.

4.10.4.37 virtual bool Arc::DataPoint::Local () const `[pure virtual]`

Returns true if file is local, e.g. file:// urls.

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

4.10.4.38 virtual bool Arc::DataPoint::LocationValid () const `[pure virtual]`

Returns false if out of retries.

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

4.10.4.39 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.

4.10.4.40 virtual void Arc::DataPoint::NextTry (void) `[virtual]`

Decrease number of retries left.

4.10.4.41 virtual Arc::DataPoint::operator bool () const `[virtual]`

Is DataPoint valid?

4.10.4.42 virtual bool Arc::DataPoint::operator! () const `[virtual]`

Is DataPoint valid?

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

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

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

Index Service postregistration.

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

Parameters:

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

Implemented in Arc::DataPointDirect, Arc::DataPointLFC, and Arc::DataPointRLS.

4.10.4.45 virtual DataStatus Arc::DataPoint::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented in Arc::DataPointIndex, and Arc::DataPointSRM.

4.10.4.46 virtual DataStatus Arc::DataPoint::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented in Arc::DataPointIndex, and Arc::DataPointSRM.

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

Index service preregistration.

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 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.

Implemented in Arc::DataPointDirect, Arc::DataPointLFC, and Arc::DataPointRLS.

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

Index Service preunregistration.

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 same name.

Implemented in Arc::DataPointDirect, Arc::DataPointLFC, and Arc::DataPointRLS.

4.10.4.49 virtual bool Arc::DataPoint::ProvidesMeta () const `[pure virtual]`

If endpoint can provide at least some meta information directly.

Implemented in Arc::DataPointDirect, Arc::DataPointIndex, Arc::DataPointGridFTP, and Arc::DataPoint-SRM.

4.10.4.50 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.

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

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

Parameters:

v true if allowed (default is false).

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

4.10.4.52 virtual bool Arc::DataPoint::Registered () const `[pure virtual]`

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

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

4.10.4.53 virtual DataStatus Arc::DataPoint::Remove () `[pure virtual]`

Remove/delete object at URL.

Implemented in Arc::DataPointIndex, Arc::DataPointARC, Arc::DataPointFile, Arc::DataPointGridFTP, Arc::DataPointHTTP, Arc::DataPointLDAP, Arc::DataPointSRM, and Arc::DataPointXrootd.

4.10.4.54 virtual DataStatus Arc::DataPoint::RemoveLocation () `[pure virtual]`

Remove current URL from list.

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

4.10.4.55 virtual DataStatus Arc::DataPoint::RemoveLocations (const DataPoint & *p*) `[pure virtual]`

Remove locations present in another DataPoint object.

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

4.10.4.56 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.

Parameters:

source true if DataPoint object represents source of information.

Implemented in Arc::DataPointDirect, Arc::DataPointLFC, and Arc::DataPointRLS.

4.10.4.57 virtual void Arc::DataPoint::SetAccessLatency (const DataPointAccessLatency & *latency*) `[virtual]`

Set value of meta-information ’access latency’.

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

Allow/disallow additional checks.

Check for existence of remote file (and probably other checks too) before initiating reading and writing operations.

Parameters:

v true if allowed (default is true).

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

4.10.4.59 virtual void Arc::DataPoint::SetCheckSum (const std::string & *val*) `[virtual]`

Set value of meta-information ’checksum’.

Reimplemented in Arc::DataPointIndex.

4.10.4.60 virtual void Arc::DataPoint::SetCreated (const Time & *val*) `[virtual]`

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

4.10.4.61 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.

Reimplemented in Arc::DataPointIndex.

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

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default depends on protocol).

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

4.10.4.63 virtual void Arc::DataPoint::SetSize (const unsigned long long int *val*) `[virtual]`

Set value of meta-information 'size'.

Reimplemented in Arc::DataPointIndex.

4.10.4.64 virtual void Arc::DataPoint::SetTries (const int *n*) `[virtual]`

Set number of retries.

Reimplemented in Arc::DataPointIndex.

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

Assigns new URL. Main purpose of this method is to reuse existing connection for accessing different object at same server. Implementation does not have to implement this method. If supplied URL is not suitable or method is not implemented false is returned.

Reimplemented in Arc::DataPointGridFTP, and Arc::DataPointHTTP.

4.10.4.66 virtual void Arc::DataPoint::SetValid (const Time & *val*) `[virtual]`

Set value of meta-information 'validity time'.

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

Sort locations according to the specified pattern.

Parameters:

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

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

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

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implemented in Arc::DataPointIndex, Arc::DataPointARC, Arc::DataPointFile, Arc::DataPointGridFTP, Arc::DataPointHTTP, Arc::DataPointLDAP, Arc::DataPointSRM, and Arc::DataPointXrootd.

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

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implemented in Arc::DataPointIndex, Arc::DataPointARC, Arc::DataPointFile, Arc::DataPointGridFTP, Arc::DataPointHTTP, Arc::DataPointLDAP, Arc::DataPointSRM, and Arc::DataPointXrootd.

4.10.4.70 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.

Implemented in Arc::DataPointARC, and Arc::DataPointLDAP.

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

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implemented in Arc::DataPointIndex, Arc::DataPointARC, Arc::DataPointFile, Arc::DataPointGridFTP, Arc::DataPointHTTP, Arc::DataPointLDAP, Arc::DataPointSRM, and Arc::DataPointXrootd.

4.10.4.72 virtual DataStatus Arc::DataPoint::StopWriting () `[pure virtual]`

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implemented in Arc::DataPointIndex, Arc::DataPointARC, Arc::DataPointFile, Arc::DataPointGridFTP, Arc::DataPointHTTP, Arc::DataPointLDAP, Arc::DataPointSRM, and Arc::DataPointXrootd.

4.10.4.73 virtual std::string Arc::DataPoint::str () const `[virtual]`

Returns a string representation of the DataPoint.

Reimplemented in Arc::DataPointLFC.

4.10.4.74 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, and Arc::DataPointSRM.

4.10.4.75 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 is unregistered.

Implemented in Arc::DataPointDirect, Arc::DataPointLFC, and Arc::DataPointRLS.

4.10.4.76 virtual bool Arc::DataPoint::WriteOutOfOrder () `[pure virtual]`

Returns true if URL can accept scattered data for ∗writing∗ operation.

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

4.10.5 Field Documentation

4.10.5.1 std::list<std::string> Arc::DataPoint::valid_url_options `[protected]`

Subclasses should add their own specific options to this list

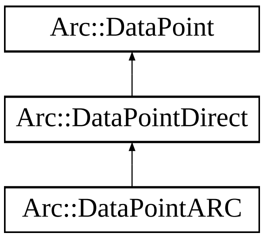
The documentation for this class was generated from the following file:

- DataPoint.h

4.11 Arc::DataPointARC Class Reference

```
#include <DataPointARC.h>
```

Inheritance diagram for Arc::DataPointARC::



Public Member Functions

- virtual DataStatus Check ()
- virtual DataStatus Remove ()
- virtual DataStatus Stat (FileInfo &file, DataPoint::DataPointInfoType verb)
- virtual DataStatus List (std::list< FileInfo > &file, DataPoint::DataPointInfoType verb)
- virtual DataStatus StartReading (DataBuffer &buffer)
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback ∗space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()

4.11.1 Detailed Description

Provides an interface to the Chelonia storage system developed by ARC.

4.11.2 Member Function Documentation

4.11.2.1 virtual DataStatus Arc::DataPointARC::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

4.11.2.2 virtual DataStatus Arc::DataPointARC::List (std::list< FileInfo > & *file*, DataPoint::DataPointInfoType *verb*) `[virtual]`

List hierarchical content of this object.

If the DataPoint represents a directory or something similar its contents will be listed.

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.

Implements Arc::DataPoint.

4.11.2.3 virtual DataStatus Arc::DataPointARC::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.11.2.4 virtual DataStatus Arc::DataPointARC::StartReading (DataBuffer & *buffer*) `[virtual]`

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before Stop-Reading() was called and returned.

Implements Arc::DataPoint.

4.11.2.5 virtual DataStatus Arc::DataPointARC::StartWriting (DataBuffer & *buffer*, DataCallback ∗ *space_cb* = `NULL`) `[virtual]`

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implements Arc::DataPoint.

4.11.2.6 virtual DataStatus Arc::DataPointARC::Stat (FileInfo & *file*, DataPoint::DataPointInfoType *verb*) `[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.

Implements Arc::DataPoint.

4.11.2.7 virtual DataStatus Arc::DataPointARC::StopReading () `[virtual]`

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.11.2.8 virtual DataStatus Arc::DataPointARC::StopWriting () `[virtual]`

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

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

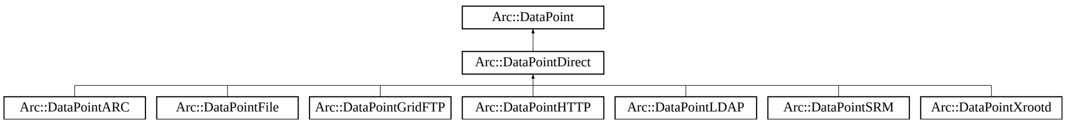
- DataPointARC.h

4.12 Arc::DataPointDirect Class Reference

This is a kind of generalized file handle.

`#include <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 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 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 RemoveLocations (const DataPoint &p)
- virtual DataStatus ClearLocations ()
- virtual void SortLocations (const std::string &, const URLMap &)

4.12.1 Detailed Description

This is a kind of generalized file handle.

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

4.12.2 Member Function Documentation

4.12.2.1 virtual bool Arc::DataPointDirect::AcceptsMeta () const `[virtual]`

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

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

Add a checksum object which will compute checksum during transmission.

Parameters:

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

Returns:

integer position in the list of checksum objects.

Implements Arc::DataPoint.

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

Add URL to list.

Parameters:

url Location URL to add.

meta Location meta information.

Implements Arc::DataPoint.

4.12.2.4 virtual int Arc::DataPointDirect::BufNum () const `[virtual]`

Get suggested number of buffers for transfers.

Implements Arc::DataPoint.

4.12.2.5 virtual long long int Arc::DataPointDirect::BufSize () const `[virtual]`

Get suggested buffer size for transfers.

Implements Arc::DataPoint.

4.12.2.6 virtual DataStatus Arc::DataPointDirect::ClearLocations () `[virtual]`

Remove all locations.

Implements Arc::DataPoint.

4.12.2.7 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.

4.12.2.8 virtual const URL& Arc::DataPointDirect::CurrentLocation () const `[virtual]`

Returns current (resolved) URL.

Implements Arc::DataPoint.

4.12.2.9 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.

4.12.2.10 virtual bool Arc::DataPointDirect::GetAdditionalChecks () const `[virtual]`

Check if additional checks before transfer will be performed.

Implements Arc::DataPoint.

4.12.2.11 virtual const CheckSum* Arc::DataPointDirect::GetCheckSumObject (int *index*) const `[virtual]`

Get CheckSum object at given position in list.

Implements Arc::DataPoint.

4.12.2.12 virtual bool Arc::DataPointDirect::GetSecure () const `[virtual]`

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

4.12.2.13 virtual bool Arc::DataPointDirect::HaveLocations () const `[virtual]`

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

4.12.2.14 virtual bool Arc::DataPointDirect::IsIndex () const `[virtual]`

Check if URL is an Indexing Service.

Implements Arc::DataPoint.

4.12.2.15 virtual bool Arc::DataPointDirect::IsStageable () const `[virtual]`

If URL should be staged or queried for Transport URL (TURL).

Reimplemented from Arc::DataPoint.

Reimplemented in Arc::DataPointSRM.

4.12.2.16 virtual bool Arc::DataPointDirect::LastLocation () `[virtual]`

Returns true if the current location is the last.

Implements Arc::DataPoint.

4.12.2.17 virtual bool Arc::DataPointDirect::Local () const `[virtual]`

Returns true if file is local, e.g. file:// urls.

Implements Arc::DataPoint.

4.12.2.18 virtual bool Arc::DataPointDirect::LocationValid () const `[virtual]`

Returns false if out of retries.

Implements Arc::DataPoint.

4.12.2.19 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.

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements Arc::DataPoint.

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

Index Service postregistration.

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

Parameters:

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

Implements Arc::DataPoint.

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

Index service preregistration.

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 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.

Implements Arc::DataPoint.

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

Index Service preunregistration.

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 same name.

Implements Arc::DataPoint.

4.12.2.24 virtual bool Arc::DataPointDirect::ProvidesMeta () const `[virtual]`

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

Reimplemented in Arc::DataPointGridFTP, and Arc::DataPointSRM.

4.12.2.25 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.

Implements Arc::DataPoint.

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

Parameters:

v true if allowed (default is false).

Implements Arc::DataPoint.

4.12.2.27 virtual bool Arc::DataPointDirect::Registered () const `[virtual]`

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

Implements Arc::DataPoint.

4.12.2.28 virtual DataStatus Arc::DataPointDirect::RemoveLocation () `[virtual]`

Remove current URL from list.

Implements Arc::DataPoint.

4.12.2.29 virtual DataStatus Arc::DataPointDirect::RemoveLocations (const DataPoint & *p*) `[virtual]`

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

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

Resolves index service URL into list of ordinary URLs.

Also obtains meta information about the file.

Parameters:

source true if DataPoint object represents source of information.

Implements Arc::DataPoint.

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

Allow/disallow additional checks.

Check for existence of remote file (and probably other checks too) before initiating reading and writing operations.

Parameters:

v true if allowed (default is true).

Implements Arc::DataPoint.

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

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default depends on protocol).

Implements Arc::DataPoint.

4.12.2.33 virtual void Arc::DataPointDirect::SortLocations (const std::string &, const URLMap &) `[inline, virtual]`

Sort locations according to the specified pattern.

Parameters:

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

Implements Arc::DataPoint.

4.12.2.34 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 is unregistered.

Implements Arc::DataPoint.

4.12.2.35 virtual bool Arc::DataPointDirect::WriteOutOfOrder () `[virtual]`

Returns true if URL can accept scattered data for ∗writing∗ operation.

Implements Arc::DataPoint.

Reimplemented in Arc::DataPointFile, and Arc::DataPointGridFTP.

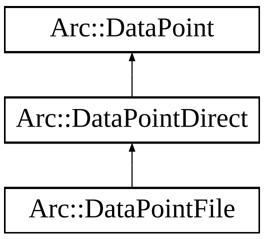
The documentation for this class was generated from the following file:

- DataPointDirect.h

4.13 Arc::DataPointFile Class Reference

```
#include <DataPointFile.h>
```

Inheritance diagram for Arc::DataPointFile::



Public Member Functions

- virtual DataStatus StartReading (DataBuffer &buffer)
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback ∗space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()
- virtual DataStatus Check ()
- virtual DataStatus Remove ()
- virtual bool WriteOutOfOrder ()

4.13.1 Detailed Description

This class allows access to the regular local filesystem through the same interface as is used for remote storage on the grid.

4.13.2 Member Function Documentation

4.13.2.1 virtual DataStatus Arc::DataPointFile::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

4.13.2.2 virtual DataStatus Arc::DataPointFile::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.13.2.3 virtual DataStatus Arc::DataPointFile::StartReading (DataBuffer & *buffer*) `[virtual]`

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implements Arc::DataPoint.

4.13.2.4 virtual DataStatus Arc::DataPointFile::StartWriting (DataBuffer & *buffer*, DataCallback ∗ *space_cb* = `NULL`) `[virtual]`

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implements Arc::DataPoint.

4.13.2.5 virtual DataStatus Arc::DataPointFile::StopReading () `[virtual]`

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.13.2.6 virtual DataStatus Arc::DataPointFile::StopWriting () `[virtual]`

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.13.2.7 virtual bool Arc::DataPointFile::WriteOutOfOrder () `[virtual]`

Returns true if URL can accept scattered data for ∗writing∗ operation.

Reimplemented from Arc::DataPointDirect.

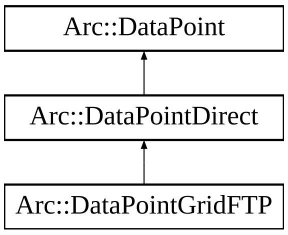
The documentation for this class was generated from the following file:

- DataPointFile.h

4.14 Arc::DataPointGridFTP Class Reference

`#include <DataPointGridFTP.h>`

Inheritance diagram for Arc::DataPointGridFTP::



Public Member Functions

- virtual bool SetURL (const URL &url)
- virtual DataStatus StartReading (DataBuffer &buf)
- virtual DataStatus StartWriting (DataBuffer &buf, DataCallback ∗space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()
- virtual DataStatus Check ()
- virtual DataStatus Remove ()
- virtual bool WriteOutOfOrder ()
- virtual bool ProvidesMeta () const
- virtual const std::string DefaultCheckSum () const

4.14.1 Detailed Description

GridFTP is essentially the FTP protocol with GSI security. This class uses libraries from the Globus Toolkit. It can also be used for regular FTP.

4.14.2 Member Function Documentation

4.14.2.1 virtual DataStatus Arc::DataPointGridFTP::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

4.14.2.2 virtual const std::string Arc::DataPointGridFTP::DefaultCheckSum () const `[virtual]`

Default checksum type.

Reimplemented from Arc::DataPoint.

4.14.2.3 virtual bool Arc::DataPointGridFTP::ProvidesMeta () const `[virtual]`

If endpoint can provide at least some meta information directly.

Reimplemented from Arc::DataPointDirect.

4.14.2.4 virtual DataStatus Arc::DataPointGridFTP::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.14.2.5 virtual bool Arc::DataPointGridFTP::SetURL (const URL & *url*) `[virtual]`

Assigns new URL. Main purpose of this method is to reuse existing connection for accessing different object at same server. Implementation does not have to implement this method. If supplied URL is not suitable or method is not implemented false is returned.

Reimplemented from Arc::DataPoint.

4.14.2.6 virtual DataStatus Arc::DataPointGridFTP::StartReading (DataBuffer & *buf*) `[virtual]`

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before Stop-Reading() was called and returned.

Implements Arc::DataPoint.

4.14.2.7 virtual DataStatus Arc::DataPointGridFTP::StartWriting (DataBuffer & *buf*, DataCallback ∗ *space_cb* = `NULL`) `[virtual]`

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implements Arc::DataPoint.

4.14.2.8 virtual DataStatus Arc::DataPointGridFTP::StopReading () [virtual]

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.14.2.9 virtual DataStatus Arc::DataPointGridFTP::StopWriting () [virtual]

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.14.2.10 virtual bool Arc::DataPointGridFTP::WriteOutOfOrder () [virtual]

Returns true if URL can accept scattered data for ∗writing∗ operation.

Reimplemented from Arc::DataPointDirect.

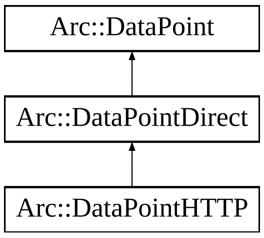
The documentation for this class was generated from the following file:

- DataPointGridFTP.h

4.15 Arc::DataPointHTTP Class Reference

`#include <DataPointHTTP.h>`

Inheritance diagram for Arc::DataPointHTTP::



Public Member Functions

- virtual bool SetURL (const URL &url)
- virtual DataStatus Check ()
- virtual DataStatus Remove ()
- virtual DataStatus StartReading (DataBuffer &buffer)
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback ∗space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()

4.15.1 Detailed Description

This class allows access through HTTP to remote resources. HTTP over SSL (HTTPS) and HTTP over GSI (HTTPG) are also supported.

4.15.2 Member Function Documentation

4.15.2.1 virtual DataStatus Arc::DataPointHTTP::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

4.15.2.2 virtual DataStatus Arc::DataPointHTTP::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.15.2.3 virtual bool Arc::DataPointHTTP::SetURL (const URL & *url*) `[virtual]`

Assigns new URL. Main purpose of this method is to reuse existing connection for accessing different object at same server. Implementation does not have to implement this method. If supplied URL is not suitable or method is not implemented false is returned.

Reimplemented from Arc::DataPoint.

4.15.2.4 virtual DataStatus Arc::DataPointHTTP::StartReading (DataBuffer & *buffer*) [virtual]

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implements Arc::DataPoint.

4.15.2.5 virtual DataStatus Arc::DataPointHTTP::StartWriting (DataBuffer & *buffer*, DataCallback ∗ *space_cb* = NULL) [virtual]

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implements Arc::DataPoint.

4.15.2.6 virtual DataStatus Arc::DataPointHTTP::StopReading () [virtual]

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.15.2.7 virtual DataStatus Arc::DataPointHTTP::StopWriting () [virtual]

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

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

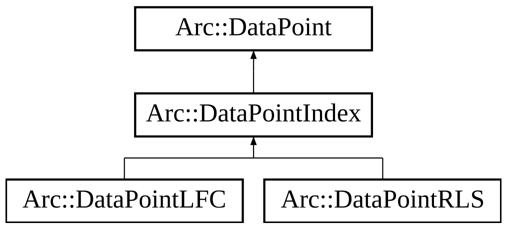
- DataPointHTTP.h

4.16 Arc::DataPointIndex Class Reference

Complements DataPoint with attributes common for Indexing Service URLs.

```
#include <DataPointIndex.h>
```

Inheritance diagram for Arc::DataPointIndex::



Public Member Functions

- virtual const URL & CurrentLocation () 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 RemoveLocation ()
- virtual DataStatus RemoveLocations (const DataPoint &p)
- 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 SetMeta (const DataPoint &p)
- 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 DataStatus PrepareReading (unsigned int timeout, unsigned int &wait_time, const std::list< std::string > &transport_protocols)
- virtual DataStatus PrepareWriting (unsigned int timeout, unsigned int &wait_time, const std::list< std::string > &transport_protocols)
- 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 ()
- 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

4.16.1 Detailed Description

Complements DataPoint with attributes common for Indexing Service URLs.

It should never be used directly. Instead inherit from it to provide a class for specific a Indexing Service.

4.16.2 Member Function Documentation

4.16.2.1 virtual bool Arc::DataPointIndex::AcceptsMeta () const `[virtual]`

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

4.16.2.2 virtual int Arc::DataPointIndex::AddCheckSumObject (CheckSum ∗ *cksum*) `[virtual]`

Add a checksum object which will compute checksum during transmission.

Parameters:

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

Returns:

integer position in the list of checksum objects.

Implements Arc::DataPoint.

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

Add URL to list.

Parameters:

url Location URL to add.

meta Location meta information.

Implements Arc::DataPoint.

4.16.2.4 virtual int Arc::DataPointIndex::BufNum () const [virtual]

Get suggested number of buffers for transfers.

Implements Arc::DataPoint.

4.16.2.5 virtual long long int Arc::DataPointIndex::BufSize () const [virtual]

Get suggested buffer size for transfers.

Implements Arc::DataPoint.

4.16.2.6 virtual DataStatus Arc::DataPointIndex::Check () [virtual]

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

Reimplemented in Arc::DataPointLFC, and Arc::DataPointRLS.

4.16.2.7 virtual DataStatus Arc::DataPointIndex::ClearLocations () [virtual]

Remove all locations.

Implements Arc::DataPoint.

4.16.2.8 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.

4.16.2.9 virtual const URL& Arc::DataPointIndex::CurrentLocation () const [virtual]

Returns current (resolved) URL.

Implements Arc::DataPoint.

4.16.2.10 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.

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

Finish reading from the URL.

Must be called after transfer of physical file has completed and 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.

Reimplemented from Arc::DataPoint.

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

Finish writing to the URL.

Must be called after transfer of physical file has completed and 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.

Reimplemented from Arc::DataPoint.

4.16.2.13 virtual DataPointAccessLatency Arc::DataPointIndex::GetAccessLatency () const `[virtual]`

Get value of meta-information ’access latency’.

Reimplemented from Arc::DataPoint.

4.16.2.14 virtual bool Arc::DataPointIndex::GetAdditionalChecks () const `[virtual]`

Check if additional checks before transfer will be performed.

Implements Arc::DataPoint.

4.16.2.15 virtual const CheckSum* Arc::DataPointIndex::GetCheckSumObject (int *index*) const `[virtual]`

Get CheckSum object at given position in list.

Implements Arc::DataPoint.

4.16.2.16 virtual bool Arc::DataPointIndex::GetSecure () const `[virtual]`

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

4.16.2.17 virtual bool Arc::DataPointIndex::HaveLocations () const `[virtual]`

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

4.16.2.18 virtual bool Arc::DataPointIndex::IsIndex () const `[virtual]`

Check if URL is an Indexing Service.

Implements Arc::DataPoint.

4.16.2.19 virtual bool Arc::DataPointIndex::IsStageable () const `[virtual]`

If URL should be staged or queried for Transport URL (TURL).

Reimplemented from Arc::DataPoint.

4.16.2.20 virtual bool Arc::DataPointIndex::LastLocation () `[virtual]`

Returns true if the current location is the last.

Implements Arc::DataPoint.

4.16.2.21 virtual bool Arc::DataPointIndex::Local () const `[virtual]`

Returns true if file is local, e.g. file:// urls.

Implements Arc::DataPoint.

4.16.2.22 virtual bool Arc::DataPointIndex::LocationValid () const `[virtual]`

Returns false if out of retries.

Implements Arc::DataPoint.

4.16.2.23 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.

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

Request passive transfers for FTP-like protocols.

Parameters:

true to request.

Implements Arc::DataPoint.

4.16.2.25 virtual DataStatus Arc::DataPointIndex::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented from Arc::DataPoint.

4.16.2.26 virtual DataStatus Arc::DataPointIndex::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented from Arc::DataPoint.

4.16.2.27 virtual bool Arc::DataPointIndex::ProvidesMeta () const `[virtual]`

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

4.16.2.28 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.

Implements Arc::DataPoint.

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

Parameters:

v true if allowed (default is false).

Implements Arc::DataPoint.

4.16.2.30 virtual bool Arc::DataPointIndex::Registered () const `[virtual]`

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

Implements Arc::DataPoint.

4.16.2.31 virtual DataStatus Arc::DataPointIndex::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.16.2.32 virtual DataStatus Arc::DataPointIndex::RemoveLocation () `[virtual]`

Remove current URL from list.

Implements Arc::DataPoint.

4.16.2.33 virtual DataStatus Arc::DataPointIndex::RemoveLocations (const DataPoint & *p*) `[virtual]`

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

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

Allow/disallow additional checks.

Check for existence of remote file (and probably other checks too) before initiating reading and writing operations.

Parameters:

v true if allowed (default is true).

Implements Arc::DataPoint.

4.16.2.35 virtual void Arc::DataPointIndex::SetCheckSum (const std::string & *val*) `[virtual]`

Set value of meta-information 'checksum'.

Reimplemented from Arc::DataPoint.

4.16.2.36 virtual void Arc::DataPointIndex::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.

Reimplemented from Arc::DataPoint.

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

Allow/disallow heavy security during data transfer.

Parameters:

v true if allowed (default depends on protocol).

Implements Arc::DataPoint.

4.16.2.38 virtual void Arc::DataPointIndex::SetSize (const unsigned long long int *val*) `[virtual]`

Set value of meta-information 'size'.

Reimplemented from Arc::DataPoint.

4.16.2.39 virtual void Arc::DataPointIndex::SetTries (const int *n*) `[virtual]`

Set number of retries.

Reimplemented from Arc::DataPoint.

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

Sort locations according to the specified pattern.

Parameters:

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

Implements Arc::DataPoint.

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

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implements Arc::DataPoint.

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

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implements Arc::DataPoint.

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

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.16.2.44 virtual DataStatus Arc::DataPointIndex::StopWriting () `[virtual]`

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.16.2.45 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.

4.16.2.46 virtual bool Arc::DataPointIndex::WriteOutOfOrder () `[virtual]`

Returns true if URL can accept scattered data for ∗writing∗ operation.

Implements Arc::DataPoint.

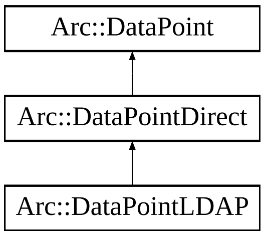
The documentation for this class was generated from the following file:

- DataPointIndex.h

4.17 Arc::DataPointLDAP Class Reference

`#include <DataPointLDAP.h>`

Inheritance diagram for Arc::DataPointLDAP::



Public Member Functions

- virtual DataStatus StartReading (DataBuffer &buffer)
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback ∗space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()
- virtual DataStatus Check ()
- virtual DataStatus Remove ()
- virtual DataStatus Stat (FileInfo &file, DataPoint::DataPointInfoType verb)
- virtual DataStatus List (std::list< FileInfo > &file, DataPoint::DataPointInfoType verb)

4.17.1 Detailed Description

LDAP is used in grids mainly to store information about grid services or resources rather than to store data itself. This class allows access to LDAP data through the same interface as other grid resources.

4.17.2 Member Function Documentation

4.17.2.1 virtual DataStatus Arc::DataPointLDAP::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

4.17.2.2 virtual DataStatus Arc::DataPointLDAP::List (std::list< FileInfo > & *file*, DataPoint::DataPointInfoType *verb*) `[virtual]`

List hierarchical content of this object.

If the DataPoint represents a directory or something similar its contents will be listed.

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.

Implements Arc::DataPoint.

4.17.2.3 virtual DataStatus Arc::DataPointLDAP::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.17.2.4 virtual DataStatus Arc::DataPointLDAP::StartReading (DataBuffer & *buffer*) `[virtual]`

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before Stop-Reading() was called and returned.

Implements Arc::DataPoint.

4.17.2.5 virtual DataStatus Arc::DataPointLDAP::StartWriting (DataBuffer & *buffer*, DataCallback ∗ *space_cb* = `NULL`) `[virtual]`

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implements Arc::DataPoint.

4.17.2.6 virtual DataStatus Arc::DataPointLDAP::Stat (FileInfo & *file*, DataPoint::DataPointInfoType *verb*) `[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.

Implements Arc::DataPoint.

4.17.2.7 virtual DataStatus Arc::DataPointLDAP::StopReading () `[virtual]`

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.17.2.8 virtual DataStatus Arc::DataPointLDAP::StopWriting () `[virtual]`

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

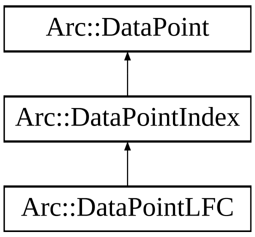
The documentation for this class was generated from the following file:

- DataPointLDAP.h

4.18 Arc::DataPointLFC Class Reference

```
#include <DataPointLFC.h>
```

Inheritance diagram for Arc::DataPointLFC::



Public Member Functions

- virtual DataStatus Resolve (bool source)
- virtual DataStatus Check ()
- virtual DataStatus PreRegister (bool replication, bool force=false)
- virtual DataStatus PostRegister (bool replication)
- virtual DataStatus PreUnregister (bool replication)
- virtual DataStatus Unregister (bool all)
- virtual const std::string DefaultCheckSum () const
- virtual std::string str () const

4.18.1 Detailed Description

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.

4.18.2 Member Function Documentation

4.18.2.1 virtual DataStatus Arc::DataPointLFC::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Reimplemented from Arc::DataPointIndex.

4.18.2.2 virtual const std::string Arc::DataPointLFC::DefaultCheckSum () const `[virtual]`

Default checksum type.

Reimplemented from Arc::DataPoint.

4.18.2.3 virtual DataStatus Arc::DataPointLFC::PostRegister (bool *replication*) `[virtual]`

Index Service postregistration.

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

Parameters:

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

Implements Arc::DataPoint.

4.18.2.4 virtual DataStatus Arc::DataPointLFC::PreRegister (bool *replication*, bool *force* = `false`) `[virtual]`

Index service preregistration.

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 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.

Implements Arc::DataPoint.

4.18.2.5 virtual DataStatus Arc::DataPointLFC::PreUnregister (bool *replication*) `[virtual]`

Index Service preunregistration.

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 same name.

Implements Arc::DataPoint.

4.18.2.6 virtual DataStatus Arc::DataPointLFC::Resolve (bool *source*) `[virtual]`

Resolves index service URL into list of ordinary URLs.

Also obtains meta information about the file.

Parameters:

source true if DataPoint object represents source of information.

Implements Arc::DataPoint.

4.18.2.7 virtual std::string Arc::DataPointLFC::str () const `[virtual]`

Returns a string representation of the DataPoint.

Reimplemented from Arc::DataPoint.

4.18.2.8 virtual DataStatus Arc::DataPointLFC::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 is unregistered.

Implements Arc::DataPoint.

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

- DataPointLFC.h

4.19 Arc::DataPointLoader Class Reference

Class used by DataHandle to load the required DMC.

```
#include <DataPoint.h>
```

4.19.1 Detailed Description

Class used by DataHandle to load the required DMC.

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

- DataPoint.h

4.20 Arc::DataPointPluginArgument Class Reference

Class representing the arguments passed to DMC plugins.

```
#include <DataPoint.h>
```

4.20.1 Detailed Description

Class representing the arguments passed to DMC plugins.

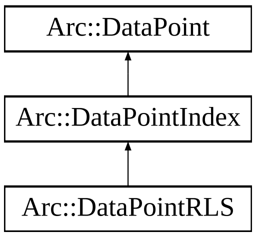
The documentation for this class was generated from the following file:

- DataPoint.h

4.21 Arc::DataPointRLS Class Reference

```
#include <DataPointRLS.h>
```

Inheritance diagram for Arc::DataPointRLS::



Public Member Functions

- virtual DataStatus Resolve (bool source)
- virtual DataStatus Check ()
- virtual DataStatus PreRegister (bool replication, bool force=false)
- virtual DataStatus PostRegister (bool replication)
- virtual DataStatus PreUnregister (bool replication)
- virtual DataStatus Unregister (bool all)

4.21.1 Detailed Description

The Replica Location Service (RLS) is a replica catalog developed by Globus. It maps filenames in a flat namespace to one or more physical locations, and can also store meta-information on each file. This class uses the Globus Toolkit libraries for accessing RLS.

4.21.2 Member Function Documentation

4.21.2.1 virtual DataStatus Arc::DataPointRLS::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Reimplemented from Arc::DataPointIndex.

4.21.2.2 virtual DataStatus Arc::DataPointRLS::PostRegister (bool *replication*) `[virtual]`

Index Service postregistration.

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

Parameters:

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

Implements Arc::DataPoint.

4.21.2.3 virtual DataStatus Arc::DataPointRLS::PreRegister (bool *replication*, bool *force* = `false`) `[virtual]`

Index service preregistration.

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 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.

Implements Arc::DataPoint.

4.21.2.4 virtual DataStatus Arc::DataPointRLS::PreUnregister (bool *replication*) `[virtual]`

Index Service preunregistration.

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 same name.

Implements Arc::DataPoint.

4.21.2.5 virtual DataStatus Arc::DataPointRLS::Resolve (bool *source*) `[virtual]`

Resolves index service URL into list of ordinary URLs.

Also obtains meta information about the file.

Parameters:

source true if DataPoint object represents source of information.

Implements Arc::DataPoint.

4.21.2.6 virtual DataStatus Arc::DataPointRLS::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 is unregistered.

Implements Arc::DataPoint.

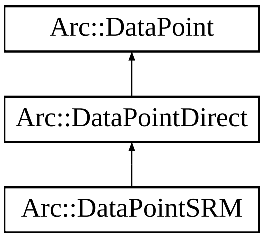
The documentation for this class was generated from the following file:

- DataPointRLS.h

4.22 Arc::DataPointSRM Class Reference

```
#include <DataPointSRM.h>
```

Inheritance diagram for Arc::DataPointSRM::



Public Member Functions

- virtual DataStatus PrepareReading (unsigned int timeout, unsigned int &wait_time, const std::list< std::string > &transport_protocols)
- virtual DataStatus PrepareWriting (unsigned int timeout, unsigned int &wait_time, const std::list< std::string > &transport_protocols)
- virtual DataStatus StartReading (DataBuffer &buffer)
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback ∗space_cb=NULL)
- virtual DataStatus StopWriting ()
- virtual DataStatus StopReading ()
- virtual DataStatus FinishReading (bool error)
- virtual DataStatus FinishWriting (bool error)
- virtual DataStatus Check ()
- virtual DataStatus Remove ()
- virtual const std::string DefaultCheckSum () const
- virtual bool ProvidesMeta () const
- virtual bool IsStageable () const
- virtual std::vector< URL > TransferLocations () const

4.22.1 Detailed Description

The Storage Resource Manager (SRM) protocol allows access to data distributed across physical storage through a unified namespace and management interface. PrepareReading() or PrepareWriting() must be used before reading or writing a physical file.

4.22.2 Member Function Documentation

4.22.2.1 virtual DataStatus Arc::DataPointSRM::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

4.22.2.2 virtual const std::string Arc::DataPointSRM::DefaultCheckSum () const `[virtual]`

Default checksum type.

Reimplemented from Arc::DataPoint.

4.22.2.3 virtual DataStatus Arc::DataPointSRM::FinishReading (bool *error*) `[virtual]`

Finish reading from the URL.

Must be called after transfer of physical file has completed and 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.

Reimplemented from Arc::DataPoint.

4.22.2.4 virtual DataStatus Arc::DataPointSRM::FinishWriting (bool *error*) `[virtual]`

Finish writing to the URL.

Must be called after transfer of physical file has completed and 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.

Reimplemented from Arc::DataPoint.

4.22.2.5 virtual bool Arc::DataPointSRM::IsStageable () const `[virtual]`

If URL should be staged or queried for Transport URL (TURL).

Reimplemented from Arc::DataPointDirect.

4.22.2.6 virtual DataStatus Arc::DataPointSRM::PrepareReading (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented from Arc::DataPoint.

4.22.2.7 virtual DataStatus Arc::DataPointSRM::PrepareWriting (unsigned int *timeout*, unsigned int & *wait_time*, const std::list< std::string > & *transport_protocols*) `[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.

transport_protocols A list of possible transport protocols for the physical file in order of preference.

Reimplemented from Arc::DataPoint.

4.22.2.8 virtual bool Arc::DataPointSRM::ProvidesMeta () const `[virtual]`

If endpoint can provide at least some meta information directly.

Reimplemented from Arc::DataPointDirect.

4.22.2.9 virtual DataStatus Arc::DataPointSRM::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.22.2.10 virtual DataStatus Arc::DataPointSRM::StartReading (DataBuffer & *buffer*) `[virtual]`

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implements Arc::DataPoint.

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

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

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

Implements Arc::DataPoint.

4.22.2.12 virtual DataStatus Arc::DataPointSRM::StopReading () [virtual]

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.22.2.13 virtual DataStatus Arc::DataPointSRM::StopWriting () [virtual]

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.22.2.14 virtual std::vector<URL> Arc::DataPointSRM::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.

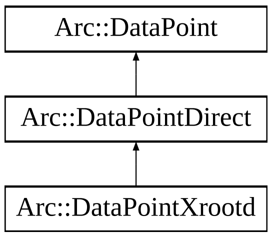
The documentation for this class was generated from the following file:

- DataPointSRM.h

4.23 Arc::DataPointXrootd Class Reference

`#include <DataPointXrootd.h>`

Inheritance diagram for Arc::DataPointXrootd::



Public Member Functions

- virtual DataStatus StartReading (DataBuffer &buffer)
- virtual DataStatus StartWriting (DataBuffer &buffer, DataCallback ∗space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()
- virtual DataStatus Check ()
- virtual DataStatus Remove ()

4.23.1 Detailed Description

xrootd is a protocol for data access across large scale storage clusters. More information can be found at http://xrootd.slac.stanford.edu/

4.23.2 Member Function Documentation

4.23.2.1 virtual DataStatus Arc::DataPointXrootd::Check () `[virtual]`

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements Arc::DataPoint.

4.23.2.2 virtual DataStatus Arc::DataPointXrootd::Remove () `[virtual]`

Remove/delete object at URL.

Implements Arc::DataPoint.

4.23.2.3 virtual DataStatus Arc::DataPointXrootd::StartReading (DataBuffer & *buffer*) `[virtual]`

Start reading data from URL.

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

Parameters:

buffer operation will use this buffer to put information into. Should not be destroyed before StopReading() was called and returned.

Implements Arc::DataPoint.

4.23.2.4 virtual DataStatus Arc::DataPointXrootd::StartWriting (DataBuffer & *buffer*, DataCallback ∗ *space_cb* = NULL) [virtual]

Start writing data to URL.

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

Parameters:

buffer operation will use this buffer to get information from. Should not be destroyed before stop_writing was called and returned.

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

Implements Arc::DataPoint.

4.23.2.5 virtual DataStatus Arc::DataPointXrootd::StopReading () [virtual]

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

4.23.2.6 virtual DataStatus Arc::DataPointXrootd::StopWriting () [virtual]

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

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

- DataPointXrootd.h

4.24 Arc::DataSpeed Class Reference

Keeps track of average and instantaneous transfer speed.

```
#include <DataSpeed.h>
```

Public Member Functions

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

4.24.1 Detailed Description

Keeps track of average and instantaneous transfer speed.

Also detects data transfer inactivity and other transfer timeouts.

4.24.2 Constructor & Destructor Documentation

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

Constructor

Parameters:

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

4.24.2.2 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:

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

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

min_speed_time

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

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

4.24.2.3 Arc::DataSpeed::∼DataSpeed (void)

Destructor.

4.24.3 Member Function Documentation

4.24.3.1 time_t Arc::DataSpeed::get_max_inactivity_time () `[inline]`

Get inactivity timeout.

4.24.3.2 void Arc::DataSpeed::hold (bool *disable*)

Turn off speed control.

Parameters:

disable true to turn off.

4.24.3.3 bool Arc::DataSpeed::max_inactivity_time_failure () `[inline]`

Check if maximal inactivity time error was triggered.

4.24.3.4 bool Arc::DataSpeed::min_average_speed_failure () `[inline]`

Check if minimal average speed error was triggered.

4.24.3.5 bool Arc::DataSpeed::min_speed_failure () `[inline]`

Check if minimal speed error was triggered.

4.24.3.6 void Arc::DataSpeed::reset (void)

Reset all counters and triggers.

4.24.3.7 void Arc::DataSpeed::set_base (time_t *base_* = `DATASPEED_AVERAGING_PERIOD`)

Set averaging time period.

Parameters:

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

4.24.3.8 void Arc::DataSpeed::set_max_data (unsigned long long int *max* = `0`)

Set amount of data to be transferred. Used in verbose messages.

Parameters:

max amount of data in bytes.

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

Set inactivity tiemout.

Parameters:

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

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

Set minmal avaerage speed.

Parameters:

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

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

Set minimal allowed speed.

Parameters:

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

min_speed_time

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

Specify which external function will print verbose messages. If not specified internal one is used.

Parameters:

pointer to function which prints information.

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

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

Parameters:

n amount of data transferred (bytes).

4.24.3.14 unsigned long long int Arc::DataSpeed::transferred_size (void) `[inline]`

Returns amount of data this object knows about.

4.24.3.15 bool Arc::DataSpeed::verbose (void)

Check if speed information is going to be printed.

4.24.3.16 void Arc::DataSpeed::verbose (const std::string & *prefix*)

Print information about current speed and amout of data.

Parameters:

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

4.24.3.17 void Arc::DataSpeed::verbose (bool *val*)

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

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

- DataSpeed.h

4.25 Arc::DataStatus Class Reference

Status code returned by many DataPoint methods.

```
#include <DataStatus.h>
```

Public Types

- Success = 0
- ReadAcquireError = 1
- WriteAcquireError = 2
- ReadResolveError = 3
- WriteResolveError = 4
- ReadStartError = 5
- WriteStartError = 6
- ReadError = 7
- WriteError = 8
- TransferError = 9
- ReadStopError = 10
- WriteStopError = 11
- PreRegisterError = 12
- PostRegisterError = 13
- UnregisterError = 14
- CacheError = 15
- CredentialsExpiredError = 16
- DeleteError = 17
- NoLocationError = 18
- LocationAlreadyExistsError = 19
- NotSupportedForDirectDataPointsError = 20
- UnimplementedError = 21
- IsReadingError = 22
- IsWritingError = 23
- CheckError = 24
- ListError = 25
- StatError = 27
- NotInitializedError = 29
- SystemError = 30
- StageError = 31
- InconsistentMetadataError = 32
- ReadPrepareError = 32
- ReadPrepareWait = 33
- WritePrepareError = 34
- WritePrepareWait = 35
- ReadFinishError = 36
- WriteFinishError = 37
- SuccessCached = 38
- UnknownError = 39

- enum DataStatusType {
  Success = 0, ReadAcquireError = 1 , WriteAcquireError = 2 , ReadResolveError = 3 ,
  WriteResolveError = 4 , ReadStartError = 5 , WriteStartError = 6 , ReadError = 7 ,
  WriteError = 8 , TransferError = 9 , ReadStopError = 10 , WriteStopError = 11 ,
  PreRegisterError = 12 , PostRegisterError = 13 , UnregisterError = 14 , CacheError = 15 ,
  CredentialsExpiredError = 16, DeleteError = 17 , NoLocationError = 18, LocationAlreadyExists-Error = 19,
  NotSupportedForDirectDataPointsError = 20, UnimplementedError = 21, IsReadingError = 22, Is-WritingError = 23,
  CheckError = 24 , ListError = 25 , StatError = 27 , NotInitializedError = 29,
  SystemError = 30, StageError = 31 , InconsistentMetadataError = 32, ReadPrepareError = 32 ,
  ReadPrepareWait = 33, WritePrepareError = 34 , WritePrepareWait = 35, ReadFinishError = 36 ,
  WriteFinishError = 37 , SuccessCached = 38, UnknownError = 39 }

Public Member Functions

- bool Passed () const
- bool Retryable () const
- void SetDesc (const std::string &d)
- std::string GetDesc () const

4.25.1 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.

4.25.2 Member Enumeration Documentation

4.25.2.1 enum Arc::DataStatus::DataStatusType

Status codes.

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 File listing failed.

StatError File/dir stating failed.

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.

SuccessCached Data was already cached.

UnknownError Undefined.

4.25.3 Member Function Documentation

4.25.3.1 std::string Arc::DataStatus::GetDesc () const `[inline]`

Get a text description of the status.

4.25.3.2 bool Arc::DataStatus::Passed () const `[inline]`

Returns true if no error occurred.

4.25.3.3 bool Arc::DataStatus::Retryable () const `[inline]`

Returns true if the error was temporary and could be retried.

4.25.3.4 void Arc::DataStatus::SetDesc (const std::string & *d*) `[inline]`

Set a text description of the status.

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

- DataStatus.h

4.26 Arc::FileCache Class Reference

FileCache provides an interface to all cache operations.

`#include <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, int cache_max=100, int cache_min=100)
- FileCache ()
- bool Start (const std::string &url, bool &available, bool &is_locked, bool use_remote=true)
- 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 Copy (const std::string &dest_path, const std::string &url, bool executable=false)
- 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)

4.26.1 Detailed Description

FileCache provides an interface to all cache operations.

An instance of FileCache should be created per job, and all files within the job are managed by that instance. 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. When a transfer has finished successfully, Link() should be called 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. Stop() must then be called to release any locks on the cache file. After the job has finished, Release() should be called to remove the hard links.

The cache directory(ies) and the optional directory to link to when the soft-links are made are set in the global configuration file. The names of cache files are formed from a hash of the URL specified as input to the job. 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 A-REX Administration Guide (NORDUGRID-TECH-14).

A metadata file with the '.meta' suffix is stored next to each cache file. This contains the URL corresponding to the cache file and the expiry time, if it is available.

While cache files are downloaded, they are locked using the FileLock class, which creates a lock file with the '.lock' suffix next to the cache file. Calling Start() creates this lock and Stop() releases it. All processes calling Start() must wait until they successfully obtain the lock before downloading can begin or an existing cache file can be used. Once a process obtains a lock it must later release it by calling Stop() or StopAndDelete(). Once a cache file is successfully linked to the per-job directory in Link(), it is also unlocked, but Stop() should still be called after.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 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.

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

4.26.2.2 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

4.26.2.3 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*, int *cache_max* = `100`, int *cache_min* = `100`)

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

cache_max maximum used space by cache, as percentage of the file system

cache_min minimum used space by cache, as percentage of the file system

4.26.2.4 Arc::FileCache::FileCache () `[inline]`

Default constructor. Invalid cache.

4.26.3 Member Function Documentation

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

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

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

Check if there is an information about creation time. Returns true if the file exists in the cache, since the creation time is the creation time of the cache file.

Parameters:

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

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

Check if the given DN is cached for authorisation.

Parameters:

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

DN the DN of the user

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

Check if there is an information about expiry time.

Parameters:

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

4.26.3.5 bool Arc::FileCache::Copy (const std::string & *dest_path*, const std::string & *url*, bool *executable* = `false`)

Copy the cache file corresponding to url to the dest_path. The session directory is accessed under the uid passed in the constructor, and switching uid involves holding a global lock. Therefore care must be taken in a multi-threaded environment.

This method is deprecated - Link() should be used instead with copy set to true.

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

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

Parameters:

url the URL to looks for in the cache

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

Get the creation time of a cached file. If the cache file does not exist, 0 is returned.

Parameters:

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

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

Get expiry time of a cached file. If the time is not available, a time equivalent to 0 is returned.

Parameters:

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

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

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 "." then files will be copied directly to the session directory rather than via the hard link.

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

4.26.3.10 Arc::FileCache::operator bool (void) `[inline]`

Returns true if object is useable.

4.26.3.11 bool Arc::FileCache::operator== (const FileCache & *a*)

Return true if all attributes are equal

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

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.

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

Set expiry time.

Parameters:

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

val expiry time

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

Prepare cache for downloading file, and lock the cached file. On success returns true. If there is another process downloading the same url, false is returned and is_locked is set to true. In this case the client should wait and retry later. If the lock has expired this process will take over the lock and the method will return as if no lock was present, ie available and is_locked are false.

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.

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

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.

Parameters:

url the url of the file that was downloaded

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

Release the cache file and delete it, because for example a failed download left an incomplete copy, or it has expired. 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().

Parameters:

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

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

- FileCache.h

4.27 Arc::FileCacheHash Class Reference

FileCacheHash provides methods to make hashes from strings.

`#include <FileCacheHash.h>`

Static Public Member Functions

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

4.27.1 Detailed Description

FileCacheHash provides methods to make hashes from strings.

Currently the SHA-1 hash from the openssl library is used.

4.27.2 Member Function Documentation

4.27.2.1 static std::string Arc::FileCacheHash::getHash (std::string *url*) `[static]`

Return a hash of the given URL, according to the current hash scheme.

4.27.2.2 static int Arc::FileCacheHash::maxLength () `[inline, static]`

Return the maximum length of a hash string.

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

- FileCacheHash.h

4.28 Arc::FileInfo Class Reference

FileInfo stores information about files (metadata).

```
#include <FileInfo.h>
```

4.28.1 Detailed Description

FileInfo stores information about files (metadata).

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

- FileInfo.h

4.29 Arc::LDAPQuery Class Reference

`#include <LDAPQuery.h>`

Public Member Functions

- LDAPQuery (const std::string &ldaphost, int ldapport, int timeout, bool anonymous=true, const std::string &usersn="")
- ∼LDAPQuery ()
- bool Query (const std::string &base, const std::string &filter="(objectclass=∗)", const std::list< std::string > &attributes=std::list< std::string >(), URL::Scope scope=URL::subtree)
- bool Result (ldap_callback callback, void ∗ref)

4.29.1 Detailed Description

LDAPQuery class; querying of LDAP servers.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 Arc::LDAPQuery::LDAPQuery (const std::string & *ldaphost*, int *ldapport*, int *timeout*, bool *anonymous* = `true`, const std::string & *usersn* = `""`)

Constructs a new LDAPQuery object and sets connection options. The connection is first established when calling Query.

4.29.2.2 Arc::LDAPQuery::∼LDAPQuery ()

Destructor. Will disconnect from the ldapserver if still connected.

4.29.3 Member Function Documentation

4.29.3.1 bool Arc::LDAPQuery::Query (const std::string & *base*, const std::string & *filter* = `"(objectclass=∗)"`, const std::list< std::string > & *attributes* = `std::list< std::string >()`, URL::Scope *scope* = `URL::subtree`)

Queries the ldap server.

4.29.3.2 bool Arc::LDAPQuery::Result (ldap_callback *callback*, void ∗ *ref*)

Retrieves the result of the query from the ldap-server.

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

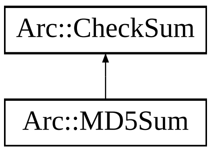
- LDAPQuery.h

4.30 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

```
#include <CheckSum.h>
```

Inheritance diagram for Arc::MD5Sum::



4.30.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

4.31 Arc::SRMClient Class Reference

```
#include <SRMClient.h>
```

Public Member Functions

- virtual ~SRMClient ()
- std::string getVersion () const
- virtual SRMReturnCode ping (std::string &version, bool report_error=true)=0
- virtual SRMReturnCode getSpaceTokens (std::list< std::string > &tokens, const std::string &description="")=0
- virtual SRMReturnCode getRequestTokens (std::list< std::string > &tokens, const std::string &description="")=0
- virtual SRMReturnCode getTURLs (SRMClientRequest &req, std::list< std::string > &urls)=0
- virtual SRMReturnCode getTURLsStatus (SRMClientRequest &req, std::list< std::string > &urls)=0
- virtual SRMReturnCode requestBringOnline (SRMClientRequest &req)=0
- virtual SRMReturnCode requestBringOnlineStatus (SRMClientRequest &req)=0
- virtual SRMReturnCode putTURLs (SRMClientRequest &req, std::list< std::string > &urls)=0
- virtual SRMReturnCode putTURLsStatus (SRMClientRequest &req, std::list< std::string > &urls)=0
- virtual SRMReturnCode releaseGet (SRMClientRequest &req)=0
- virtual SRMReturnCode releasePut (SRMClientRequest &req)=0
- virtual SRMReturnCode release (SRMClientRequest &req)=0
- virtual SRMReturnCode abort (SRMClientRequest &req)=0
- virtual SRMReturnCode info (SRMClientRequest &req, std::list< struct SRMFileMetaData > &metadata, const int recursive=0, bool report_error=true)=0
- virtual SRMReturnCode remove (SRMClientRequest &req)=0
- virtual SRMReturnCode copy (SRMClientRequest &req, const std::string &source)=0
- virtual SRMReturnCode mkDir (SRMClientRequest &req)=0
- virtual SRMReturnCode checkPermissions (SRMClientRequest &req)=0

Static Public Member Functions

- static SRMClient ∗ getInstance (const UserConfig &usercfg, const std::string &url, bool &timedout)

Protected Member Functions

- SRMClient (const UserConfig &usercfg, const SRMURL &url)
- SRMReturnCode process (PayloadSOAP ∗request, PayloadSOAP ∗∗response)

Protected Attributes

- std::string service_endpoint
- MCCConfig cfg
- ClientSOAP ∗ client
- NS ns
- SRMImplementation implementation
- time_t user_timeout
- std::string version

Static Protected Attributes

- static Logger logger

4.31.1 Detailed Description

A client interface to the SRM protocol. Instances of SRM clients are created by calling the getInstance() factory method. One client instance can be used to make many requests to the same server (with the same protocol version), but not multiple servers.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 Arc::SRMClient::SRMClient (const UserConfig & *usercfg*, const SRMURL & *url*) `[protected]`

Constructor

4.31.2.2 virtual Arc::SRMClient::∼SRMClient () `[virtual]`

Destructor

4.31.3 Member Function Documentation

4.31.3.1 virtual SRMReturnCode Arc::SRMClient::abort (SRMClientRequest & *req*) `[pure virtual]`

Called in the case of failure during transfer or releasePut. Releases all TURLs involved in the transfer.

Parameters:

req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.2 virtual SRMReturnCode Arc::SRMClient::checkPermissions (SRMClientRequest & *req*) `[pure virtual]`

Check permissions for the SURL in the request using the current credentials. req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.3 virtual SRMReturnCode Arc::SRMClient::copy (SRMClientRequest & *req*, const std::string & *source*) `[pure virtual]`

Copy a file between two SRM storages.

Parameters:

req The request object

source The source SURL

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.4 static SRMClient∗ Arc::SRMClient::getInstance (const UserConfig & *usercfg*, const std::string & *url*, bool & *timedout*) `[static]`

Returns an SRMClient instance with the required protocol version. This must be used to create SRMClient instances. Specifying a version explicitly forces creation of a client with that version.

Parameters:

usercfg The user configuration.

url A SURL. A client connects to the service host derived from this SURL. All operations with a client instance must use SURLs with the same host as this one.

timedout Whether the connection timed out

conn_timeout Connection timeout to the SRM service

Returns:

A pointer to an instance of SRMClient is returned, or NULL if it was not possible to create one.

4.31.3.5 virtual SRMReturnCode Arc::SRMClient::getRequestTokens (std::list< std::string > & *tokens*, const std::string & *description* = "") `[pure virtual]`

Returns a list of request tokens for the user calling the method which are still active requests, or the tokens corresponding to the token description, if given.

Parameters:

tokens The list filled by the service

description The user request description, which can be specified when the request is created

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.6 virtual SRMReturnCode Arc::SRMClient::getSpaceTokens (std::list< std::string > & *tokens*, const std::string & *description* = "") `[pure virtual]`

Find the space tokens available to write to which correspond to the space token description, if given. The list of tokens is a list of numbers referring to the SRM internal definition of the spaces, not user-readable strings.

Parameters:

tokens The list filled by the service *description* The space token description

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.7 virtual SRMReturnCode Arc::SRMClient::getTURLs (SRMClientRequest & *req*, std::list< std::string > & *urls*) `[pure virtual]`

If the user wishes to copy a file from somewhere, getTURLs() is called to retrieve the transport URL(s) to copy the file from. It may be used synchronously or asynchronously, depending on the synchronous property of the request object. In the former case it will block until the TURLs are ready, in the latter case it will return after making the request and getTURLsStatus() must be used to poll the request status if it was not completed.

Parameters:

req The request object

urls A list of TURLs filled by the method

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.8 virtual SRMReturnCode Arc::SRMClient::getTURLsStatus (SRMClientRequest & *req*, std::list< std::string > & *urls*) `[pure virtual]`

In the case where getTURLs was called asynchronously and the request was not completed, this method should be called to poll the status of the request. getTURLs must be called before this method and the request object must have ongoing request status.

Parameters:

req The request object. Status must be ongoing.

urls A list of TURLs filled by the method if the request completed successfully

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.9 std::string Arc::SRMClient::getVersion () const `[inline]`

Returns the version of the SRM protocol used by this instance

4.31.3.10 virtual SRMReturnCode Arc::SRMClient::info (SRMClientRequest & *req*, std::list< struct SRMFileMetaData > & *metadata*, const int *recursive* = `0`, bool *report_error* = `true`) `[pure virtual]`

Returns information on a file or files (v2.2 and higher) stored in an SRM, such as file size, checksum and estimated access latency.

Parameters:

req The request object

metadata A list of structs filled with file information

recursive The level of recursion into sub directories

report_error Determines if errors should be reported

Returns:

SRMReturnCode specifying outcome of operation

See also:

SRMFileMetaData

4.31.3.11 virtual SRMReturnCode Arc::SRMClient::mkDir (SRMClientRequest & *req*) `[pure virtual]`

Make required directories for the SURL in the request

Parameters:

req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.12 virtual SRMReturnCode Arc::SRMClient::ping (std::string & *version*, bool *report_error* = `true`) `[pure virtual]`

Find out the version supported by the server this client is connected to. Since this method is used to determine which client version to instantiate, we may not want to report an error to the user, so setting report_error to false supresses the error message.

Parameters:

version The version returned by the server

report_error Whether an error should be reported

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.13 SRMReturnCode Arc::SRMClient::process (PayloadSOAP $*$ *request*, PayloadSOAP $**$ *response*) `[protected]`

Process SOAP request

4.31.3.14 virtual SRMReturnCode Arc::SRMClient::putTURLs (SRMClientRequest & *req*, std::list< std::string > & *urls*) `[pure virtual]`

If the user wishes to copy a file to somewhere, putTURLs() is called to retrieve the transport URL(s) to copy the file to. It may be used synchronously or asynchronously, depending on the synchronous property of the request object. In the former case it will block until the TURLs are ready, in the latter case it will return after making the request and putTURLsStatus() must be used to poll the request status if it was not completed.

Parameters:

req The request object

urls A list of TURLs filled by the method

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.15 virtual SRMReturnCode Arc::SRMClient::putTURLsStatus (SRMClientRequest & *req*, std::list< std::string > & *urls*) `[pure virtual]`

In the case where putTURLs was called asynchronously and the request was not completed, this method should be called to poll the status of the request. putTURLs must be called before this method and the request object must have ongoing request status.

Parameters:

req The request object. Status must be ongoing.

urls A list of TURLs filled by the method if the request completed successfully

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.16 virtual SRMReturnCode Arc::SRMClient::release (SRMClientRequest & *req*) `[pure virtual]`

Used in SRM v1 only. Called to release files after successful transfer.

Parameters:

req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.17 virtual SRMReturnCode Arc::SRMClient::releaseGet (SRMClientRequest & *req*) `[pure virtual]`

Should be called after a successful copy from SRM storage.

Parameters:

req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.18 virtual SRMReturnCode Arc::SRMClient::releasePut (SRMClientRequest & *req*) `[pure virtual]`

Should be called after a successful copy to SRM storage.

Parameters:

req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.19 virtual SRMReturnCode Arc::SRMClient::remove (SRMClientRequest & *req*) `[pure virtual]`

Delete a file physically from storage and the SRM namespace.

Parameters:

req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.20 virtual SRMReturnCode Arc::SRMClient::requestBringOnline (SRMClientRequest & *req*) `[pure virtual]`

Submit a request to bring online files. If the synchronous property of the request object is false, this operation is asynchronous and the status of the request can be checked by calling requestBringOnline-Status() with the request token in req which is assigned by this method. If the request is synchronous, this operation blocks until the file(s) are online or the timeout specified in the SRMClient constructor has passed.

Parameters:

req The request object

Returns:

SRMReturnCode specifying outcome of operation

4.31.3.21 virtual SRMReturnCode Arc::SRMClient::requestBringOnlineStatus (SRMClientRequest & *req*) `[pure virtual]`

Query the status of a request to bring files online. The SURLs map of the request object is updated if the status of any files in the request has changed. requestBringOnline() but be called before this method.

Parameters:

req The request object to query the status of

Returns:

SRMReturnCode specifying outcome of operation

4.31.4 Field Documentation

4.31.4.1 MCCConfig Arc::SRMClient::cfg `[protected]`

SOAP configuraton object

4.31.4.2 ClientSOAP* Arc::SRMClient::client `[protected]`

SOAP client object

4.31.4.3 SRMImplementation Arc::SRMClient::implementation `[protected]`

The implementation of the server

4.31.4.4 Logger Arc::SRMClient::logger `[static, protected]`

Logger

4.31.4.5 NS Arc::SRMClient::ns `[protected]`

SOAP namespace

4.31.4.6 std::string Arc::SRMClient::service_endpoint `[protected]`

The URL of the service endpoint, eg httpg://srm.ndgf.org:8443/srm/managerv2 All SURLs passed to methods must correspond to this endpoint.

4.31.4.7 time_t Arc::SRMClient::user_timeout `[protected]`

Timeout for requests to the SRM service

4.31.4.8 std::string Arc::SRMClient::version `[protected]`

The version of the SRM protocol used

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

- SRMClient.h

4.32 Arc::SRMClientRequest Class Reference

```
#include <SRMClient.h>
```

Public Member Functions

- SRMClientRequest (const std::list< std::string > &urls) throw (SRMInvalidRequestException)
- SRMClientRequest (const std::string &url="", const std::string &id="") throw (SRMInvalidRequest-Exception)
- void request_id (int id)
- void request_token (const std::string &token)
- void file_ids (const std::list< int > &ids)
- void space_token (const std::string &token)
- std::list< std::string > surls () const
- void surl_statuses (const std::string &surl, SRMFileLocality locality)
- void surl_failures (const std::string &surl, const std::string &reason)
- void waiting_time (int wait_time)
- void finished_success ()
- void request_timeout (unsigned int timeout)
- void total_size (unsigned long long size)
- void long_list (bool list)
- void transport_protocols (const std::list< std::string > &protocols)

4.32.1 Detailed Description

Class to represent a request which may be used for multiple operations, for example calling getTURLs() sets the request token in the request object (for a v2.2 client) and then same object is passed to releaseGet().

4.32.2 Constructor & Destructor Documentation

4.32.2.1 Arc::SRMClientRequest::SRMClientRequest (const std::list< std::string > & *urls*) throw (SRMInvalidRequestException) `[inline]`

Creates a request object with multiple SURLs. The URLs here are in the form srm://srm.ndgf.org/data/atlas/disk/user/user.mlassnig.dataset.1/file3

4.32.2.2 Arc::SRMClientRequest::SRMClientRequest (const std::string & *url* = "", const std::string & *id* = "") throw (SRMInvalidRequestException) `[inline]`

Creates a request object with a single SURL. The URL here are in the form srm://srm.ndgf.org/data/atlas/disk/user/user.mlassnig.dataset.1/file3

4.32.3 Member Function Documentation

4.32.3.1 void Arc::SRMClientRequest::file_ids (const std::list< int > & *ids*) `[inline]`

set and get file id list

4.32.3.2 void Arc::SRMClientRequest::finished_success () `[inline]`

set and get status of request

4.32.3.3 void Arc::SRMClientRequest::long_list (bool *list*) `[inline]`

set and get long list flag

4.32.3.4 void Arc::SRMClientRequest::request_id (int *id*) `[inline]`

set and get request id

4.32.3.5 void Arc::SRMClientRequest::request_timeout (unsigned int *timeout*) `[inline]`

set and get request timeout

4.32.3.6 void Arc::SRMClientRequest::request_token (const std::string & *token*) `[inline]`

set and get request token

4.32.3.7 void Arc::SRMClientRequest::space_token (const std::string & *token*) `[inline]`

set and get space token

4.32.3.8 void Arc::SRMClientRequest::surl_failures (const std::string & *surl*, const std::string & *reason*) `[inline]`

set and get surl failures

4.32.3.9 void Arc::SRMClientRequest::surl_statuses (const std::string & *surl*, SRMFileLocality *locality*) `[inline]`

set and get surl statuses

4.32.3.10 std::list<std::string> Arc::SRMClientRequest::surls () const `[inline]`

get SURLs

4.32.3.11 void Arc::SRMClientRequest::total_size (unsigned long long *size*) `[inline]`

set and get total size

4.32.3.12 void Arc::SRMClientRequest::transport_protocols (const std::list< std::string > & *protocols*) `[inline]`

set and get transport protocols

4.32.3.13 void Arc::SRMClientRequest::waiting_time (int *wait_time*) [inline]

set and get waiting time. A waiting time of zero means no estimate was given by the remote service.

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

- SRMClient.h

4.33 SRMFileInfo Class Reference

```
#include <SRMInfo.h>
```

4.33.1 Detailed Description

Info about a particular entry in the SRM info file

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

- SRMInfo.h

4.34 Arc::SRMFileMetaData Struct Reference

```
#include <SRMClient.h>
```

4.34.1 Detailed Description

File metadata

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

- SRMClient.h

4.35 SRMInfo Class Reference

```
#include <SRMInfo.h>
```

4.35.1 Detailed Description

Represents SRM info stored in file. A combination of host and SRM version make a unique entry.

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

- SRMInfo.h

Index