

ARC Data Library libarcdata

Generated by Doxygen 1.7.1

Fri Oct 19 2012 12:26:16

Contents

1	Summary of libarcdata	1
2	Deprecated List	3
3	Data Structure Index	5
3.1	Class Hierarchy	5
4	Data Structure Index	7
4.1	Data Structures	7
5	Data Structure Documentation	9
5.1	Arc::CacheParameters Struct Reference	9
5.1.1	Detailed Description	9
5.2	Arc::DataBuffer Class Reference	9
5.2.1	Detailed Description	10
5.2.2	Constructor & Destructor Documentation	10
5.2.2.1	DataBuffer	10
5.2.2.2	DataBuffer	11
5.2.3	Member Function Documentation	11
5.2.3.1	add	11
5.2.3.2	buffer_size	11
5.2.3.3	checksum_object	11
5.2.3.4	checksum_valid	11
5.2.3.5	eof_read	11
5.2.3.6	eof_read	12
5.2.3.7	eof_write	12
5.2.3.8	eof_write	12
5.2.3.9	error	12
5.2.3.10	error_read	12
5.2.3.11	error_write	12

5.2.3.12	for_read	12
5.2.3.13	for_read	13
5.2.3.14	for_write	13
5.2.3.15	for_write	13
5.2.3.16	is_notwritten	13
5.2.3.17	is_notwritten	13
5.2.3.18	is_read	13
5.2.3.19	is_read	14
5.2.3.20	is_written	14
5.2.3.21	is_written	14
5.2.3.22	set	14
5.2.3.23	wait_any	14
5.3	Arc::DataCallback Class Reference	15
5.3.1	Detailed Description	15
5.4	Arc::DataHandle Class Reference	15
5.4.1	Detailed Description	15
5.4.2	Member Function Documentation	17
5.4.2.1	GetPoint	17
5.5	Arc::DataMover Class Reference	17
5.5.1	Detailed Description	18
5.5.2	Member Function Documentation	18
5.5.2.1	checks	18
5.5.2.2	checks	18
5.5.2.3	force_to_meta	18
5.5.2.4	secure	18
5.5.2.5	set_default_max_inactivity_time	18
5.5.2.6	set_default_min_average_speed	19
5.5.2.7	set_default_min_speed	19
5.5.2.8	Transfer	19
5.5.2.9	Transfer	19
5.5.2.10	verbose	19
5.6	Arc::DataPoint Class Reference	20
5.6.1	Detailed Description	22
5.6.2	Member Enumeration Documentation	23
5.6.2.1	DataPointAccessLatency	23
5.6.2.2	DataPointInfoType	24

5.6.3	Constructor & Destructor Documentation	24
5.6.3.1	DataPoint	24
5.6.4	Member Function Documentation	24
5.6.4.1	AddChecksumObject	24
5.6.4.2	AddLocation	25
5.6.4.3	AddURLOptions	25
5.6.4.4	Check	25
5.6.4.5	CompareLocationMetadata	25
5.6.4.6	CompareMeta	25
5.6.4.7	CreateDirectory	25
5.6.4.8	CurrentLocationMetadata	26
5.6.4.9	FinishReading	26
5.6.4.10	FinishWriting	26
5.6.4.11	GetFailureReason	26
5.6.4.12	List	26
5.6.4.13	NextLocation	27
5.6.4.14	Passive	27
5.6.4.15	PostRegister	27
5.6.4.16	PrepareReading	27
5.6.4.17	PrepareWriting	28
5.6.4.18	PreRegister	28
5.6.4.19	PreUnregister	28
5.6.4.20	ProvidesMeta	29
5.6.4.21	Range	29
5.6.4.22	ReadOutOfOrder	29
5.6.4.23	Registered	29
5.6.4.24	Rename	29
5.6.4.25	Resolve	29
5.6.4.26	Resolve	30
5.6.4.27	SetAdditionalChecks	30
5.6.4.28	SetMeta	30
5.6.4.29	SetSecure	30
5.6.4.30	SetURL	31
5.6.4.31	SortLocations	31
5.6.4.32	StartReading	31
5.6.4.33	StartWriting	31

5.6.4.34	Stat	32
5.6.4.35	Stat	32
5.6.4.36	StopReading	32
5.6.4.37	StopWriting	32
5.6.4.38	Transfer3rdParty	33
5.6.4.39	Transfer3rdParty	33
5.6.4.40	TransferLocations	33
5.6.4.41	Unregister	34
5.6.4.42	WriteOutOfOrder	34
5.6.5	Field Documentation	34
5.6.5.1	valid_url_options	34
5.7	Arc::DataPointDirect Class Reference	34
5.7.1	Detailed Description	35
5.7.2	Member Function Documentation	35
5.7.2.1	AddChecksumObject	35
5.7.2.2	AddLocation	36
5.7.2.3	CompareLocationMetadata	36
5.7.2.4	CurrentLocationMetadata	36
5.7.2.5	NextLocation	36
5.7.2.6	Passive	36
5.7.2.7	PostRegister	37
5.7.2.8	PreRegister	37
5.7.2.9	PreUnregister	37
5.7.2.10	ProvidesMeta	37
5.7.2.11	Range	37
5.7.2.12	ReadOutOfOrder	38
5.7.2.13	Registered	38
5.7.2.14	Resolve	38
5.7.2.15	Resolve	38
5.7.2.16	SetAdditionalChecks	38
5.7.2.17	SetSecure	39
5.7.2.18	SortLocations	39
5.7.2.19	Stat	39
5.7.2.20	Unregister	40
5.7.2.21	WriteOutOfOrder	40
5.8	Arc::DataPointIndex Class Reference	40

5.8.1	Detailed Description	41
5.8.2	Member Function Documentation	41
5.8.2.1	AddChecksumObject	41
5.8.2.2	AddLocation	42
5.8.2.3	Check	42
5.8.2.4	CompareLocationMetadata	42
5.8.2.5	CurrentLocationMetadata	42
5.8.2.6	FinishReading	42
5.8.2.7	FinishWriting	43
5.8.2.8	NextLocation	43
5.8.2.9	Passive	43
5.8.2.10	PrepareReading	43
5.8.2.11	PrepareWriting	44
5.8.2.12	ProvidesMeta	44
5.8.2.13	Range	44
5.8.2.14	ReadOutOfOrder	44
5.8.2.15	Registered	44
5.8.2.16	SetAdditionalChecks	45
5.8.2.17	SetMeta	45
5.8.2.18	SetSecure	45
5.8.2.19	SortLocations	45
5.8.2.20	StartReading	45
5.8.2.21	StartWriting	46
5.8.2.22	StopReading	46
5.8.2.23	StopWriting	46
5.8.2.24	TransferLocations	46
5.8.2.25	WriteOutOfOrder	47
5.9	Arc::DataPointLoader Class Reference	47
5.9.1	Detailed Description	47
5.10	Arc::DataPointPluginArgument Class Reference	47
5.10.1	Detailed Description	47
5.11	Arc::DataSpeed Class Reference	47
5.11.1	Detailed Description	48
5.11.2	Constructor & Destructor Documentation	48
5.11.2.1	DataSpeed	48
5.11.2.2	DataSpeed	48

5.11.3	Member Function Documentation	49
5.11.3.1	hold	49
5.11.3.2	set_base	49
5.11.3.3	set_max_data	49
5.11.3.4	set_max_inactivity_time	49
5.11.3.5	set_min_average_speed	49
5.11.3.6	set_min_speed	50
5.11.3.7	set_progress_indicator	50
5.11.3.8	transfer	50
5.11.3.9	verbose	50
5.11.3.10	verbose	50
5.12	Arc::DataStatus Class Reference	50
5.12.1	Detailed Description	52
5.12.2	Member Enumeration Documentation	52
5.12.2.1	DataStatusType	52
5.12.3	Constructor & Destructor Documentation	55
5.12.3.1	DataStatus	55
5.12.4	Member Function Documentation	55
5.12.4.1	Retryable	55
5.13	Arc::FileCache Class Reference	55
5.13.1	Detailed Description	56
5.13.2	Constructor & Destructor Documentation	56
5.13.2.1	FileCache	56
5.13.2.2	FileCache	57
5.13.2.3	FileCache	57
5.13.3	Member Function Documentation	57
5.13.3.1	AddDN	57
5.13.3.2	CheckCreated	57
5.13.3.3	CheckDN	58
5.13.3.4	CheckValid	58
5.13.3.5	File	58
5.13.3.6	GetCreated	58
5.13.3.7	GetValid	58
5.13.3.8	Link	59
5.13.3.9	Release	59
5.13.3.10	SetValid	59

5.13.3.11 Start	59
5.13.3.12 Stop	60
5.13.3.13 StopAndDelete	60
5.14 Arc::FileCacheHash Class Reference	60
5.14.1 Detailed Description	61
5.15 Arc::FileInfo Class Reference	61
5.15.1 Detailed Description	61
5.16 Arc::URLMap Class Reference	61

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). The DataHandle class should be used to automatically load the required DMC at runtime. To create a new DMC for a protocol which is not yet supported see the instruction and examples in the DataPoint class documentation. This documentation also gives a complete overview of the interface.

The following protocols are currently supported in standard distributions of ARC (except XRootd, which is not yet distributed).

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

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

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

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

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

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

RLS (rls://) - 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.

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

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

Chapter 2

Deprecated List

Global [Arc::DataStatus::CacheErrorRetryable](#)

Global [Arc::DataStatus::CheckErrorRetryable](#)

Global [Arc::DataStatus::CreateDirectoryErrorRetryable](#)

Global [Arc::DataStatus::DeleteErrorRetryable](#)

Global [Arc::DataStatus::GenericErrorRetryable](#)

Global [Arc::DataStatus::ListErrorRetryable](#)

Global [Arc::DataStatus::ListNonDirError](#) ListError with errno set to ENOTDIR should be used instead

Global [Arc::DataStatus::PostRegisterErrorRetryable](#)

Global [Arc::DataStatus::PreRegisterErrorRetryable](#)

Global [Arc::DataStatus::ReadAcquireErrorRetryable](#)

Global [Arc::DataStatus::ReadErrorRetryable](#)

Global [Arc::DataStatus::ReadFinishErrorRetryable](#)

Global [Arc::DataStatus::ReadPrepareErrorRetryable](#)

Global [Arc::DataStatus::ReadResolveErrorRetryable](#)

Global [Arc::DataStatus::ReadStartErrorRetryable](#)

Global [Arc::DataStatus::ReadStopErrorRetryable](#)

Global [Arc::DataStatus::RenameErrorRetryable](#)

Global [Arc::DataStatus::StageErrorRetryable](#)

Global [Arc::DataStatus::StatErrorRetryable](#)

Global [Arc::DataStatus::StatNotPresentError](#) StatError with errno set to ENOENT should be used instead

Global [Arc::DataStatus::TransferErrorRetryable](#)

Global [Arc::DataStatus::UnregisterErrorRetryable](#)

Global [Arc::DataStatus::WriteAcquireErrorRetryable](#)

Global [Arc::DataStatus::WriteErrorRetryable](#)

Global [Arc::DataStatus::WriteFinishErrorRetryable](#)

Global [Arc::DataStatus::WritePrepareErrorRetryable](#)

Global [Arc::DataStatus::WriteResolveErrorRetryable](#)

Global [Arc::DataStatus::WriteStartErrorRetryable](#)

Global [Arc::DataStatus::WriteStopErrorRetryable](#)

Chapter 3

Data Structure Index

3.1 Class Hierarchy

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

Arc::CacheParameters	9
Arc::DataBuffer	9
Arc::DataCallback	15
Arc::DataHandle	15
Arc::DataMover	17
Arc::DataPoint	20
Arc::DataPointDirect	34
Arc::DataPointIndex	40
Arc::DataPointLoader	47
Arc::DataPointPluginArgument	47
Arc::DataSpeed	47
Arc::DataStatus	50
Arc::FileCache	55
Arc::FileCacheHash	60
Arc::FileInfo	61
Arc::URLMap	61

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Arc::CacheParameters (Contains data on the parameters of a cache)	9
Arc::DataBuffer (Represents set of buffers)	9
Arc::DataCallback (This class is used by DataHandle to report missing space on local filesystem)	15
Arc::DataHandle (This class is a wrapper around the DataPoint class)	15
Arc::DataMover (DataMover provides an interface to transfer data between two DataPoints) . .	17
Arc::DataPoint (A DataPoint represents a data resource and is an abstraction of a URL)	20
Arc::DataPointDirect (This is a kind of generalized file handle)	34
Arc::DataPointIndex (Complements DataPoint with attributes common for Indexing Service URLs)	40
Arc::DataPointLoader (Class used by DataHandle to load the required DMC)	47
Arc::DataPointPluginArgument (Class representing the arguments passed to DMC plugins) . .	47
Arc::DataSpeed (Keeps track of average and instantaneous transfer speed)	47
Arc::DataStatus (Status code returned by many DataPoint methods)	50
Arc::FileCache (FileCache provides an interface to all cache operations)	55
Arc::FileCacheHash (FileCacheHash provides methods to make hashes from strings)	60
Arc::FileInfo (FileInfo stores information about files (metadata))	61
Arc::URLMap	61

Chapter 5

Data Structure Documentation

5.1 Arc::CacheParameters Struct Reference

Contains data on the parameters of a cache.

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

5.1.1 Detailed Description

Contains data on the parameters of a cache.

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

- FileCache.h

5.2 Arc::DataBuffer Class Reference

Represents set of buffers.

```
#include <DataBuffer.h>
```

Data Structures

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

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

5.2.1 Detailed Description

Represents set of buffers. This class is used during data transfer using [DataPoint](#) classes.

5.2.2 Constructor & Destructor Documentation

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

Contructor

Parameters

size size of every buffer in bytes.

blocks number of buffers.

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

Constructor

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.

5.2.3 Member Function Documentation

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

Add a checksum object which will compute checksum of buffer.

Parameters

cksum object which will compute checksum. Should not be destroyed till [DataBuffer](#) itself.

Returns

integer position in the list of checksum objects.

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

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

5.2.3.3 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.

5.2.3.4 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.

5.2.3.5 bool Arc::DataBuffer::eof_read ()

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

5.2.3.6 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.

5.2.3.7 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.

5.2.3.8 bool Arc::DataBuffer::eof_write ()

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

5.2.3.9 bool Arc::DataBuffer::error ()

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

5.2.3.10 void Arc::DataBuffer::error_read (bool v)

Informs object if error occurred on 'read' side.

Parameters

v true if error.

5.2.3.11 void Arc::DataBuffer::error_write (bool v)

Informs object if error occurred on 'write' side.

Parameters

v true if error.

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

5.2.3.13 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.

5.2.3.14 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 available buffers, method will wait for one. For python bindings pattern of this method is (bool, handle, length, offset, buffer) for_write(wait). Here buffer is string with content of buffer provided by [DataBuffer](#) object;

5.2.3.15 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.

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

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

Parameters

handle buffer's number.

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

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

Parameters

buf - address of buffer

5.2.3.18 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.

5.2.3.19 **bool Arc::DataBuffer::is_read (int *handle*, unsigned int *length*, unsigned long long int *offset*)**

Inform object that data was read into buffer.

Parameters

handle buffer's number.

length amount of data.

offset offset in stream, file, etc. For python bindings pattern of that method is bool is_read(handle,buffer,offset). Here buffer is string containing content of buffer to be passed to [DataBuffer](#) object.

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

Inform object that data was written from buffer.

Parameters

handle buffer's number.

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

Inform object that data was written from buffer.

Parameters

buf - address of buffer

5.2.3.22 **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.

5.2.3.23 **bool Arc::DataBuffer::wait_any ()**

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

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

- [DataBuffer.h](#)

5.3 Arc::DataCallback Class Reference

This class is used by [DataHandle](#) to report missing space on local filesystem.

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

5.3.1 Detailed Description

This class is used by [DataHandle](#) to report missing space on local filesystem. One of 'cb' functions here will be called if operation initiated by `DataHandle::StartReading` runs out of disk space.

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

- [DataCallback.h](#)

5.4 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

Static Public Member Functions

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

5.4.1 Detailed Description

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

This class is main way to access remote data items and obtain information about them. Below is an example of accessing last 512 bytes of files stored at GridFTP server.

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

```

#include <arc/data/DataBuffer.h>

using namespace Arc;

int main(void) {
    #define DESIRED_SIZE 512
    Arc::UserConfig usercfg;
    URL url("gsiftp://localhost/files/file_test_21");
    DataPoint* handle = DataHandle::GetPoint(url,usercfg);
    if(!handle) {
        std::cerr<<"Unsupported URL protocol or malformed URL"<<std::endl;
        return -1;
    };
    FileInfo info;
    if(!handle->Stat(info)) {
        std::cerr<<"Failed Stat"<<std::endl;
        return -1;
    };
    unsigned long long int fsize = handle->GetSize();
    if(fsize == (unsigned long long int)-1) {
        std::cerr<<"file size is not available"<<std::endl;
        return -1;
    };
    if(fsize == 0) {
        std::cerr<<"file is empty"<<std::endl;
        return -1;
    };
    unsigned long long int foffset;
    if(fsize > DESIRED_SIZE) {
        handle->Range(fsize-DESIRED_SIZE,fsize-1);
    };
    unsigned int wto;
    DataBuffer buffer;
    if(!handle->PrepareReading(10,wto)) {
        std::cerr<<"Failed PrepareReading"<<std::endl;
        return -1;
    };
    if(!handle->StartReading(buffer)) {
        std::cerr<<"Failed StopReading"<<std::endl;
        return -1;
    };
    for(;;) {
        int n;
        unsigned int length;
        unsigned long long int offset;
        if(!buffer.for_write(n,length,offset,true)) {
            break;
        };
        std::cout<<"BUFFER: "<<offset<<": "<<length<<": "<<std::string((const char*)
            (buffer[n]),length)<<std::endl;
        buffer.is_written(n);
    };
    if(buffer.error()) {
        std::cerr<<"Transfer failed"<<std::endl;
    };
    handle->StopReading();
    handle->FinishReading();
    return 0;
}

```

And the same example in python

```

import arc

desired_size = 512
usercfg = arc.UserConfig()
url = arc.URL("gsiftp://localhost/files/file_test_21")

```

```

handle = arc.DataHandle.GetPoint(url,usercfg)
info = arc.FileInfo("")
handle.Stat(info)
print "Name: ", info.GetName()
fsize = info.GetSize()
if fsize > desired_size:
    handle.Range(fsize-desired_size,fsize-1)
buffer = arc.DataBuffer()
res, wto = handle.PrepareReading(10)
handle.StartReading(buffer)
while True:
    n = 0
    length = 0
    offset = 0
    ( r, n, length, offset, buf) = buffer.for_write(True)
    if not r: break
    print "BUFFER: ", offset, ": ", length, " :", buf
    buffer.is_written(n);

```

5.4.2 Member Function Documentation

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

Returns a pointer to new [DataPoint](#) object corresponding to URL. This static method is mostly for bindings to other languages and if availability scope of obtained [DataPoint](#) is undefined.

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

- `DataHandle.h`

5.5 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)
- void [Cancel](#) ()
- bool [verbose](#) ()
- void [verbose](#) (bool)
- void [verbose](#) (const std::string &prefix)
- bool [retry](#) ()
- void [retry](#) (bool)

- void [secure](#) (bool)
- void [passive](#) (bool)
- void [force_to_meta](#) (bool)
- bool [checks](#) ()
- void [checks](#) (bool v)
- void [set_default_min_speed](#) (unsigned long long int min_speed, time_t min_speed_time)
- void [set_default_min_average_speed](#) (unsigned long long int min_average_speed)
- void [set_default_max_inactivity_time](#) (time_t max_inactivity_time)
- void [set_progress_indicator](#) (DataSpeed::show_progress_t func=NULL)
- void [set_preferred_pattern](#) (const std::string &pattern)

5.5.1 Detailed Description

[DataMover](#) provides an interface to transfer data between two DataPoints. Its main action is represented by Transfer methods

5.5.2 Member Function Documentation

5.5.2.1 bool Arc::DataMover::checks ()

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

5.5.2.2 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).

5.5.2.3 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.

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

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

5.5.2.5 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.

5.5.2.6 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.

5.5.2.7 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.

5.5.2.8 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.

5.5.2.9 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.

5.5.2.10 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.

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

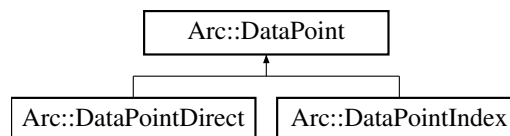
- DataMover.h

5.6 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

- enum [DataPointAccessLatency](#) { [ACCESS_LATENCY_ZERO](#), [ACCESS_LATENCY_SMALL](#), [ACCESS_LATENCY_LARGE](#) }
- enum [DataPointInfoType](#) {
[INFO_TYPE_MINIMAL](#) = 0, [INFO_TYPE_NAME](#) = 1, [INFO_TYPE_TYPE](#) = 2, [INFO_TYPE_TIMES](#) = 4,
[INFO_TYPE_CONTENT](#) = 8, [INFO_TYPE_ACCESS](#) = 16, [INFO_TYPE_STRUCT](#) = 32, [INFO_TYPE_REST](#) = 64,
[INFO_TYPE_ALL](#) = 127 }
- typedef void(* [Callback3rdParty](#))(unsigned long long int bytes_transferred)

Public Member Functions

- virtual [~DataPoint](#) ()
- virtual const URL & [GetURL](#) () const
- virtual const UserConfig & [GetUserConfig](#) () const
- virtual bool [SetURL](#) (const URL &url)
- virtual std::string [str](#) () const
- virtual operator bool () const
- virtual bool operator! () const
- virtual [DataStatus](#) [PrepareReading](#) (unsigned int timeout, unsigned int &wait_time)
- virtual [DataStatus](#) [PrepareWriting](#) (unsigned int timeout, unsigned int &wait_time)
- virtual [DataStatus](#) [StartReading](#) ([DataBuffer](#) &buffer)=0
- virtual [DataStatus](#) [StartWriting](#) ([DataBuffer](#) &buffer, [DataCallback](#) *space_cb=NULL)=0
- virtual [DataStatus](#) [StopReading](#) ()=0
- virtual [DataStatus](#) [StopWriting](#) ()=0

- virtual [DataStatus FinishReading](#) (bool error=false)
- virtual [DataStatus FinishWriting](#) (bool error=false)
- virtual [DataStatus Check](#) ()=0
- virtual [DataStatus Remove](#) ()=0
- virtual [DataStatus Stat](#) ([FileInfo](#) &file, [DataPointInfoType](#) verb=INFO_TYPE_ALL)=0
- virtual [DataStatus Stat](#) (std::list< [FileInfo](#) > &files, const std::list< [DataPoint](#) * > &urls, [DataPointInfoType](#) verb=INFO_TYPE_ALL)=0
- virtual [DataStatus List](#) (std::list< [FileInfo](#) > &files, [DataPointInfoType](#) verb=INFO_TYPE_ALL)=0
- virtual [DataStatus CreateDirectory](#) (bool with_parents=false)=0
- virtual [DataStatus Rename](#) (const URL &newurl)=0
- virtual void [ReadOutOfOrder](#) (bool v)=0
- virtual bool [WriteOutOfOrder](#) ()=0
- virtual void [SetAdditionalChecks](#) (bool v)=0
- virtual bool [GetAdditionalChecks](#) () const =0
- virtual void [SetSecure](#) (bool v)=0
- virtual bool [GetSecure](#) () const =0
- virtual void [Passive](#) (bool v)=0
- virtual [DataStatus GetFailureReason](#) (void) const
- virtual void [Range](#) (unsigned long long int start=0, unsigned long long int end=0)=0
- virtual [DataStatus Resolve](#) (bool source)=0
- virtual [DataStatus Resolve](#) (bool source, const std::list< [DataPoint](#) * > &urls)=0
- virtual bool [Registered](#) () const =0
- virtual [DataStatus PreRegister](#) (bool replication, bool force=false)=0
- virtual [DataStatus PostRegister](#) (bool replication)=0
- virtual [DataStatus PreUnregister](#) (bool replication)=0
- virtual [DataStatus Unregister](#) (bool all)=0
- virtual bool [CheckSize](#) () const
- virtual void [SetSize](#) (const unsigned long long int val)
- virtual unsigned long long int [GetSize](#) () const
- virtual bool [CheckChecksum](#) () const
- virtual void [SetChecksum](#) (const std::string &val)
- virtual const std::string & [GetChecksum](#) () const
- virtual const std::string [DefaultChecksum](#) () const
- virtual bool [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 [DataPoint](#) * [CurrentLocationHandle](#) () const =0
- virtual [DataStatus](#) [CompareLocationMetadata](#) () const =0
- virtual bool [NextLocation](#) ()=0
- virtual bool [LocationValid](#) () const =0
- virtual bool [LastLocation](#) ()=0
- virtual bool [HaveLocations](#) () const =0
- virtual [DataStatus](#) [AddLocation](#) (const URL &url, const std::string &meta)=0
- virtual [DataStatus](#) [RemoveLocation](#) ()=0
- virtual [DataStatus](#) [RemoveLocations](#) (const [DataPoint](#) &p)=0
- virtual [DataStatus](#) [ClearLocations](#) ()=0
- virtual int [AddChecksumObject](#) (Checksum *cksum)=0
- virtual const Checksum * [GetChecksumObject](#) (int index) const =0
- virtual void [SortLocations](#) (const std::string &pattern, const [URLMap](#) &url_map)=0
- virtual void [AddURLOptions](#) (const std::map< std::string, std::string > &options)

Static Public Member Functions

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

Protected Member Functions

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

Protected Attributes

- std::set< std::string > [valid_url_options](#)

5.6.1 Detailed Description

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

[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& userc
    fg) {
    ...
}

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 }
};
```

5.6.2 Member Enumeration Documentation

5.6.2.1 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.

5.6.2.2 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.

5.6.3 Constructor & Destructor Documentation**5.6.3.1 Arc::DataPoint::DataPoint (const URL & *url*, const UserConfig & *usercfg*, PluginArgument * *parg*) [protected]**

Constructor.

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

Parameters

url The URL representing the [DataPoint](#)

usercfg User configuration object

5.6.4 Member Function Documentation**5.6.4.1 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](#).

5.6.4.2 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](#).

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

Add URL options to this DataPoint's URL object. Invalid options for the [DataPoint](#) instance will not be added.

5.6.4.4 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](#).

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

Compare metadata of [DataPoint](#) and current location.

Returns inconsistency error or error encountered during operation, or success

Implemented in [Arc::DataPointDirect](#), and [Arc::DataPointIndex](#).

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

Compare meta information from another object.

Undefined values are not used for comparison.

Parameters

p object to which to compare.

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

Create a directory.

If the protocol supports it, this method creates the last directory in the path to the URL. It assumes the last component of the path is a file-like object and not a directory itself, unless the path ends in a directory separator. If *with_parents* is true then all missing parent directories in the path will also be created.

Parameters

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

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

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implemented in [Arc::DataPointDirect](#), and [Arc::DataPointIndex](#).

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

Finish reading from the URL.

Must be called after transfer of physical file has completed 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](#).

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

Finish writing to the URL.

Must be called after transfer of physical file has completed 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](#).

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

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

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

List hierarchical content of this object.

If the [DataPoint](#) represents a directory or something similar its contents will be listed.

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.

5.6.4.13 virtual bool Arc::DataPoint::NextLocation () [pure virtual]

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implemented in [Arc::DataPointDirect](#), and [Arc::DataPointIndex](#).

5.6.4.14 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](#).

5.6.4.15 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](#).

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

Prepare [DataPoint](#) for reading.

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

Parameters

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

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

Reimplemented in [Arc::DataPointIndex](#).

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

Prepare [DataPoint](#) for writing.

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

Parameters

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

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

Reimplemented in [Arc::DataPointIndex](#).

5.6.4.18 **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](#).

5.6.4.19 **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](#).

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

If endpoint can provide at least some meta information directly.

Implemented in [Arc::DataPointDirect](#), and [Arc::DataPointIndex](#).

5.6.4.21 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](#).

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

Allow/disallow [DataPoint](#) to produce scattered data during reading* operation.

Parameters

v true if allowed (default is false).

Implemented in [Arc::DataPointDirect](#), and [Arc::DataPointIndex](#).

5.6.4.23 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](#).

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

Rename a URL.

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

Parameters

newurl The new name for the URL

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

Resolves several index service URLs.

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

Parameters

- source* true if [DataPoint](#) objects represent source of information
- urls* List of DataPoints to resolve. Protocols and hosts must match and match this DataPoint's protocol and host.

Implemented in [Arc::DataPointDirect](#).

5.6.4.26 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](#).

5.6.4.27 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](#).

5.6.4.28 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](#).

5.6.4.29 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](#).

5.6.4.30 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.

5.6.4.31 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](#).

5.6.4.32 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. If [StopReading\(\)](#) is not called explicitly to release buffer it will be released in destructor of [DataPoint](#) which also usually calls [StopReading\(\)](#).

Implemented in [Arc::DataPointIndex](#).

5.6.4.33 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 [StopWriting\(\)](#) was called and returned. If [StopWriting\(\)](#) is not called explicitly to release buffer it will be released in destructor of [DataPoint](#) which also usually calls [StopWriting\(\)](#).

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

Implemented in [Arc::DataPointIndex](#).

5.6.4.34 **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.

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

Retrieve information about several DataPoints.

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

Parameters

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

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

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

Implemented in [Arc::DataPointDirect](#).

5.6.4.36 **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](#).

5.6.4.37 **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](#).

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

Perform third party transfer.

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

Parameters

- source* Source URL to pull data from
- destination* Destination URL which pulls data to itself
- callback* Optional monitoring callback

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

Perform third party transfer.

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

Parameters

- source* Source URL to pull data from
- destination* Destination URL which pulls data to itself
- usercfg* Configuration information
- callback* Optional monitoring callback

5.6.4.40 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](#).

5.6.4.41 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](#).

5.6.4.42 virtual bool Arc::DataPoint::WriteOutOfOrder () [pure virtual]

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

Implemented in [Arc::DataPointDirect](#), and [Arc::DataPointIndex](#).

5.6.5 Field Documentation

5.6.5.1 std::set<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:

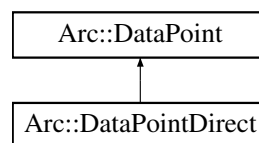
- [DataPoint.h](#)

5.7 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](#) [Stat](#) (std::list< [FileInfo](#) > &files, const std::list< [DataPoint](#) * > &urls, [DataPointInfoType](#) verb=INFO_TYPE_ALL)
- virtual [DataStatus](#) [Resolve](#) (bool source)
- virtual [DataStatus](#) [Resolve](#) (bool source, const std::list< [DataPoint](#) * > &urls)
- virtual bool [Registered](#) () const
- virtual [DataStatus](#) [PreRegister](#) (bool replication, bool force=false)
- virtual [DataStatus](#) [PostRegister](#) (bool replication)
- virtual [DataStatus](#) [PreUnregister](#) (bool replication)
- virtual [DataStatus](#) [Unregister](#) (bool all)
- virtual bool [AcceptsMeta](#) () const
- virtual bool [ProvidesMeta](#) () const
- virtual const URL & [CurrentLocation](#) () const
- virtual [DataPoint](#) * [CurrentLocationHandle](#) () const
- virtual const std::string & [CurrentLocationMetadata](#) () const
- virtual [DataStatus](#) [CompareLocationMetadata](#) () const
- virtual bool [NextLocation](#) ()
- virtual bool [LocationValid](#) () const
- virtual bool [HaveLocations](#) () const
- virtual bool [LastLocation](#) ()
- virtual [DataStatus](#) [AddLocation](#) (const URL &url, const std::string &meta)
- virtual [DataStatus](#) [RemoveLocation](#) ()
- virtual [DataStatus](#) [RemoveLocations](#) (const [DataPoint](#) &p)
- virtual [DataStatus](#) [ClearLocations](#) ()
- virtual void [SortLocations](#) (const std::string &, const [URLMap](#) &)

5.7.1 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.

5.7.2 Member Function Documentation

5.7.2.1 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](#).

5.7.2.2 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](#).

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

Compare metadata of [DataPoint](#) and current location.

Returns inconsistency error or error encountered during operation, or success

Implements [Arc::DataPoint](#).

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

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implements [Arc::DataPoint](#).

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

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implements [Arc::DataPoint](#).

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

Request passive transfers for FTP-like protocols.

Parameters

true to request.

Implements [Arc::DataPoint](#).

5.7.2.7 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](#).

5.7.2.8 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](#).

5.7.2.9 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](#).

5.7.2.10 virtual bool Arc::DataPointDirect::ProvidesMeta () const [virtual]

If endpoint can provide at least some meta information directly.

Implements [Arc::DataPoint](#).

5.7.2.11 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](#).

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

Allow/disallow [DataPoint](#) to produce scattered data during reading* operation.

Parameters

v true if allowed (default is false).

Implements [Arc::DataPoint](#).

5.7.2.13 **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](#).

5.7.2.14 **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](#).

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

Resolves several index service URLs.

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

Parameters

source true if [DataPoint](#) objects represent source of information

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

Implements [Arc::DataPoint](#).

5.7.2.16 **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](#).

5.7.2.17 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](#).

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

Sort locations according to the specified pattern.

Parameters

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

Implements [Arc::DataPoint](#).

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

Retrieve information about several DataPoints.

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

Parameters

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

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

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

Implements [Arc::DataPoint](#).

5.7.2.20 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](#).

5.7.2.21 virtual `bool Arc::DataPointDirect::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:

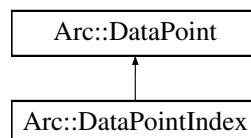
- `DataPointDirect.h`

5.8 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 [DataPoint](#) * [CurrentLocationHandle](#) () const
- virtual [DataStatus](#) [CompareLocationMetadata](#) () const
- virtual bool [NextLocation](#) ()
- virtual bool [LocationValid](#) () const
- virtual bool [HaveLocations](#) () const
- virtual bool [LastLocation](#) ()
- virtual [DataStatus](#) [RemoveLocation](#) ()
- virtual [DataStatus](#) [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)
- virtual [DataStatus](#) [PrepareWriting](#) (unsigned int timeout, unsigned int &wait_time)
- virtual [DataStatus](#) [StartReading](#) ([DataBuffer](#) &buffer)
- virtual [DataStatus](#) [StartWriting](#) ([DataBuffer](#) &buffer, [DataCallback](#) *space_cb=NULL)
- virtual [DataStatus](#) [StopReading](#) ()
- virtual [DataStatus](#) [StopWriting](#) ()
- virtual [DataStatus](#) [FinishReading](#) (bool error=false)
- virtual [DataStatus](#) [FinishWriting](#) (bool error=false)
- virtual std::vector< URL > [TransferLocations](#) () const
- virtual [DataStatus](#) [Check](#) ()
- 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

5.8.1 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.

5.8.2 Member Function Documentation

5.8.2.1 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](#).

5.8.2.2 **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](#).

5.8.2.3 **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](#).

5.8.2.4 **virtual DataStatus Arc::DataPointIndex::CompareLocationMetadata () const [virtual]**

Compare metadata of [DataPoint](#) and current location.

Returns inconsistency error or error encountered during operation, or success

Implements [Arc::DataPoint](#).

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

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implements [Arc::DataPoint](#).

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

Finish reading from the URL.

Must be called after transfer of physical file has completed 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](#).

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

Finish writing to the URL.

Must be called after transfer of physical file has completed 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](#).

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

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implements [Arc::DataPoint](#).

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

Request passive transfers for FTP-like protocols.

Parameters

true to request.

Implements [Arc::DataPoint](#).

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

Prepare [DataPoint](#) for reading.

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

Parameters

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

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

Reimplemented from [Arc::DataPoint](#).

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

Prepare [DataPoint](#) for writing.

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

Parameters

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

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

Reimplemented from [Arc::DataPoint](#).

5.8.2.12 **virtual bool Arc::DataPointIndex::ProvidesMeta () const [virtual]**

If endpoint can provide at least some meta information directly.

Implements [Arc::DataPoint](#).

5.8.2.13 **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](#).

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

Allow/disallow [DataPoint](#) to produce scattered data during reading* operation.

Parameters

v true if allowed (default is false).

Implements [Arc::DataPoint](#).

5.8.2.15 **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](#).

5.8.2.16 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](#).

5.8.2.17 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](#).

5.8.2.18 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](#).

5.8.2.19 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](#).

5.8.2.20 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. If [StopReading\(\)](#) is not called explicitly to release buffer it will be released in destructor of [DataPoint](#) which also usually calls [StopReading\(\)](#).

Implements [Arc::DataPoint](#).

5.8.2.21 **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 [StopWriting\(\)](#) was called and returned. If [StopWriting\(\)](#) is not called explicitly to release buffer it will be released in destructor of [DataPoint](#) which also usually calls [StopWriting\(\)](#).

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

Implements [Arc::DataPoint](#).

5.8.2.22 **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](#).

5.8.2.23 **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](#).

5.8.2.24 **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](#).

5.8.2.25 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

5.9 Arc::DataPointLoader Class Reference

Class used by [DataHandle](#) to load the required DMC.

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

5.9.1 Detailed Description

Class used by [DataHandle](#) to load the required DMC.

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

- DataPoint.h

5.10 Arc::DataPointPluginArgument Class Reference

Class representing the arguments passed to DMC plugins.

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

5.10.1 Detailed Description

Class representing the arguments passed to DMC plugins.

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

- DataPoint.h

5.11 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)

5.11.1 Detailed Description

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 `Arc::DataSpeed::DataSpeed (time_t base = DATASPEED_AVERAGING_PERIOD)`

Constructor

Parameters

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

5.11.2.2 `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.

5.11.3 Member Function Documentation

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

Turn off speed control.

Parameters

disable true to turn off.

5.11.3.2 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).

5.11.3.3 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.

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

Set inactivity timeout.

Parameters

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

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

Set minimal average speed.

Parameters

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

5.11.3.6 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

5.11.3.7 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.

5.11.3.8 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).

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

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

5.11.3.10 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.

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

- DataSpeed.h

5.12 Arc::DataStatus Class Reference

Status code returned by many [DataPoint](#) methods.

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

Public Types

- enum [DataStatusType](#) {
 - [Success](#), [ReadAcquireError](#), [WriteAcquireError](#), [ReadResolveError](#),
 - [WriteResolveError](#), [ReadStartError](#), [WriteStartError](#), [ReadError](#),
 - [WriteError](#), [TransferError](#), [ReadStopError](#), [WriteStopError](#),
 - [PreRegisterError](#), [PostRegisterError](#), [UnregisterError](#), [CacheError](#),
 - [CredentialsExpiredError](#), [DeleteError](#), [NoLocationError](#), [LocationAlreadyExistsError](#),
 - [NotSupportedForDirectDataPointsError](#), [UnimplementedError](#), [IsReadingError](#), [IsWritingError](#),
 - [CheckError](#), [ListError](#), [ListNonDirError](#), [StatError](#),
 - [StatNotPresentError](#), [NotInitializedError](#), [SystemError](#), [StageError](#),
 - [InconsistentMetadataError](#), [ReadPrepareError](#), [ReadPrepareWait](#), [WritePrepareError](#),
 - [WritePrepareWait](#), [ReadFinishError](#), [WriteFinishError](#), [CreateDirectoryError](#),
 - [RenameError](#), [SuccessCached](#), [SuccessCancelled](#), [GenericError](#),
 - [UnknownError](#), [ReadAcquireErrorRetryable](#) = [DataStatusRetryableBase](#)+[ReadAcquireError](#),
 - [WriteAcquireErrorRetryable](#) = [DataStatusRetryableBase](#)+[WriteAcquireError](#), [ReadResolveErrorRetryable](#) = [DataStatusRetryableBase](#)+[ReadResolveError](#),
 - [WriteResolveErrorRetryable](#) = [DataStatusRetryableBase](#)+[WriteResolveError](#), [ReadStartErrorRetryable](#) = [DataStatusRetryableBase](#)+[ReadStartError](#),
 - [WriteStartErrorRetryable](#) = [DataStatusRetryableBase](#)+[WriteStartError](#), [ReadErrorRetryable](#) = [DataStatusRetryableBase](#)+[ReadError](#),
 - [WriteErrorRetryable](#) = [DataStatusRetryableBase](#)+[WriteError](#), [TransferErrorRetryable](#) = [DataStatusRetryableBase](#)+[TransferError](#),
 - [ReadStopErrorRetryable](#) = [DataStatusRetryableBase](#)+[ReadStopError](#), [WriteStopErrorRetryable](#) = [DataStatusRetryableBase](#)+[WriteStopError](#),
 - [PreRegisterErrorRetryable](#) = [DataStatusRetryableBase](#)+[PreRegisterError](#), [PostRegisterErrorRetryable](#) = [DataStatusRetryableBase](#)+[PostRegisterError](#),
 - [UnregisterErrorRetryable](#) = [DataStatusRetryableBase](#)+[UnregisterError](#), [CacheErrorRetryable](#) = [DataStatusRetryableBase](#)+[CacheError](#),
 - [DeleteErrorRetryable](#) = [DataStatusRetryableBase](#)+[DeleteError](#), [CheckErrorRetryable](#) = [DataStatusRetryableBase](#)+[CheckError](#),
 - [ListErrorRetryable](#) = [DataStatusRetryableBase](#)+[ListError](#), [StatErrorRetryable](#) = [DataStatusRetryableBase](#)+[StatError](#),
 - [StageErrorRetryable](#) = [DataStatusRetryableBase](#)+[StageError](#), [ReadPrepareErrorRetryable](#) = [DataStatusRetryableBase](#)+[ReadPrepareError](#),
 - [WritePrepareErrorRetryable](#) = [DataStatusRetryableBase](#)+[WritePrepareError](#), [ReadFinishErrorRetryable](#) = [DataStatusRetryableBase](#)+[ReadFinishError](#),
 - [WriteFinishErrorRetryable](#) = [DataStatusRetryableBase](#)+[WriteFinishError](#), [CreateDirectoryErrorRetryable](#) = [DataStatusRetryableBase](#)+[CreateDirectoryError](#),
 - [RenameErrorRetryable](#) = [DataStatusRetryableBase](#)+[RenameError](#), [GenericErrorRetryable](#) = [DataStatusRetryableBase](#)+[GenericError](#) }

Public Member Functions

- [DataStatus](#) (const [DataStatusType](#) &status, std::string desc="")
- [DataStatus](#) (const [DataStatusType](#) &status, int error_no, const std::string &desc="")
- [DataStatus](#) ()
- bool [Passed](#) () const
- bool [Retryable](#) () const
- void [SetErrNo](#) (int error_no)
- int [GetErrno](#) () const
- std::string [GetStrErrno](#) () const
- void [SetDesc](#) (const std::string &d)

- `std::string GetDesc () const`
- `operator std::string (void) const`

5.12.1 Detailed Description

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

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

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

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

5.12.2 Member Enumeration Documentation

5.12.2.1 enum `Arc::DataStatus::DataStatusType`

Status codes.

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

Enumerator:

- Success*** Operation completed successfully.
- ReadAcquireError*** Source is bad URL or can't be used due to some reason.
- WriteAcquireError*** Destination is bad URL or can't be used due to some reason.
- ReadResolveError*** Resolving of index service URL for source failed.
- WriteResolveError*** Resolving of index service URL for destination failed.
- ReadStartError*** Can't read from source.
- WriteStartError*** Can't write to destination.
- ReadError*** Failed while reading from source.
- WriteError*** Failed while writing to destination.
- TransferError*** Failed while transferring data (mostly timeout).
- ReadStopError*** Failed while finishing reading from source.
- WriteStopError*** Failed while finishing writing to destination.
- PreRegisterError*** First stage of registration of index service URL failed.
- PostRegisterError*** Last stage of registration of index service URL failed.
- UnregisterError*** Unregistration of index service URL failed.
- CacheError*** Error in caching procedure.
- CredentialsExpiredError*** Error due to provided credentials are expired.

DeleteError Error deleting location or URL.

NoLocationError No valid location available.

LocationAlreadyExistsError No valid location available.

NotSupportedForDirectDataPointsError Operation has no sense for this kind of URL.

UnimplementedError Feature is unimplemented.

IsReadingError [DataPoint](#) is already reading.

IsWritingError [DataPoint](#) is already writing.

CheckError Access check failed.

ListError Directory listing failed.

Deprecated

ListNonDirError ListError with errno set to ENOTDIR should be used instead

StatError File/dir stating failed.

Deprecated

StatNotPresentError StatError with errno set to ENOENT should be used instead

NotInitializedError Object initialization failed.

SystemError Error in OS.

StageError Staging error.

InconsistentMetadataError Inconsistent metadata.

ReadPrepareError Can't prepare source.

ReadPrepareWait Wait for source to be prepared.

WritePrepareError Can't prepare destination.

WritePrepareWait Wait for destination to be prepared.

ReadFinishError Can't finish source.

WriteFinishError Can't finish destination.

CreateDirectoryError Can't create directory.

RenameError Can't rename URL.

SuccessCached Data was already cached.

SuccessCancelled Operation was cancelled successfully.

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

UnknownError Undefined.

Deprecated

ReadAcquireErrorRetryable

Deprecated

WriteAcquireErrorRetryable

Deprecated

ReadResolveErrorRetryable

Deprecated

WriteResolveErrorRetryable

Deprecated

ReadStartErrorRetryable

	Deprecated
<i>WriteStartErrorRetryable</i>	Deprecated
<i>ReadErrorRetryable</i>	Deprecated
<i>WriteErrorRetryable</i>	Deprecated
<i>TransferErrorRetryable</i>	Deprecated
<i>ReadStopErrorRetryable</i>	Deprecated
<i>WriteStopErrorRetryable</i>	Deprecated
<i>PreRegisterErrorRetryable</i>	Deprecated
<i>PostRegisterErrorRetryable</i>	Deprecated
<i>UnregisterErrorRetryable</i>	Deprecated
<i>CacheErrorRetryable</i>	Deprecated
<i>DeleteErrorRetryable</i>	Deprecated
<i>CheckErrorRetryable</i>	Deprecated
<i>ListErrorRetryable</i>	Deprecated
<i>StatErrorRetryable</i>	Deprecated
<i>StageErrorRetryable</i>	Deprecated
<i>ReadPrepareErrorRetryable</i>	Deprecated
<i>WritePrepareErrorRetryable</i>	Deprecated
<i>ReadFinishErrorRetryable</i>	Deprecated
<i>WriteFinishErrorRetryable</i>	

Deprecated*CreateDirectoryErrorRetryable***Deprecated***RenameErrorRetryable***Deprecated***GenericErrorRetryable***5.12.3 Constructor & Destructor Documentation**

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

Construct a new [DataStatus](#) with errno and optional text description.

If the status is an error condition then error_no must be set to a non-zero value

5.12.4 Member Function Documentation

5.12.4.1 `bool Arc::DataStatus::Retryable () const`

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

Retryable error numbers are EAGAIN, EBUSY, ETIMEDOUT, EARCSVCTMP, EARCTransfer-TIMEOUT and EARCCHECKSUM.

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

- DataStatus.h

5.13 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)
- [FileCache](#) ()
- bool [Start](#) (const std::string &url, bool &available, bool &is_locked, bool use_remote=true, bool delete_first=false)
- bool [Stop](#) (const std::string &url)
- bool [StopAndDelete](#) (const std::string &url)
- std::string [File](#) (const std::string &url)

- bool [Link](#) (const std::string &link_path, const std::string &url, bool copy, bool executable, bool holding_lock, bool &try_again)
- bool [Release](#) () const
- bool [AddDN](#) (const std::string &url, const std::string &DN, const Time &expiry_time)
- bool [CheckDN](#) (const std::string &url, const std::string &DN)
- bool [CheckCreated](#) (const std::string &url)
- Time [GetCreated](#) (const std::string &url)
- bool [CheckValid](#) (const std::string &url)
- Time [GetValid](#) (const std::string &url)
- bool [SetValid](#) (const std::string &url, const Time &val)
- operator bool ()
- bool operator== (const [FileCache](#) &a)

5.13.1 Detailed Description

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

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

The cache directory(ies) and the optional directory to link to when the soft-links are made are set in the constructor. The names of cache files are formed from an SHA-1 hash of the URL to cache. To ease the load on the file system, the cache files are split into subdirectories based on the first two characters in the hash. For example the file with hash 76f11edda169848038efbd9fa3df5693 is stored in 76/f11edda169848038efbd9fa3df5693. A cache filename can be found by passing the URL to [Find\(\)](#). For more information on the structure of the cache, see the ARC Computing Element System Administrator Guide (NORDUGRID-MANUAL-20).

5.13.2 Constructor & Destructor Documentation

5.13.2.1 `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

5.13.2.2 Arc::FileCache::FileCache (const std::vector< std::string > & *cache*s, const std::string & *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new [FileCache](#) instance with multiple cache dirs

Parameters

*cache*s 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

5.13.2.3 Arc::FileCache::FileCache (const std::vector< std::string > & *cache*s, const std::vector< std::string > & *remote_cache*s, const std::vector< std::string > & *draining_cache*s, const std::string & *id*, uid_t *job_uid*, gid_t *job_gid*)

Create a new [FileCache](#) instance with multiple cache dirs, remote caches and draining cache directories.

Parameters

*cache*s a vector of strings describing caches. The format of each string is "cache_dir[link_path]".
*remote_cache*s 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_cache*s 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

5.13.3 Member Function Documentation

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

Store a DN in the permissions cache for the given url.

Add the given DN to the list of cached DNs with the given expiry time.

Parameters

url the url corresponding to the cache file to which we want to add a cached DN
DN the DN of the user
expiry_time the expiry time of this DN in the DN cache

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

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

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

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

Check if a DN exists in the permission cache for the given url.

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

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

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

Parameters

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

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

Get the cache filename for the given URL.

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

Parameters

url the URL to look for in the cache

5.13.3.6 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

5.13.3.7 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

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

Link a cache file to the place it will be used.

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

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

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

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

Parameters

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

url url of file to link to or copy

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

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

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

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

5.13.3.9 `bool Arc::FileCache::Release () const`

Release cache files used in this cache.

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

5.13.3.10 `bool Arc::FileCache::SetValid (const std::string & url, const Time & val)`

Set expiry time of a cache file.

Parameters

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

val expiry time

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

Start preparing to cache the file specified by url.

[Start\(\)](#) returns true if the file was successfully prepared. The `available` parameter is set to true if the file already exists and in this case [Link\(\)](#) can be called immediately. If `available` is false the caller should write

the file and then call [Link\(\)](#) followed by [Stop\(\)](#). It returns false if it was unable to prepare the cache file for any reason. In this case the `is_locked` parameter should be checked and if it is true the file is locked by another process and the caller should try again later.

Parameters

url url that is being downloaded

available true on exit if the file is already in cache

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

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

delete_first If true then any existing cache file is deleted.

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

Stop the cache after a file was downloaded.

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

Parameters

url the url of the file that was downloaded

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

Stop the cache after a file was downloaded and delete the cache file.

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

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

5.14 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](#) ()

5.14.1 Detailed Description

[FileCacheHash](#) provides methods to make hashes from strings. Currently the SHA-1 hash from the openssl library is used.

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

- FileCacheHash.h

5.15 Arc::FileInfo Class Reference

[FileInfo](#) stores information about files (metadata).

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

5.15.1 Detailed Description

[FileInfo](#) stores information about files (metadata).

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

- FileInfo.h

5.16 Arc::URLMap Class Reference

Data Structures

- class [map_entry](#)

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

- URLMap.h

Index

ACCESS_LATENCY_LARGE
 Arc::DataPoint, 24
ACCESS_LATENCY_SMALL
 Arc::DataPoint, 24
ACCESS_LATENCY_ZERO
 Arc::DataPoint, 23
add
 Arc::DataBuffer, 11
AddChecksumObject
 Arc::DataPoint, 24
 Arc::DataPointDirect, 35
 Arc::DataPointIndex, 41
AddDN
 Arc::FileCache, 57
AddLocation
 Arc::DataPoint, 24
 Arc::DataPointDirect, 36
 Arc::DataPointIndex, 42
AddURLOptions
 Arc::DataPoint, 25
Arc::CacheParameters, 9
Arc::DataBuffer, 9
 add, 11
 buffer_size, 11
 checksum_object, 11
 checksum_valid, 11
 DataBuffer, 10
 eof_read, 11
 eof_write, 12
 error, 12
 error_read, 12
 error_write, 12
 for_read, 12
 for_write, 13
 is_notwritten, 13
 is_read, 13
 is_written, 14
 set, 14
 wait_any, 14
Arc::DataCallback, 15
Arc::DataHandle, 15
 GetPoint, 17
Arc::DataMover, 17
 checks, 18
 force_to_meta, 18
 secure, 18
 set_default_max_inactivity_time, 18
 set_default_min_average_speed, 18
 set_default_min_speed, 19
 Transfer, 19
 verbose, 19
Arc::DataPoint, 20
 ACCESS_LATENCY_LARGE, 24
 ACCESS_LATENCY_SMALL, 24
 ACCESS_LATENCY_ZERO, 23
 AddChecksumObject, 24
 AddLocation, 24
 AddURLOptions, 25
 Check, 25
 CompareLocationMetadata, 25
 CompareMeta, 25
 CreateDirectory, 25
 CurrentLocationMetadata, 26
 DataPoint, 24
 DataPointAccessLatency, 23
 DataPointInfoType, 24
 FinishReading, 26
 FinishWriting, 26
 GetFailureReason, 26
 INFO_TYPE_ACCESS, 24
 INFO_TYPE_ALL, 24
 INFO_TYPE_CONTENT, 24
 INFO_TYPE_MINIMAL, 24
 INFO_TYPE_NAME, 24
 INFO_TYPE_REST, 24
 INFO_TYPE_STRUCT, 24
 INFO_TYPE_TIMES, 24
 INFO_TYPE_TYPE, 24
 List, 26
 NextLocation, 27
 Passive, 27
 PostRegister, 27
 PrepareReading, 27
 PrepareWriting, 28
 PreRegister, 28
 PreUnregister, 28
 ProvidesMeta, 29
 Range, 29
 ReadOutOfOrder, 29
 Registered, 29

- Rename, [29](#)
- Resolve, [29](#), [30](#)
- SetAdditionalChecks, [30](#)
- SetMeta, [30](#)
- SetSecure, [30](#)
- SetURL, [31](#)
- SortLocations, [31](#)
- StartReading, [31](#)
- StartWriting, [31](#)
- Stat, [31](#), [32](#)
- StopReading, [32](#)
- StopWriting, [32](#)
- Transfer3rdParty, [33](#)
- TransferLocations, [33](#)
- Unregister, [33](#)
- valid_url_options, [34](#)
- WriteOutOfOrder, [34](#)
- Arc::DataPointDirect, [34](#)
 - AddChecksumObject, [35](#)
 - AddLocation, [36](#)
 - CompareLocationMetadata, [36](#)
 - CurrentLocationMetadata, [36](#)
 - NextLocation, [36](#)
 - Passive, [36](#)
 - PostRegister, [36](#)
 - PreRegister, [37](#)
 - PreUnregister, [37](#)
 - ProvidesMeta, [37](#)
 - Range, [37](#)
 - ReadOutOfOrder, [38](#)
 - Registered, [38](#)
 - Resolve, [38](#)
 - SetAdditionalChecks, [38](#)
 - SetSecure, [39](#)
 - SortLocations, [39](#)
 - Stat, [39](#)
 - Unregister, [39](#)
 - WriteOutOfOrder, [40](#)
- Arc::DataPointIndex, [40](#)
 - AddChecksumObject, [41](#)
 - AddLocation, [42](#)
 - Check, [42](#)
 - CompareLocationMetadata, [42](#)
 - CurrentLocationMetadata, [42](#)
 - FinishReading, [42](#)
 - FinishWriting, [43](#)
 - NextLocation, [43](#)
 - Passive, [43](#)
 - PrepareReading, [43](#)
 - PrepareWriting, [43](#)
 - ProvidesMeta, [44](#)
 - Range, [44](#)
 - ReadOutOfOrder, [44](#)
 - Registered, [44](#)
 - SetAdditionalChecks, [44](#)
 - SetMeta, [45](#)
 - SetSecure, [45](#)
 - SortLocations, [45](#)
 - StartReading, [45](#)
 - StartWriting, [46](#)
 - StopReading, [46](#)
 - StopWriting, [46](#)
 - TransferLocations, [46](#)
 - WriteOutOfOrder, [47](#)
- Arc::DataPointLoader, [47](#)
- Arc::DataPointPluginArgument, [47](#)
- Arc::DataSpeed, [47](#)
 - DataSpeed, [48](#)
 - hold, [49](#)
 - set_base, [49](#)
 - set_max_data, [49](#)
 - set_max_inactivity_time, [49](#)
 - set_min_average_speed, [49](#)
 - set_min_speed, [49](#)
 - set_progress_indicator, [50](#)
 - transfer, [50](#)
 - verbose, [50](#)
- Arc::DataStatus, [50](#)
 - CacheError, [52](#)
 - CacheErrorRetryable, [54](#)
 - CheckError, [53](#)
 - CheckErrorRetryable, [54](#)
 - CreateDirectoryError, [53](#)
 - CreateDirectoryErrorRetryable, [54](#)
 - CredentialsExpiredError, [52](#)
 - DataStatus, [55](#)
 - DataStatusType, [52](#)
 - DeleteError, [52](#)
 - DeleteErrorRetryable, [54](#)
 - GenericError, [53](#)
 - GenericErrorRetryable, [55](#)
 - InconsistentMetadataError, [53](#)
 - IsReadingError, [53](#)
 - IsWritingError, [53](#)
 - ListError, [53](#)
 - ListErrorRetryable, [54](#)
 - ListNonDirError, [53](#)
 - LocationAlreadyExistsError, [53](#)
 - NoLocationError, [53](#)
 - NotInitializedError, [53](#)
 - NotSupportedForDirectDataPointsError, [53](#)
 - PostRegisterError, [52](#)
 - PostRegisterErrorRetryable, [54](#)
 - PreRegisterError, [52](#)
 - PreRegisterErrorRetryable, [54](#)
 - ReadAcquireError, [52](#)
 - ReadAcquireErrorRetryable, [53](#)
 - ReadError, [52](#)

- ReadErrorRetryable, 54
- ReadFinishError, 53
- ReadFinishErrorRetryable, 54
- ReadPrepareError, 53
- ReadPrepareErrorRetryable, 54
- ReadPrepareWait, 53
- ReadResolveError, 52
- ReadResolveErrorRetryable, 53
- ReadStartError, 52
- ReadStartErrorRetryable, 53
- ReadStopError, 52
- ReadStopErrorRetryable, 54
- RenameError, 53
- RenameErrorRetryable, 55
- Retryable, 55
- StageError, 53
- StageErrorRetryable, 54
- StatError, 53
- StatErrorRetryable, 54
- StatNotPresentError, 53
- Success, 52
- SuccessCached, 53
- SuccessCancelled, 53
- SystemError, 53
- TransferError, 52
- TransferErrorRetryable, 54
- UnimplementedError, 53
- UnknownError, 53
- UnregisterError, 52
- UnregisterErrorRetryable, 54
- WriteAcquireError, 52
- WriteAcquireErrorRetryable, 53
- WriteError, 52
- WriteErrorRetryable, 54
- WriteFinishError, 53
- WriteFinishErrorRetryable, 54
- WritePrepareError, 53
- WritePrepareErrorRetryable, 54
- WritePrepareWait, 53
- WriteResolveError, 52
- WriteResolveErrorRetryable, 53
- WriteStartError, 52
- WriteStartErrorRetryable, 53
- WriteStopError, 52
- WriteStopErrorRetryable, 54
- Arc::FileCache, 55
 - AddDN, 57
 - CheckCreated, 57
 - CheckDN, 57
 - CheckValid, 58
 - File, 58
 - FileCache, 56, 57
 - GetCreated, 58
 - GetValid, 58
 - Link, 58
 - Release, 59
 - SetValid, 59
 - Start, 59
 - Stop, 60
 - StopAndDelete, 60
- Arc::FileCacheHash, 60
- Arc::FileInfo, 61
- Arc::URLMap, 61
- buffer_size
 - Arc::DataBuffer, 11
- CacheError
 - Arc::DataStatus, 52
- CacheErrorRetryable
 - Arc::DataStatus, 54
- Check
 - Arc::DataPoint, 25
 - Arc::DataPointIndex, 42
- CheckCreated
 - Arc::FileCache, 57
- CheckDN
 - Arc::FileCache, 57
- CheckError
 - Arc::DataStatus, 53
- CheckErrorRetryable
 - Arc::DataStatus, 54
- checks
 - Arc::DataMover, 18
- checksum_object
 - Arc::DataBuffer, 11
- checksum_valid
 - Arc::DataBuffer, 11
- CheckValid
 - Arc::FileCache, 58
- CompareLocationMetadata
 - Arc::DataPoint, 25
 - Arc::DataPointDirect, 36
 - Arc::DataPointIndex, 42
- CompareMeta
 - Arc::DataPoint, 25
- CreateDirectory
 - Arc::DataPoint, 25
- CreateDirectoryError
 - Arc::DataStatus, 53
- CreateDirectoryErrorRetryable
 - Arc::DataStatus, 54
- CredentialsExpiredError
 - Arc::DataStatus, 52
- CurrentLocationMetadata
 - Arc::DataPoint, 26
 - Arc::DataPointDirect, 36
 - Arc::DataPointIndex, 42

- DataBuffer
 - Arc::DataBuffer, 10
- DataPoint
 - Arc::DataPoint, 24
- DataPointAccessLatency
 - Arc::DataPoint, 23
- DataPointInfoType
 - Arc::DataPoint, 24
- DataSpeed
 - Arc::DataSpeed, 48
- DataStatus
 - Arc::DataStatus, 55
- DataStatusType
 - Arc::DataStatus, 52
- DeleteError
 - Arc::DataStatus, 52
- DeleteErrorRetryable
 - Arc::DataStatus, 54
- eof_read
 - Arc::DataBuffer, 11
- eof_write
 - Arc::DataBuffer, 12
- error
 - Arc::DataBuffer, 12
- error_read
 - Arc::DataBuffer, 12
- error_write
 - Arc::DataBuffer, 12
- File
 - Arc::FileCache, 58
- FileCache
 - Arc::FileCache, 56, 57
- FinishReading
 - Arc::DataPoint, 26
 - Arc::DataPointIndex, 42
- FinishWriting
 - Arc::DataPoint, 26
 - Arc::DataPointIndex, 43
- for_read
 - Arc::DataBuffer, 12
- for_write
 - Arc::DataBuffer, 13
- force_to_meta
 - Arc::DataMover, 18
- GenericError
 - Arc::DataStatus, 53
- GenericErrorRetryable
 - Arc::DataStatus, 55
- GetCreated
 - Arc::FileCache, 58
- GetFailureReason
 - Arc::DataPoint, 26
- GetPoint
 - Arc::DataHandle, 17
- GetValid
 - Arc::FileCache, 58
- hold
 - Arc::DataSpeed, 49
- InconsistentMetadataError
 - Arc::DataStatus, 53
- INFO_TYPE_ACCESS
 - Arc::DataPoint, 24
- INFO_TYPE_ALL
 - Arc::DataPoint, 24
- INFO_TYPE_CONTENT
 - Arc::DataPoint, 24
- INFO_TYPE_MINIMAL
 - Arc::DataPoint, 24
- INFO_TYPE_NAME
 - Arc::DataPoint, 24
- INFO_TYPE_REST
 - Arc::DataPoint, 24
- INFO_TYPE_STRUCT
 - Arc::DataPoint, 24
- INFO_TYPE_TIMES
 - Arc::DataPoint, 24
- INFO_TYPE_TYPE
 - Arc::DataPoint, 24
- is_notwritten
 - Arc::DataBuffer, 13
- is_read
 - Arc::DataBuffer, 13
- is_written
 - Arc::DataBuffer, 14
- IsReadingError
 - Arc::DataStatus, 53
- IsWritingError
 - Arc::DataStatus, 53
- Link
 - Arc::FileCache, 58
- List
 - Arc::DataPoint, 26
- ListError
 - Arc::DataStatus, 53
- ListErrorRetryable
 - Arc::DataStatus, 54
- ListNonDirError
 - Arc::DataStatus, 53
- LocationAlreadyExistsError
 - Arc::DataStatus, 53
- NextLocation

- Arc::DataPoint, 27
- Arc::DataPointDirect, 36
- Arc::DataPointIndex, 43
- NoLocationError
 - Arc::DataStatus, 53
- NotInitializedError
 - Arc::DataStatus, 53
- NotSupportedForDirectDataPointsError
 - Arc::DataStatus, 53
- Passive
 - Arc::DataPoint, 27
 - Arc::DataPointDirect, 36
 - Arc::DataPointIndex, 43
- PostRegister
 - Arc::DataPoint, 27
 - Arc::DataPointDirect, 36
- PostRegisterError
 - Arc::DataStatus, 52
- PostRegisterErrorRetryable
 - Arc::DataStatus, 54
- PrepareReading
 - Arc::DataPoint, 27
 - Arc::DataPointIndex, 43
- PrepareWriting
 - Arc::DataPoint, 28
 - Arc::DataPointIndex, 43
- PreRegister
 - Arc::DataPoint, 28
 - Arc::DataPointDirect, 37
- PreRegisterError
 - Arc::DataStatus, 52
- PreRegisterErrorRetryable
 - Arc::DataStatus, 54
- PreUnregister
 - Arc::DataPoint, 28
 - Arc::DataPointDirect, 37
- ProvidesMeta
 - Arc::DataPoint, 29
 - Arc::DataPointDirect, 37
 - Arc::DataPointIndex, 44
- Range
 - Arc::DataPoint, 29
 - Arc::DataPointDirect, 37
 - Arc::DataPointIndex, 44
- ReadAcquireError
 - Arc::DataStatus, 52
- ReadAcquireErrorRetryable
 - Arc::DataStatus, 53
- ReadError
 - Arc::DataStatus, 52
- ReadErrorRetryable
 - Arc::DataStatus, 54
- ReadFinishError
 - Arc::DataStatus, 53
- ReadFinishErrorRetryable
 - Arc::DataStatus, 54
- ReadOutOfOrder
 - Arc::DataPoint, 29
 - Arc::DataPointDirect, 38
 - Arc::DataPointIndex, 44
- ReadPrepareError
 - Arc::DataStatus, 53
- ReadPrepareErrorRetryable
 - Arc::DataStatus, 54
- ReadPrepareWait
 - Arc::DataStatus, 53
- ReadResolveError
 - Arc::DataStatus, 52
- ReadResolveErrorRetryable
 - Arc::DataStatus, 53
- ReadStartError
 - Arc::DataStatus, 52
- ReadStartErrorRetryable
 - Arc::DataStatus, 53
- ReadStopError
 - Arc::DataStatus, 52
- ReadStopErrorRetryable
 - Arc::DataStatus, 54
- Registered
 - Arc::DataPoint, 29
 - Arc::DataPointDirect, 38
 - Arc::DataPointIndex, 44
- Release
 - Arc::FileCache, 59
- Rename
 - Arc::DataPoint, 29
- RenameError
 - Arc::DataStatus, 53
- RenameErrorRetryable
 - Arc::DataStatus, 55
- Resolve
 - Arc::DataPoint, 29, 30
 - Arc::DataPointDirect, 38
- Retryable
 - Arc::DataStatus, 55
- secure
 - Arc::DataMover, 18
- set
 - Arc::DataBuffer, 14
- set_base
 - Arc::DataSpeed, 49
- set_default_max_inactivity_time
 - Arc::DataMover, 18
- set_default_min_average_speed
 - Arc::DataMover, 18

- set_default_min_speed
 - Arc::DataMover, 19
- set_max_data
 - Arc::DataSpeed, 49
- set_max_inactivity_time
 - Arc::DataSpeed, 49
- set_min_average_speed
 - Arc::DataSpeed, 49
- set_min_speed
 - Arc::DataSpeed, 49
- set_progress_indicator
 - Arc::DataSpeed, 50
- SetAdditionalChecks
 - Arc::DataPoint, 30
 - Arc::DataPointDirect, 38
 - Arc::DataPointIndex, 44
- SetMeta
 - Arc::DataPoint, 30
 - Arc::DataPointIndex, 45
- SetSecure
 - Arc::DataPoint, 30
 - Arc::DataPointDirect, 39
 - Arc::DataPointIndex, 45
- SetURL
 - Arc::DataPoint, 31
- SetValid
 - Arc::FileCache, 59
- SortLocations
 - Arc::DataPoint, 31
 - Arc::DataPointDirect, 39
 - Arc::DataPointIndex, 45
- StageError
 - Arc::DataStatus, 53
- StageErrorRetryable
 - Arc::DataStatus, 54
- Start
 - Arc::FileCache, 59
- StartReading
 - Arc::DataPoint, 31
 - Arc::DataPointIndex, 45
- StartWriting
 - Arc::DataPoint, 31
 - Arc::DataPointIndex, 46
- Stat
 - Arc::DataPoint, 31, 32
 - Arc::DataPointDirect, 39
- StatError
 - Arc::DataStatus, 53
- StatErrorRetryable
 - Arc::DataStatus, 54
- StatNotPresentError
 - Arc::DataStatus, 53
- Stop
 - Arc::FileCache, 60
- StopAndDelete
 - Arc::FileCache, 60
- StopReading
 - Arc::DataPoint, 32
 - Arc::DataPointIndex, 46
- StopWriting
 - Arc::DataPoint, 32
 - Arc::DataPointIndex, 46
- Success
 - Arc::DataStatus, 52
- SuccessCached
 - Arc::DataStatus, 53
- SuccessCancelled
 - Arc::DataStatus, 53
- SystemError
 - Arc::DataStatus, 53
- Transfer
 - Arc::DataMover, 19
- transfer
 - Arc::DataSpeed, 50
- Transfer3rdParty
 - Arc::DataPoint, 33
- TransferError
 - Arc::DataStatus, 52
- TransferErrorRetryable
 - Arc::DataStatus, 54
- TransferLocations
 - Arc::DataPoint, 33
 - Arc::DataPointIndex, 46
- UnimplementedError
 - Arc::DataStatus, 53
- UnknownError
 - Arc::DataStatus, 53
- Unregister
 - Arc::DataPoint, 33
 - Arc::DataPointDirect, 39
- UnregisterError
 - Arc::DataStatus, 52
- UnregisterErrorRetryable
 - Arc::DataStatus, 54
- valid_url_options
 - Arc::DataPoint, 34
- verbose
 - Arc::DataMover, 19
 - Arc::DataSpeed, 50
- wait_any
 - Arc::DataBuffer, 14
- WriteAcquireError
 - Arc::DataStatus, 52
- WriteAcquireErrorRetryable

- Arc::DataStatus, [53](#)
- WriteError
 - Arc::DataStatus, [52](#)
- WriteErrorRetryable
 - Arc::DataStatus, [54](#)
- WriteFinishError
 - Arc::DataStatus, [53](#)
- WriteFinishErrorRetryable
 - Arc::DataStatus, [54](#)
- WriteOutOfOrder
 - Arc::DataPoint, [34](#)
 - Arc::DataPointDirect, [40](#)
 - Arc::DataPointIndex, [47](#)
- WritePrepareError
 - Arc::DataStatus, [53](#)
- WritePrepareErrorRetryable
 - Arc::DataStatus, [54](#)
- WritePrepareWait
 - Arc::DataStatus, [53](#)
- WriteResolveError
 - Arc::DataStatus, [52](#)
- WriteResolveErrorRetryable
 - Arc::DataStatus, [53](#)
- WriteStartError
 - Arc::DataStatus, [52](#)
- WriteStartErrorRetryable
 - Arc::DataStatus, [53](#)
- WriteStopError
 - Arc::DataStatus, [52](#)
- WriteStopErrorRetryable
 - Arc::DataStatus, [54](#)