xrootdlib Documentation

Release 0.2.0

Max Fischer

Sep 23, 2021

Library Overview

CHAPTER 1

xrootdlib

1.1 xrootdlib package

1.1.1 Subpackages

xrootdlib.streams package

Subpackages

xrootdlib.streams.XrdXrootdMon package

Stream of the XrdXrootdMon monitoring messages

This module presents a high-level stream representation of the *XrdXrootdMon* `struct` primitives. It provides two stream mechanisms of different complexity:

- *stream_packets* provides an ordered stream of raw *Packet*
- `map_streams` provides a stream of *fstat*, *redir*, *trace*, and *map*

class stream_packets(**slave_args*, ***slave_kwargs*)
Bases: `chainlet.genlink.GeneratorLink`

Provide a stream of packets from a readable bytes buffer

Parameters

- **packet_source** – a bytes buffer providing serialised packet data
- **sort_window** – window size in which to sort packets by their assigned order

The `packet_source` may be any bytes buffer supporting `read` operations. This can be an open file, but also a socket or wrapped memory. While the buffer may be extended indefinitely, it should *always* contain complete packets.

The `sort_window` is required to ensure ordering of the packet data. Since XRootD uses several packet buffers, concurrent packets may arrive in arbitrary order. To remove this ambiguity, `sort_window` packets are buffered and provided in order.

Note that an ideal `sort_window` corresponds to the *number* of out-of-order packets. It is *not* to the maximum difference in ordering of each packet.

map_streams
alias of `xrootdlib.streams.XrdXrootdMon.StreamMapper`

Submodules

xrootdlib.streams.XrdXrootdMon.fstat module

class Close(*client: Union[xrootdlib.streams.XrdXrootdMon.map.UserInfo, xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo], lfn: bytes, stats: xrootdlib.structs.XrdXrootdMon.fstat.FileCLS*)
Bases: `object`

A client closed a file

client

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.fstat.FileCLS, stod: int, map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

lfn

stats

class Disconnect(*client: xrootdlib.streams.XrdXrootdMon.map.UserInfo*)
Bases: `object`

A client disconnected from the server

client

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.fstat.FileDSC, stod: int, map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

class FstatWindow(*server_info: xrootdlib.streams.XrdXrootdMon.map.ServerInfo, start: int, end: int, records: List[Union[xrootdlib.streams.XrdXrootdMon.fstat.Disconnect, xrootdlib.streams.XrdXrootdMon.fstat.Open, xrootdlib.streams.XrdXrootdMon.fstat.Close, xrootdlib.streams.XrdXrootdMon.fstat.Transfer]]*)
Bases: `object`

Sequence of Open, Close and Disconnect events in a time window

end

records

server_info

start

class Open(*client: Union[xrootdlib.streams.XrdXrootdMon.map.UserInfo, xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo], lfn: bytes, readwrite: bool, filesize: int*)
Bases: `object`

A client opened a file

client

filesize

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.fstat.FileOPN*, *stod: int*, *map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

lfn

readwrite

class Transfer(*client: Union[xrootdlib.streams.XrdXrootdMon.map.UserInfo, xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo]*, *lfn: bytes*, *stats: xrootdlib.structs.XrdXrootdMon.fstat.FileXFR*)

Bases: object

A client transfered a file

client

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.fstat.FileXFR*, *stod: int*, *map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

lfn

stats

digest_packet(*stod: int*, *fstat_struct: xrootdlib.structs.XrdXrootdMon.Fstat*, *map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

Digest a packet containing fstat data

xrootdlib.streams.XrdXrootdMon.map module

exception MapInfoError

Bases: Exception

An item was not in the map store

class MapInfoStore

Bases: object

digest_map(*stod: int*, *map_struct: xrootdlib.structs.XrdXrootdMon.Map*)

Digest map data from a packet

free_path(*stod: int*, *dictid: int*) → None

free_user(*stod: int*, *dictid: int*) → None

get_path(*stod: int*, *dictid: int*) → xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo

get_server(*stod: int*, *sid: int*) → xrootdlib.streams.XrdXrootdMon.map.ServerInfo

get_user(*stod: int*, *dictid: int*) → xrootdlib.streams.XrdXrootdMon.map.UserInfo

class MapInfoStoreCleaner(*store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*, *clean_delay: int = 30*)

Bases: threading.Thread

add_deletion(*instruction*, **args*, ***kwargs*)

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class PathAccessInfo(*user_id: xrootdlib.structs.XrdXrootdMon.map.UserId*, *server_info: xrootdlib.streams.XrdXrootdMon.map.ServerInfo*, *path_info: xrootdlib.structs.XrdXrootdMon.map.Path*)

Bases: object

auth_info = None

host

path

pid

protocol

server

user

class ServerInfo(*user_id: xrootdlib.structs.XrdXrootdMon.map.UserId*, *server_info: xrootdlib.structs.XrdXrootdMon.map.SrvInfo*)

Bases: object

host

instance

pid

port

program

protocol

sid

site

user

version

class UserInfo(*user_id: xrootdlib.structs.XrdXrootdMon.map.UserId*, *server_info: xrootdlib.streams.XrdXrootdMon.map.ServerInfo*, *auth_info: xrootdlib.structs.XrdXrootdMon.map.AuthInfo = None*)

Bases: object

auth_info

host

pid

protocol

server

user

xrootdlib.streams.XrdXrootdMon.redir module

class CmsdRedir(*action: xrootdlib.structs.XrdXrootdMon.redir.XROOTD_MON*, *target: bytes*, *port: int*, *client: xrootdlib.streams.XrdXrootdMon.map.UserInfo*, *path: bytes*)

Bases: *xrootdlib.streams.XrdXrootdMon.redir.Redirection*

class RedirWindow(*server_info: xrootdlib.streams.XrdXrootdMon.map.ServerInfo, start: int, end: int, records: List[xrootdlib.streams.XrdXrootdMon.redir.Redirection]*)

Bases: object

Sequence of events in a time window

end

records

server_info

start

class Redirection(*action: xrootdlib.structs.XrdXrootdMon.redir.XROOTD_MON, target: bytes, port: int, client: xrootdlib.streams.XrdXrootdMon.map.UserInfo, path: bytes*)

Bases: object

action

client

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.redir.Redirect, stod: int, map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

path

port

target

class XrootdRedir(*action: xrootdlib.structs.XrdXrootdMon.redir.XROOTD_MON, target: bytes, port: int, client: xrootdlib.streams.XrdXrootdMon.map.UserInfo, path: bytes*)

Bases: xrootdlib.streams.XrdXrootdMon.redir.Redirection

digest_packet(*stod: int, burr_struct: xrootdlib.structs.XrdXrootdMon.Burr, map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

Digest a packet containing redir data

xrootdlib.streams.XrdXrootdMon.trace module

class Close(*client: Union[xrootdlib.streams.XrdXrootdMon.map.UserInfo, xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo], lfn: bytes, rtot: int, wtot: int*)

Bases: object

A client closed a file

client

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.trace.Close, stod: int, map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

lfn

rtot

wtot

class Disconnect(*client: xrootdlib.streams.XrdXrootdMon.map.UserInfo, duration: int, forced: bool*)

Bases: object

A client disconnected from the server

client

duration

forced

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.trace.Disc*, *stod: int*, *map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

class Open(*client: Union[xrootdlib.streams.XrdXrootdMon.map.UserInfo, xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo]*, *lfn: bytes*, *filesize: int*)

Bases: object

A client opened a file

client

filesize

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.trace.Open*, *stod: int*, *map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

lfn

class ReadVector(*client: Union[xrootdlib.streams.XrdXrootdMon.map.UserInfo, xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo]*, *lfn: bytes*, *reads: List[int]*)

Bases: object

client

lfn

reads

class ReadWrite(*client: Union[xrootdlib.streams.XrdXrootdMon.map.UserInfo, xrootdlib.streams.XrdXrootdMon.map.PathAccessInfo]*, *lfn: bytes*, *offset: int*, *read: int*, *write: int*)

Bases: object

A client read from or wrote to a file

client

classmethod from_record(*record_struct: xrootdlib.structs.XrdXrootdMon.trace.ReadWrite*, *stod: int*, *map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

lfn

offset

read

write

class TraceWindow(*server_info: xrootdlib.streams.XrdXrootdMon.map.ServerInfo*, *start: int*, *end: int*, *records: List[T]*)

Bases: object

Sequence of events in a time window

end

records

server_info

start

digest_packet(*stod: int*, *buff_struct: xrootdlib.structs.XrdXrootdMon.Buff*, *map_store: xrootdlib.streams.XrdXrootdMon.map.MapInfoStore*)

Digest a packet containing trace data

ignore_not_implemented(*record_struct*, *stod*, *map_store*)
Skip structs that are currently not implemented

xrootdlib.streams.XrdXrootdMon.utility module

class PSeq(*pseq: int*)
Bases: `object`

Sortable *Packet Sequence*, an Integer from a wrapping (0, 255) range

Parameters **pseq** – the `pseq` of a packet

This represents the XRootD Packet Sequence for the purpose of comparisons. It ensures that comparisons respects wrapping from 255 to 0. In effect, a high-valued PSeq compares *less than* a low-valued PSeq.

Warning This class does not implement a full Integer interface.

exception PacketBufferExhausted
Bases: `Exception`

The buffer of packet data is exhausted

packet_from_buffer(*packet_source: IO[bytes]*)
Read a packet from a bytes buffer

xrootdlib.structs package

Representation of XRootD structs

Subpackages

xrootdlib.structs.XrdXrootdMon package

Structs used for the *Detailed Monitoring Data Format* streams sent by servers. See the `all.monitor` directive and XRootD Monitoring for details.

All types implement a `Type[T].from_XXX(buffer: bytes) -> T` constructor method, where XXX describes the appropriate section. These section are `buffer` or `record`, which represent the stream buffer either at the start of a packet or the start of a record. Unless you explicitly have a need otherwise, use *`Packet.from_buffer()`* to read an entire packet at a time.

class Buff(*records: List[Union[xrootdlib.structs.XrdXrootdMon.trace.AppId, xrootdlib.structs.XrdXrootdMon.trace.Close, xrootdlib.structs.XrdXrootdMon.trace.Disc, xrootdlib.structs.XrdXrootdMon.trace.Open, xrootdlib.structs.XrdXrootdMon.trace.ReadWrite, xrootdlib.structs.XrdXrootdMon.trace.ReadU, xrootdlib.structs.XrdXrootdMon.trace.ReadV, xrootdlib.structs.XrdXrootdMon.trace.Window]]*)
Bases: `object`

`XrdXrootdMonBuff` ("t-stream") describing trace events

Parameters **records** – individual file operation events

The `records` field contains various trace records framed by window marks (*`Window`*). In other words, `records` is a *flat* sequence of one *or more* sequences of records, with marks at the start, end and between sequences.

classmethod from_record(*record_data: bytes*, *record_code: bytes = b't'*) → xrootdlib.structs.XrdXrootdMon.Buff

Extract the record from the record portion of a stream packet buffer

Parameters

- **record_data** – buffer at the start of the record of a monitor stream packet
- **record_code** – the *code* field for this packet

Note The only valid record code for this class is `b't'`.

payload_dispath = {<XROOTD_MON.OPEN: 128>: <class 'xrootdlib.structs.XrdXrootdMon.tra

records

class Burr(*sid: xrootdlib.structs.XrdXrootdMon.redir.ServerIdent*, *records: List[Union[xrootdlib.structs.XrdXrootdMon.redir.Redirect, xrootdlib.structs.XrdXrootdMon.redir.ServerIdent, xrootdlib.structs.XrdXrootdMon.redir.WindowMark]]*)

Bases: `object`

`XrdXrootdMonBurr` ("r-stream") describing redirection events

Parameters

- **sid** – identification of the server sending events
- **records** – individual operations requested by clients

The `records` field contains redirection records (*Redirect*) framed by window marks (*WindowMark*). In other words, `records` is a *flat* sequence of one *or more* sequences of records, with marks at the start, end and between sequences.

end

classmethod from_record(*record_data: bytes*, *record_code: bytes = b'r'*) → xrootdlib.structs.XrdXrootdMon.Burr

Extract the record from the record portion of a stream packet buffer

Parameters

- **record_data** – buffer at the start of the record of a monitor stream packet
- **record_code** – the *code* field for this packet

Note The only valid record code for this class is `b'r'`.

records

sid

start

class Fstat(*tod: xrootdlib.structs.XrdXrootdMon.fstat.FileTOD*, *records: List[Union[xrootdlib.structs.XrdXrootdMon.fstat.FileTOD, xrootdlib.structs.XrdXrootdMon.fstat.FileDSC, xrootdlib.structs.XrdXrootdMon.fstat.FileOPN, xrootdlib.structs.XrdXrootdMon.fstat.FileCLS, xrootdlib.structs.XrdXrootdMon.fstat.FileXFR]]*)

Bases: `object`

`XrdXrootdMonFstat` ("f-stream") describing the general file access

Parameters

- **tod** – identifier for the server and time window
- **records** – file operations and statistics

The `records` is a flat sequence of open records (*FileOPN*), transfer records (*FileXFR*, if `fstat xfr` is configured), close records (*FileCLS*), and disconnect records (*FileDSC*). As per the specification, for every access the records are provided in this order. However, records for one access may be spread over multiple time windows.

end

classmethod from_record(*record_data: bytes*, *record_code: bytes = b'f'*) → xrootdlib.structs.XrdXrootdMon.Fstat

Extract the record from the record portion of a stream packet buffer

Parameters

- **record_data** – buffer at the start of the record of a monitor stream packet
- **record_code** – the *code* field for this packet

Note The only valid record code for this class is `b'f'`.

payload_dispath = {<recType.isClose: 0>: <class 'xrootdlib.structs.XrdXrootdMon.fsta

records

start

tod

class Header(*code: bytes*, *pseq: int*, *plen: int*, *stod: int*)

Bases: `object`

`XrdXrootdMonHeader` shared by all packets

Parameters

- **code** – identifier for the record type
- **pseq** – wrapping counter for packet sequence
- **plen** – size of the packet in bytes
- **stod** – daemon start timestamp

code

classmethod from_buffer(*buffer: bytes*) → xrootdlib.structs.XrdXrootdMon.Header

Extract the header from the start of a stream packet buffer

Parameters **buffer** – buffer containing a monitor stream packet

plen

pseq

size = 8

stod

struct_parser = <Struct object>

class Map(*dictid: int*, *userid: xrootdlib.structs.XrdXrootdMon.map.UserId*, *payload: Union[xrootdlib.structs.XrdXrootdMon.map.SrvInfo, xrootdlib.structs.XrdXrootdMon.map.Path, xrootdlib.structs.XrdXrootdMon.map.AppInfo, xrootdlib.structs.XrdXrootdMon.map.PrgInfo, xrootdlib.structs.XrdXrootdMon.map.AuthInfo, xrootdlib.structs.XrdXrootdMon.map.XfrInfo]*)

Bases: `object`

`XrdXrootdMonMap` describing transactions and general information

Parameters

- **dictid** – identifier shared by all records referring to the same information
- **userid** – identifier for the client session being monitored
- **payload** – the actual information of this message

The *Map* provides general information that applies across several monitoring events. Events of other streams reference this with the `dictid`, or the `sid` of the *UserId* of *SrvInfo* payloads. Note that in case of *SrvInfo* payloads, the `userid` contains the *server* user data.

dictid

classmethod from_record(*record_data: bytes*, *record_code: bytes*) → xrootdlib.structs.XrdXrootdMon.Map

Extract the record from the record portion of a stream packet buffer

Parameters

- **record_data** – buffer at the start of the record of a monitor stream packet
- **record_code** – the *code* field for this packet

payload

userid

class Packet(*header: xrootdlib.structs.XrdXrootdMon.Header*, *record: Union[xrootdlib.structs.XrdXrootdMon.Map, xrootdlib.structs.XrdXrootdMon.Burr, xrootdlib.structs.XrdXrootdMon.Fstat, xrootdlib.structs.XrdXrootdMon.Buff]*)

Bases: `object`

`XrdXrootdMon` packet for a map, r, t or f stream

Parameters

- **header** – the header specifying type, ordering and size of the packet
- **record** – the actual information carried by the packet

classmethod from_buffer(*buffer: bytes*)

Extract the entire packet from the start of a stream packet buffer

Parameters **buffer** – buffer containing a monitor stream packet

header

record

record_dispath = {b'=': <class 'xrootdlib.structs.XrdXrootdMon.Map'>, b'd': <class '

size

PacketRecord = typing.Union[xrootdlib.structs.XrdXrootdMon.Map, xrootdlib.structs.XrdXrootd

Record types in a packet

Submodules

xrootdlib.structs.XrdXrootdMon.constants module

XROOTD_MON_SIDMASK = 281474976710655

The server ID is encoded in the lower 48 bits of the first 8 bytes

xrootdlib.structs.XrdXrootdMon.fstat module

class FileCLS(*flags: int, size: int, fileid: int, read: int, readv: int, write: int, ops: Optional[xrootdlib.structs.XrdXrootdMon.fstat.StatOPS], ssq: Optional[xrootdlib.structs.XrdXrootdMon.fstat.StatSSQ]*)

Bases: `object`

`XrdXrootdMonFileCLS` indicating that a client closed a file

Parameters

- **flags** – indicator for fields present and close type
- **fileid** – file identifier (see *Map*)
- **read** – bytes read using `read()`
- **readv** – bytes read using `readv()`
- **write** – bytes written
- **ops** – file operation statistics
- **ssq** – file operation statistic deviations

fileid

flags

classmethod from_buffer(*buffer: bytes*)

ops

read

readv

size

ssq

struct_parser = <Struct object>

write

xfr

class FileDSC(*flags: int*, *dictid: str*)

Bases: `object`

`XrdXrootdMonFileDSC` indicating that a client disconnected from the server

Parameters

- **flags** – unused for this record type
- **dictid** – client identifier (see *Map*)

dictid

flags

classmethod from_buffer(*buffer: bytes*)

size = 8

struct_parser = <Struct object>

class FileLFNView (*file_opn: xrootdlib.structs.XrdXrootdMon.fstat.FileOPN*)
Bases: object

lfn

user

class FileOPN (*flags: int*, *size: int*, *fileid: int*, *filesize: int*, *user: Optional[int] = None*, *lfn: Optional[bytes] = None*)
Bases: object

XrdXrootdMonFileOPN indicating that a client opened a file

Parameters

- **flags** – indicator for fields present and access type
- **size** – size of the struct in bytes
- **fileid** – file identifier (see *Map*)
- **filesize** – size of the file in bytes
- **user** – client identifier (see *Map*)
- **lfn** – the (logical) path of the file

fileid

filesize

flags

classmethod from_buffer (*buffer: bytes*)

fsz

lfn

size

struct_parser = <Struct object>

ufn

user

FileRecord = typing.Union[xrootdlib.structs.XrdXrootdMon.fstat.FileTOD, xrootdlib.structs.X
Type of records in the f stream

class FileTOD (*flags: int*, *records_xfr: int*, *records_total: int*, *start: int*, *end: int*, *sid: int*)
Bases: object

XrdXrootdMonFileTOD identifying the sender and time range

Parameters

- **flags** – indicator for fields present
- **records_xfr** – number of *FileXFR* records in the packet
- **records_total** – number of *FileRecord* records in the packet
- **start** – timestamp of the first record
- **end** – timestamp the packet was sent
- **sid** – server identifier (see *Map*)

end

flags

classmethod from_buffer(*buffer: bytes*)

records_total

records_xfr

sid

size = 24

start

struct_parser = <Struct object>

class FileXFR(*flags: int*, *fileid: int*, *read: int*, *readv: int*, *write: int*)

Bases: `object`

`XrdXrootdMonFileXFR` indicating file transfer statistics

Parameters

- **flags** – unused
- **fileid** – client identifier (see *Map*)
- **read** – bytes read using `read()`
- **readv** – bytes read using `readv()`
- **write** – bytes written

fileid

flags

classmethod from_buffer(*buffer: bytes*)

read

readv

size = 32

struct_parser = <Struct object>

write

xfr

class StatOPS(*read: int*, *readv: int*, *write: int*, *rsmin: int*, *rsmax: int*, *rsegs: int*, *rdmin: int*, *rdmax: int*, *rvmin: int*, *rvmax: int*, *wrmin: int*, *wrmax: int*)

Bases: `object`

`XrdXrootdMonStatOPS` describing file operation statistics

classmethod from_buffer(*buffer: bytes*)

rdmax

rdmin

read

readv

rsegs

rsmax

rsmin

rvmax

rvmin

size = 48

struct_parser = <Struct object>

write

wrmax

wrmin

class StatSSQ(*read: int*, *readv: int*, *rsegs: int*, *write: int*)

Bases: `object`

`XrdXrootdMonStatSSQ` describing file operation statistic deviations

classmethod from_buffer(*buffer: bytes*)

read

readv

rsegs

size = 32

struct_parser = <Struct object>

write

class StatXFRView(*file_struct: Union[xrootdlib.structs.XrdXrootdMon.fstat.FileCLS, xrootdlib.structs.XrdXrootdMon.fstat.FileXFR]*)

Bases: `object`

read

readv

write

class recFval

Bases: `int`, `enum.Enum`

Record flags for the f stream

forced = 1
Disconnect prior to close

hasLFN = 1
XrdXroodMonFileLFN present

hasOPS = 2
XrdXroodMonFileOPS present

hasRW = 2
FileRecord opened for reads & writes

hasSID = 1
The sID member is present

hasSSQ = 4
XrdXroodMonFileSSQ present (implies `hasOPS`)

class recType
Bases: int, enum.Enum

Record type identifiers for the f stream

isClose = 0
XrdXrootdMonFileCLS

isDisc = 4
XrdXrootdMonFileDSC

isOpen = 1
XrdXrootdMonFileOPN

isTime = 2
XrdXrootdMonFileTOD

isXFR = 3
XrdXrootdMonFileXFR

xrootdlib.structs.XrdXrootdMon.map module

Contents of the XrdXrootdMonMap struct

Each struct contains a `userid/npayload` info field. The `userid` is always represented by *UserId*, while the `payload` can be any of the types of `MapPayload`.

Each class is capable of parsing its respective *portion* of the info field. That is, it expects a buffer (a `bytes`, `bytesarray` or `memoryview`) starting with the `XrdXrootdMonMap` info; trailing buffer content is allowed and the buffer is never modified.

class AppInfo(*appinfo: bytes*)
Bases: object

`*/appinfo` containing un-interpreted application supplied information

appinfo

classmethod from_buffer(*buffer: bytes*)

class AuthInfo(*protocol: bytes*, *name: bytes*, *host: bytes*, *organisation: bytes*, *role: bytes*, *group: bytes*, *m: bytes*, *executable: bytes*, *moninfo: bytes*)
Bases: object

`*/authinfo` describing an authenticating user

executable

classmethod from_buffer(*buffer: bytes*)

g

group

h

host

m

moninfo

n

name

o

organisation

p

protocol

r

role

x

y

class Path(*path: bytes*)

Bases: object

*/path containing full path name of a file being opened

classmethod from_buffer(*buffer: bytes*)

path

class PrgInfo(*xfn: bytes*, *tod: int*, *sz: int*, *at: int*, *ct: int*, *mt: int*, *fn: bytes*)

Bases: object

*/prginfo describing the purging of a file

at

ct

fn

classmethod from_buffer(*buffer: bytes*)

mt

sz

tod

xfn

class SrvInfo(*pgm: bytes*, *ver: bytes*, *inst: bytes*, *port: int*, *site: bytes*)

Bases: object

*/srvinfo describing the xrootd instance sending reports

classmethod from_buffer(*buffer: bytes*)

inst

pgm

port

site

ver

class UserId(*prot: bytes*, *user: bytes*, *pid: int*, *sid: int*, *host: bytes*)

Bases: object

userid/* describing the user performing an action

classmethod from_buffer(*buffer: bytes*)

host

pid

prot

sid

user

class XfrInfo(*lfn: bytes*, *tod: int*, *sz: int*, *tm: int*, *op: str*, *rc: int*, *pd: bytes*)

Bases: object

`*/xfrinfo` describing the transfer of a file

classmethod from_buffer(*buffer: bytes*)

lfn

op

pd

rc

sz

tm

tod

xrootdlib.structs.XrdXrootdMon.redir module

Redir = typing.Union[xrootdlib.structs.XrdXrootdMon.redir.Redirect, xrootdlib.structs.XrdX

Type of records in the r stream

class Redirect(*type: xrootdlib.structs.XrdXrootdMon.redir.XROOTD_MON*, *subtype: xrootdlib.structs.XrdXrootdMon.redir.XROOTD_MON*, *dent: int*, *port: int*, *dictid: int*, *server: bytes*, *path: bytes*)

Bases: object

XrdXrootdMonRedir representing a redirect record

Parameters

- **type** – the type of the record, i.e. REDIRECT or `REDLOCAL`
- **subtype** – the requested operation, i.e. CHMOD, LOCALTE, ...
- **dent** – size of the struct in bytes plus one
- **port** – port to which the client was redirected
- **dictid** – client identifier (see *Map*)
- **server** – hostname or address of the target server
- **path** – path of the file on the target server

arg0

arg1

dent

dictid

classmethod from_buffer(*buffer: bytes*)

local
Whether the redirection target is the same server

path

port

server

size

struct_parser = <Struct object>

subtype

type

class RedirectArg0View(*redirect: xrootdlib.structs.XrdXrootdMon.redir.Redirect*)
Bases: object

dent

port

type

class RedirectArg1View(*redirect: xrootdlib.structs.XrdXrootdMon.redir.Redirect*)
Bases: object

dictid

path

server

serverpath

class ServerIdent(*sid: int*)
Bases: object

XrdXrootdMonRedir representing a server identification record

Parameters **sid** – server identifier (see *Map*)

classmethod from_buffer(*buffer: bytes*)

sid

size = 8

struct_parser = <Struct object>

type = 240

class WindowMark(*timestamp: int*, *prev_duration: int*)
Bases: object

XrdXrootdMonRedir representing a window timing mark

Parameters

- **timestamp** – timestamp when the *current* window started
- **prev_duration** – duration of the *previous* window

Note that windows are usually not adjacent: the *WindowMark* signifies the beginning of a *new* window. The previous window may have been closed an arbitrary time before.

To get the time range covered by a window, the prev_duration of the next window is required:

```
# window => tuple(start, end)
window_range[i] = (
    window_marks[i].timestamp,
    window_marks[i].timestamp + window_marks[i+1].prev_duration
)
```

arg0

arg1

classmethod from_buffer(*buffer: bytes*)

prev_duration

size = 8

struct_parser = <Struct object>

timestamp

type = 0

class WindowMarkArg0View(*window: xrootdlib.structs.XrdXrootdMon.redir.WindowMark*)
Bases: object

type

window

class WindowMarkArg1View(*window: xrootdlib.structs.XrdXrootdMon.redir.WindowMark*)
Bases: object

window

class XROOTD_MON
Bases: int, enum.Enum

XROOTD_MON_XYZ constants for the r-stream

CHMOD = 1
Change file mode

LOCATE = 2
Locate file or directory

MKDIR = 7
Create a directory or path

MV = 8
Rename a file or directory

OPENC = 4
Open file for creation

OPENDIR = 3
Open director for reading

OPENR = 5
Open file for reading

OPENW = 6
Open file for writing

PREP = 9
Prepare request

QUERY = 10
Query information request

REDIRECT = 128
Redirect event generated by cmsd

REDLOCAL = 144
Redirect event generated by xrootd

REDSID = 240
Server identification

REDTIME = 0
Window timing mark

RM = 11
Remove a file

RMDIR = 12
Remove a directory

STAT = 13
Stat a file or directory

TRUNC = 14
Truncate a file

xrootdlib.structs.XrdXrootdMon.trace module

class AppId(*appid: bytes*)
Bases: object

appid

classmethod from_buffer(*buffer: bytes*)

size = 16

struct_parser = <Struct object>

class Close(*rtot: int*, *wtot: int*, *dictid: int*)
Bases: object

dictid

classmethod from_buffer(*buffer: bytes*)

rtot

size = 16

struct_parser = <Struct object>

wtot

class Disc(*flags: int*, *buflen: int*, *dictid: int*)
Bases: object

buflen

dictid

flags

classmethod from_buffer(*buffer: bytes*)

size = 16

struct_parser = <Struct object>

class Open(*filesize: int*, *dictid: int*)
Bases: object

dictid

filesize

classmethod from_buffer(*buffer: bytes*)

size = 16

struct_parser = <Struct object>

class Read(*readid: int*, *count: int*, *buflen: int*, *dictid: int*)
Bases: object

buflen

count

dictid

classmethod from_buffer(*buffer: bytes*)

readid

size = 16

struct_parser = <Struct object>

class ReadU(*readid: int*, *count: int*, *buflen: int*, *dictid: int*)
Bases: *xrootdlib.structs.XrdXrootdMon.trace.Read*

class ReadV(*readid: int*, *count: int*, *buflen: int*, *dictid: int*)
Bases: *xrootdlib.structs.XrdXrootdMon.trace.Read*

class ReadWrite(*val: int*, *buflen: int*, *dictid: int*)
Bases: object

buflen

dictid

classmethod from_buffer(*buffer: bytes*)

readlen

size = 16

struct_parser = <Struct object>

val

writelen

TRACE_SIZE = 16
Entries in the I/O and non-I/O event streams are always of fixed size (i.e., 16 characters)

class Window(*sid: int*, *end: int*, *start: int*)
Bases: object

end

classmethod from_buffer(*buffer: bytes*)

sid

size = 16

start

struct_parser = <Struct object>

class XROOTD_MON

Bases: int, enum.Enum

An enumeration.

APPID = 160
Application provided marker

BOUNDP = 2
Entry for a bound path

CLOSE = 192
File has been closed

DISC = 208
Client has disconnected

FORCED = 1
Entry due to forced disconnect

OPEN = 128
File has been opened

READU = 145
Unpacked details for kXR_readv

READV = 144
Details for a kXR_readv request

WINDOW = 224
Window timing mark

1.1.2 Submodules

xrootdlib.utility module

parse_cgi(*cgi_data: AnyStr*) → Dict[AnyStr, AnyStr]

Parse cgi data in the form `&key1=value1&key2=value2` to a mapping

Parameters **cgi_data** – raw CGI data in as unicode or bytes

Returns a mapping from keys to values

Note that this does not perform any implicit type conversions: keys and values have the same type as `cgi_data`. For example, a value of `b'1'` is not converted to the integer `1`.

```
>>> parse_cgi(b'&foo=1')
{b'foo': b'1'}
```

slot_repr(*instance*)

The `xrootdlib` offers building blocks and basic tools to work with the XRootD data access middleware. It is meant to facilitate auxiliary work, such as monitoring, accounting and orchestration.

CHAPTER 2

Contributing and Feedback

The project is hosted on github. If you have issues or suggestion, check the issue tracker: For direct contributions, feel free to fork the development branch and open a pull request.

2.1 Indices and tables

- genindex
- modindex
- search

Documentation built from xrootdlib 0.2.0 at Sep 23, 2021.

Python Module Index

x

Index

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y