XRootD Client Configuration & API Reference

November 9, 2018
Release 4.9.0 and above
Michal Simon (CERN)

Contents

1 Introduction

TODO philosophy:
- connection handling hidden from user
- 100% async + event loop
- threading model: all callbacks in thread-pool + poller thread + timers
- tuned through envvars or XrdCl::DefaultEnv
- OOD API (doxygen)
- Posix API (link xrootd.org)
- New declarative API (ref last section)
- command line tools (ref xrdcp / xrdfs)

. . . .

2 Configuration File

This section describes the XRootD client configuration file. By default XRootD client will use the global config file: */ets/xrootd/client.conf*. However, those settings migh be overwritten by the user specific config file: *~/.xrootd/client.conf* and Environment Variables. For the complete list of configurable parameters please consult the Index of Environment Variables.
XRootD client supports protocol- and endpoint-level plug-ins. By convention a single config file is expected per plug-in, as they are discovered and configured by scanning configuration files. The plug-in manager will search for configuration files in:

- /etc/xrootd/client.plugins.d/,
- ~/.xrootd/client.plugins.d/,
- and at a location pointed to by: XRD_PLUGINCONFDIR

An XRootD client plug-in configuration file should a contain following key-value pairs:

- **url** followed by list of endpoints (by default root protocol is assumed) or protocols
- **lib** followed by a path to the library implementing given plug-in
- **enable** followed by *true* or *false*

For example the following config file defines a plug-in for *host.domain.edu* endpoint (root protocol is being assumed) and *http* protocol:

```
   url = host.domain.edu;http://*
   lib = /usr/lib64/libAwsomePlugIn.so
   enabled = true

```

3 Command line tools (xrdcp/xrdfs)

3.1 xrdcp - copy files

xrdcp [options] source destination

DESCRIPTION

The **xrdcp** utility copies one or more files from one location to another. The data source and destination may be a local or remote file or directory. Additionally, the data source may also reside on multiple servers.

OPTIONS

-C | **--cksum** type [**:**value | print | source]
Obtains the checksum of type (i.e. adler32, crc32, or md5) from the source, computes the checksum at the destination, and verifies that they are the same. If a value is specified, it is used as the source checksum. When print is specified, the checksum at the destination is printed but is not verified.

-d | **--debug** lvl
Debug level: 1 (low), 2 (medium), 3 (high).

-F | **--coerce**
Ignores locking semantics on the destination file. This option may lead to file corruption if not properly used.

-f | **--force**
Re-creates a file if it is already present.

-h | **--help**
Displays usage information.

-H | **--license**
Displays license terms and conditions.

-N | **--nopbar**
Does not display the progress bar.

-P | **--posc**
Requests POSC (persist-on-successful-close) processing to create a new file. Files are automatically deleted should they not be successfully closed.

-D | **--proxy** proxyaddr**:**proxyport [NOT YET IMPLEMENTED]
Use proxyaddr**:**proxyport as a SOCKS4 proxy. Only numerical addresses are supported.

-r | **--recursive**
Recursively copy all files starting at the given source directory.

--server
Runs as if in a server environment. Used only for server-side third party copy support.

-s | **--silent**
Neither produces summary information nor displays the progress bar.

-y | **--sources** num
Uses up to num sources to copy the file.

-S | **--streams** num
Uses num additional parallel streams to do the transfer. The maximum value is 15. The default is 0 (i.e., use only the main stream).

--tpc [**delegate**] **first** | **only**
Copies the file from remote server to remote server using third-party-copy protocol (i.e., data flows from server to server). The source and destination servers must support third party copies. Additional security restrictions may apply and may cause the copy to fail if they cannot be satisfied. Argument '**first**' tries tpc and if it fails, does a normal copy; while '**only**' fails the copy unless tpc succeeds. When '**delegate**' is specified, the copy delegates the command issuer's credentials to the target server which uses those credentials to authenticate with the source server. Delegation is ignored if the target server is not configured to use delegated credentials. Currently, only gsi credentials can be delegated.

-v | **--verbose**
Displays summary output.

-V | **--version**
Displays version information and immediately exits.

-z | **--zip** file
Copy given file from a ZIP archive (same as xrdcl.unzip opaque info).

-X | **--xrate** rate [NOT YET IMPLEMENTED]
Limits the copy speed to the specified rate. The rate may be qualified with the letter **k**, **m**, or **g** to indicate kilo, mega, or giga bytes, respectively. The option only applies when the source or destination is local.

-Z | **--dynamic-src**
File size may change during the copy.

-I | **--infiles** fn
Specifies the file that contains a list of input files.

-p | **--path**
Automatically create remote destination path.

--parallel n
Number of copy jobs to be run simultaneously.

--allow-http
Allow HTTP as source or destination protocol. Requires the XrdClHttp client plugin.

LEGACY OPTIONS

Legacy options are provided for backward compatability. These are now deprecated and should be avoided.

-adler
Equivalent to ”–**cksum adler32:source**”.

-DI pname numberval
Set the internal parameter pname with the numeric value numberval.

-DS pname stringval
Set the internal parameter pname with the string value stringval.

-md5
Equivalent to ”–**cksum md5:source**”.

-np
Equivalent to ”–**nopbar**”.

-OD cgi
Add cgi information cgi to any destination xrootd URL. You should specify the opaque information directly on the destination URL.

-OS cgi
Add cgi information cgi to any source xrootd URL.

-x
Equivalent to ”–**sources 12**”.

OPERANDS

source: a dash (i.e. **-**) indicating stanard in, a local file, a local directory name suffixed by /, or an xrootd URL in the form of:
  xroot:// [user@] host [**:**port] /absolutepath
The absolutepath can be a directory.

destination: a dash (i.e. **-**) indicating stanard out, a local file, a local directory name suffixed by /, or an xrootd URL in the form:
  xroot:// [user@] host [**:**port] /absolutepath
The absolutepath can be a directory.

3.2 xrdfs - xrootd file and directory meta-data utility

....

3.3 Return Codes

- **50** : generic error (e.g. config, internal, data, OS, command line option)
- **51** : socket related error
- **52** : postmaster related error
- **53** : XRootD related error
- **54** : redirection error
- **55** : query response was negative (this is not an error)

4 Environment Variables

This section describes XRootD client environment variables. The following list of environment variables applies to *xrdcp*, *xrdfs* any other application using the libXrdCl library, unless specified otherwise.

4.1 Categories

Limits/Performance:	• XRD_PARALLELEVTLOOP • XRD_REDIRECTLIMIT • XRD_SUBSTREAMSPERCHANNEL • XRD_WORKERTHREADS

Logging:	• XRD_LOGLEVEL • XRD_LOGFILE • XRD_LOGMASK
Metalinks:	• XRD_METALINKPROCESSING • XRD_LOCALMETALINKFILE • XRD_GLFNREDIRECTOR • XRD_MAXMETALINKWAIT
Monitoring:	• XRD_APPNAME • XRD_CLIENTMONITOR • XRD_CLIENTMONITORPARAM
Networking:	• XRD_NETWORKSTACK • XRD_PREFERIPV4
Plug-in:	• XRD_PLUGIN • XRD_PLUGINCONFDIR
Recovery:	• XRD_CONNECTIONRETRY • XRD_OPENRECOVERY • XRD_READRECOVERY • XRD_WRITERECOVERY • XRD_STREAMERRORWINDOW

TCP:	• XRD_NODELAY • XRD_TCPKEEPALIVE • XRD_TCPKEEPALIVEINTERVAL • XRD_TCPKEEPALIVEPROBES • XRD_TCPKEEPALIVETIME
Timeouts:	• XRD_CONNECTIONWINDOW • XRD_REQUESTTIMEOUT • XRD_STREAMTIMEOUT • XRD_DATASERVERTTL • XRD_LOADBALANCERTTL • XRD_TIMEOUTRESOLUTION
XrCl::CopyProcess / xrdcp:	• XRD_CPCHUNKSIZE • XRD_CPINITTIMEOUT • XRD_CPTPCTIMEOUT • XRD_CPPARALLELCHUNKS • XRD_XCPBLOCKSIZE
Others:	• XRD_POLLERPREFERENCE • XRD_RUNFORKHANDLER

4.2 Index of Environment Variables

4.2.1 XRD_APPNAME

Override the application name reported to the server.
Default: disabled

4.2.2 XRD_CLIENTMONITOR

Path to the client monitor library.
Default: disabled

4.2.3 XRD_CLIENTMONITORPARAM

Additional optional parameters that will be passed to the monitoring object on initialization.
Default: disabled

4.2.4 XRD_CONNECTIONWINDOW

A time window for the connection establishment. A connection failure is declared if the connection is not established within the time window. If a connection failure happens earlier then another connection attempt will only be made at the beginning of the next window.
Default: 120 (seconds)

4.2.5 XRD_CONNECTIONRETRY

Number of connection attempts that should be made (number of available connection windows) before declaring a permanent failure.
Default: 5

4.2.6 XRD_CPCHUNKSIZE

Size of a single data chunk handled by xrdcp / XrdCl::CopyProcess.
Default: 16KiB

4.2.7 XRD_CPINITTIMEOUT

Maximum time allowed for the copy process to initialize, ie. open the source and destination files.
Default: 600 (seconds)

4.2.8 XRD_CPPARALLELCHUNKS

Maximum number of asynchronous requests being processed by the xrdcp / XrdCl::CopyProcess command at any given time.
Default: 4

4.2.9 XRD_CPTPCTIMEOUT

Maximum time allowed for a third-party copy operation to finish.
Default: 1800 (seconds)

4.2.10 XRD_DATASERVERTTL

Time period after which an idle connection to a data server should be closed.
Default: 300 (seconds)

4.2.11 XRD_GLFNREDIRECTOR

The redirector will be used as a last resort if the GLFN tag is specified in a Metalink file.
Default: none

4.2.12 XRD_LOADBALANCERTTL

Time period after which an idle connection to a manager or a load balancer should be closed.
Default: 1200 (seconds)

4.2.13 XRD_LOCALMETALINKFILE

Enable/Disable local Metalink file processing (by convention the following URL schema has to be used: *root://localfile//path/filename.meta4*) The *localfile* semantic is now deprecated, use *file://localhost/path/filename.meta4* instead!
Default: 0

4.2.14 XRD_LOGFILE

If set, the diagnostics will be printed to the specified file instead of *stderr*.
Default: disabled

4.2.15 XRD_LOGLEVEL

Determines the amount of diagnostics that should be printed. Valid values are: Dump, Debug, Info, Warning, and Error.
Default: disabled

4.2.16 XRD_LOGMASK

Determines which diagnostics topics should be printed at all levels. It's a "|" separated list of topics. The first element may be "All" in which case all the topics are enabled and the subsequent elements may turn them off, or "None" in which case all the topics are disabled and the subsequent flags may turn them on. If the topic name is prefixed with "^", then it means that the topic should be disabled. If the topic name is not prefixed, then it means that the topic should be enabled.

The log mask may as well be handled for each diagnostic level separately by setting one or more of the following variables: XRD_LOGMASK_ERROR, XRD_LOGMASK_WARNING, XRD_LOGMASK_INFO, XRD_LOGMASK_DEBUG, and XRD_LOGMASK_DUMP.

Available topics: AppMsg, UtilityMsg, FileMsg, PollerMsg, PostMasterMsg, XRootDTransportMsg, TaskMgrMsg, XRootDMsg, FileSystemMsg, AsyncSockMsg
Default: The default for each level is **"All"**, except for the Dump level, where

the default is **"All| ˆPollerMsg"**. This means that, at the Dump level, all the topics but "PollerMsg" are enabled.

4.2.17 XRD_MAXMETALINKWAIT

The maximum time in seconds a client can be stalled by the server if a Metalink redirector is available.
Default: 60 (seconds)

4.2.18 XRD_METALINKPROCESSING

Enable/Disable Metalink processing.
Default: 1

4.2.19 XRD_NETWORKSTACK

The network stack that the client should use to connect to the server. Possible values are:

- **IPAuto** - automatically detect which IP stack to use
- **IPAll** - use IPv6 stack (AF_INET6 sockets) and both IPv6 and IPv4 (mapped to IPv6) addresses
- **IPv6** - use only IPv6 stack and addresses
- **IPv4** - use only IPv4 stack (AF_INET sockets) and addresses
- **IPv4Mapped6** - use IPv6 stack and mapped IPv4 addresses

Default: IPAuto

4.2.20 XRD_NODELAY

Disables the Nagle algorithm if set to 1 (default), enables it if set to 0.
Default: 1

4.2.21 XRD_OPENRECOVERY

Determines if open recovery should be enabled or disabled for mutable (truncate or create) opens.
Default: true

4.2.22 XRD_PARALLELEVTLOOP

The number of event loops.
Default: 1

4.2.23 XRD_PLUGIN

A default client plug-in to be used.
Default: none

4.2.24 XRD_PLUGINCONFDIR

A custom location containing client plug-in config files.
Default: none

4.2.25 XRD_POLLERPREFERENCE

A comma separated list of poller implementations in order of preference.
Default: built-in

4.2.26 XRD_PREFERIPV4

If set the client tries first IPv4 address (turned off by default).
Default: 0

4.2.27 XRD_READRECOVERY

Determines if read recovery should be enabled or disabled.
Default: true

4.2.28 XRD_REDIRECTLIMIT

Maximum number of allowed redirections.
Default: 16

4.2.29 XRD_REQUESTTIMEOUT

Default value for the time after which an error is declared if it was impossible to get a response to a request.
Default: 1800 (seconds)

4.2.30 XRD_RUNFORKHANDLER

Determines whether the fork handlers should be enabled, making the API fork safe.
Default: 0

4.2.31 XRD_STREAMERRORWINDOW

Time after which the permanent failure flags are cleared out and a new connection may be attempted if needed.
Default: 1800

4.2.32 XRD_STREAMTIMEOUT

Default value for the time after which a connection error is declared (and a recovery attempted) if there are unfulfilled requests and there is no socket activity or a registered wait timeout.
Default: 60 (seconds)

4.2.33 XRD_SUBSTREAMSPERCHANNEL

Number of streams per session.
Default: 1

4.2.34 XRD_TCPKEEPALIVE

Enable/Disable the TCP keep alive functionality.
Default: 0

4.2.35 XRD_TCPKEEPALIVEINTERVAL

Interval between subsequent keepalive probes (Linux only).
Default: 75

4.2.36 XRD_TCPKEEPALIVEPROBES

Number of unacknowledged probes before considering the connection dead (Linux only).
Default: 9

4.2.37 XRD_TCPKEEPALIVETIME

Time between last data packet sent and the first keepalive probe (Linux only).
Default: 7200

4.2.38 XRD_TIMEOUTRESOLUTION

Resolution for the timeout events. Ie. timeout events will be processed only every XRD_TIMEOUTRESOLUTION seconds.
Default: 15 (seconds)

4.2.39 XRD_WORKERTHREADS

Number of threads processing user callbacks.
Default: 3

4.2.40 XRD_WRITERECOVERY

Determines if write recovery should be enabled or disabled.
Default: true

4.2.41 XRD_XCPBLOCKSIZE

Maximu size of a data block assigned to a single source in case of an extreme copy transfer.
Default: 128MiB

4.3 Timeouts Explained

4.3.1 Connection Window and Connection Retry

The *ConnectionWindow* parameter is applied during client-server connection and controls two aspects of this process:

- First of all, it controls the length of time allowed to establish an XRootD connection (physical connection, XRootD hand-shake, and authentication if required). It is important to note that ***Connection Window* is applied per physical address**. This means that if a connection fails before the end of current *ConnectionWindow* and another physical address is available it will be tried immediately.
- Secondly (if there are no more available physical addresses), it defines the length of time that must elapse after a connection failure before the connection can be retried. More precisely, the client has to wait until the end of the current *ConnectionWindow* before attempting another connection. The number of retries that might be attempted is controlled by *ConnectionRetry* environment variable.

Let us illustrate all this with following example. Suppose XRootD client wants to connect to a server with three physical IP address (2x IPv6 and 1x IPv4). For the sake of argument let us suppose it will fail after 60s during the hand-shake procedure, while connecting to the 1^{st} IPv6 address. What will happen next? Since there are two more addresses available, the client will immediately proceed to the next one. Now let us suppose that the cumulative time spent on establishing the physical connection and on carrying out the hand-shake exceeded 120s (nominal value of *ConnectionWindow*). In this case the second connection attempt will be timed out, and XRootD client will proceed to the 3^{rd} IP address. Again, let us suppose that similarly as in case of the 1^{st} IP address the connection failed after 60s. Since there are no more address to try, the client will have to wait until the end of the current *ConnectionWindow* (that is for another 60s) before the connection procedure can be restarted. Now how all this relates to the *ConnectioRetry*? The nominal value of *ConnectioRetry* is 5, which means we can retry the whole procedure four more times (Note: ***ConnectionRetry* is not applied per single physical connection but rather to the whole connection procedure**).

4.3.2 Stream Timeout

The *StreamTimeout* parameter is applied during every request/response exchange after the client and the server established a connection. It defines the

maximum length of time that may elapse between the moment when the client has sent a request and the moment when the client has received a response for the request in question. If the time spent waiting for response from the server exceeds the *StreamTimeout* an error is declared (and the client will disconnect form the server).
There are two exceptions to the above stated rule:

- The server may force the client to reissue the request by sending *kXR_-wait* response. In this case *StreamTimeout* does not apply to the original request anymore.
- The server may explicitly instruct the client to not apply the *StreamTime-out* to given request by sending *kXR_waitresp*.

4.3.3 Stream Error Window

The *StreamErrorWindow* controls the length of time that needs to elapse after a fatal error before the client may attempt to reconnect to the server. A fatal error is declared eg. if the host name cannot be resolved, a low level Posix system call fails (eg. *connect*/*fcntl*/*epool*), or client runs out of connection retries.

4.3.4 RequestTimeout

The *RequestTimeout* parameter is applied to a logical XRootD operation (eg. opening a file, listing directory, etc.) as a whole. It is the maximum length of time that may elapse from the moment an operation has been issued using XRootD client API until it has been resolved (no matter how many underlying requests it will trigger). If the *RequestTimeout* is exceeded an error is declared and the operation is resolved as failed.
Note: The value of this parameter might be overwritten directly by the user of XRootD client API by setting the timeout argument.

4.3.5 Time To Live

A Time To Live (TTL) timeout controls the lifetime of an idle physical connection. If for the given communication channel the time length elapsed from last exchange of request/response between the client and server exceeds the TTL timeout the given connection will be terminated. There are two types of TTL timeouts in XRootD client:

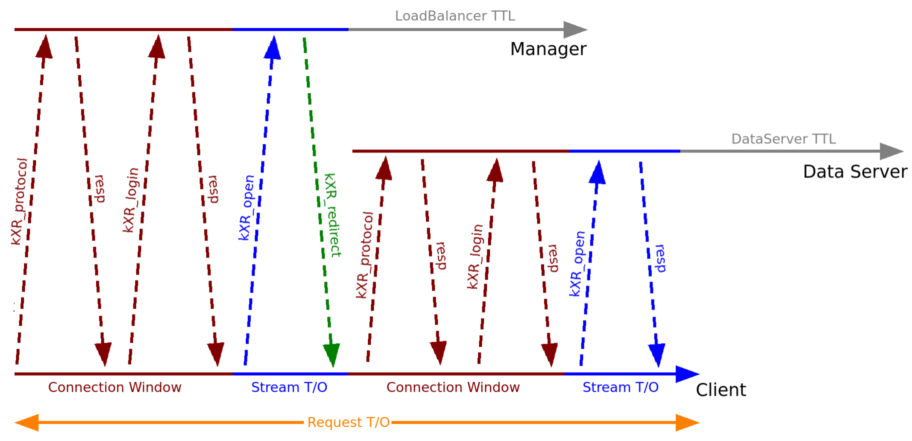
- *DataServerTTL*: a TTL timeout that is applied to Data Servers
- *LoadBalancerTTL*: a TTL timeout that is applied to Managers

4.3.6 How does it all come together?

Let us now consider an example in order to illustrate how all those timeouts play along (**for clarity please consult the diagram below**). Suppose that an *XrdCl::File::Open(. . .)* operation is being called and that there is no open connection between the client and the server. The client will have to establish the XRootD connection first (subject to ***ConnectionWindow***):

- open physical connection
- carry out hand-shake procedure (*kXR_protocol*, *kXR_login*, etc.)

Subsequently, the client will issue a *kXR_open* request (subject to ***StreamTimeout***). Let us suppose that the server will respond with a *kXR_redirect* redirecting the client to a data server. In this case, the client will have to open another XRootD connection (again, subject to ***ConnectionWindow***) and then send an open request (again, subject to ***StreamTimeout***). Finally, once the server responds, the open operation will be resolved. The whole process described in the scenario above is subject to ***RequestTimeout***.
Once the connections to the manager and data server become idle they will be subject to respective ***TTL timeouts***.



4.3.7 xrdcp / XrdCl::CopyProcess Third-Party-Copy timeouts

The *CPInitTimeout* parameter is applied during initialization of a Third-Party-Copy (TPC) transfer. It defines the maximum length of time that may elapse until TPC transfer has been initialized (ie. *open* destination, *open* source, issue *sync*, for more details please consult the TPC Protocol Reference).
The *CPTPCTimeout* parameter defines the maximum length of time that may elapse between the moment when the actual transfer has been started and the moment when it is finished (ie. it is applied to the second *sync*, for more details please consult the TPC Protocol Reference).

5 Client Declarative API

This section describes XRootD client declarative API introduced in 4.9.0. For the standard *XrdCl::File* and *XrdCl::FileSystem*API please consult our Doxygen documentation.

5.1 List of Operations

open, close, read, write . . . .
+ arguments and response type

5.2 Special Operations

parallel, later optional

5.3 Operation Handlers

ResponseHandler, function/functor/lambda, std::future, std::packaged_task

5.4 Utilities

Fwd class

5.5 Pipelining Semantics

examples on how to create and run pipelines