

# Programming with cURLpp

Jean-Philippe Barrette-LaPierre

July 15, 2018

## 1 About this Document

This document attempts to describe the general principles and some basic approaches to consider when programming with cURLpp. Don't forget that cURLpp is a C++ wrapper of libcurl, so cURLpp needs libcurl to be installed already.

This document will refer to 'the user' as the person writing the source code that uses cURLpp. That would probably be you or someone in a similar position. What will be generally referred to as 'the program' will be the collective source code that you write and that is using cURLpp for transfers. The program is outside cURLpp and cURLpp is outside of the program.

To get more details on all options and functions described herein, please refer to their respective man pages. You should always have in mind that this is a C++ wrapper of libcurl. It would be unproductive to duplicate libcurl's documentation here, so this document will show you how to interact with cURLpp, but you should read the libcurl programming tutorial, which this document is strongly inspired from, and the libcurl man pages.

## 2 Building

There are many different ways to build C++ programs. This chapter will assume a unix-style build process. If you use a different build system, you can still read this to get general information that may apply to your environment as well. Note that cURLpp need libcurl to be already installed.

## 2.1 Compiling the Program

Your compiler needs to know where cURLpp's and libcurl's headers are located. Therefore you must set your compiler's include path to point to the directory where you installed them. The 'curlpp-config'<sup>1</sup> tool can be used to get this information:

```
# curlpp-config --cflags
```

If pkg-config is installed, you can use it this way:

```
# pkg-config --cflags curlpp
```

But, if you're using `autoconf` for your project you can use `pkg-config` macros. See `pkg-config` man pages for more details.

## 2.2 Linking the Program with cURLpp

When having compiled the program, you need to link your object files to create a single executable. For that to succeed, you need to link with cURLpp and possibly also with other libraries that cURLpp itself depends on (such as libcurl). This may include OpenSSL libraries and even some standard OS libraries may be needed on the command line. To figure out which flags to use, the 'curlpp-config' tool comes to the rescue once again:

```
# curlpp-config --libs
```

Again, if pkg-config is installed, you can use it this way:

```
# pkg-config --libs curlpp
```

---

<sup>1</sup>The curlpp-config tool, which wraps all functions of curl-config, is generated at build-time (on unix-like systems) and should be installed with the 'make install' or similar instruction that installs the library, header files, man pages etc.

## 2.3 SSL or Not

cURLpp, like libcurl, can be built and customized in many ways. One of the things that varies between different libraries and builds is the support for SSL-based transfers, like HTTPS and FTPS. If OpenSSL was detected properly by libcurl at build-time, cURLpp will be built with SSL support. To figure out if an installed cURLpp has been built with SSL support enabled, use 'curlpp-config' like this:

```
# curlpp-config --feature
```

If SSL is supported, the keyword 'SSL' will be written to stdout, possibly together with a few other features that can be on and off on different cURLpps.

## 2.4 Portable Code in a Portable World

The people behind libcurl have put a considerable effort to make libcurl work on a large number of different operating systems and environments.

You program cURLpp the same way on all platforms that cURLpp runs on. There are only very few minor considerations that differ. If you make sure just to write your code portably enough, you may very well create yourself a very portable program. cURLpp shouldn't stop you from that.

## 3 Global Preparation

The program must initialize some of the cURLpp functionality globally. That means it should be done exactly once, no matter how many times you intend to use the library. Once for your program's entire lifetime. This is done using

```
cURLpp::initialize( long flags = cURLpp::CURL_GLOBAL_ALL )
```

and it takes one parameter which is a bit pattern that tells cURLpp what to initialize. Check the man page of `curl_global_init` for more details on flags.

When the program no longer uses cURLpp, it should call `cURLpp::terminate()`, which is the opposite of the init call. It will then do the operations needed to cleanup the resources that the `cURLpp::initialize()` call initialized. Repeated calls to `cURLpp::initialize()` and `cURLpp::terminate()` must not be made. They must only be called once each.

## 4 Handle the Easy cURLpp

To use the easy interface, you must first create yourself an easy handle. You need one handle for each easy session you want to perform. Basically, you should use one handle for every thread you plan to use for transferring. You must never share the same handle in multiple threads.

Get an easy handle with

```
cURLpp::Easy easyhandle;
```

This creates an easy handle. Using that you proceed to the next step: setting up your preferred actions. A handle is just a logic entity for the upcoming transfer or series of transfers. You can use it to do HTTP or FTP requests.

You set properties and options for this handle using `cURLpp::Options`, or `cURLpp::OptionList` classes; we will discuss `cURLpp::OptionList` later. They control how the subsequent transfer or transfers will be made. Options remain set in the handle until set again to something different. Alas, multiple requests using the same handle will use the same options.

Many of the informationals you set in cURLpp are C++ standard library strings. Keep in mind that when you set strings with member functions, cURLpp will copy the data. It will not merely point to the data. You don't need to make sure that the data remains available for cURLpp.

One of the most basic properties to set in the handle is the URL. You set your preferred URL to transfer with a `void cURLpp::Options::Url(const char *link)` option class, in a manner similar to:

```
easyhandle.setOpt(cURLpp::Options::Url("http://example.com/"));
```

There are of course many more options you can set, and we'll get back to a few of them later. Let's instead continue to the actual transfer:

```
easyhandle.perform();
```

The `cURLpp::Easy::perform()` will connect to the remote site, do the necessary commands and receive the transfer. Whenever it receives data, it calls the trait function we previously set. The function may get one byte at a time, or it may get many kilobytes at once. cURLpp delivers as much as possible as often as possible. Your trait function should return the number

of bytes it "took care of". If that is not the exact same amount of bytes that was passed to it, `cURLpp` will abort the operation and throw an exception.

When the transfer is complete, the function throws a `cURLpp::Exception` if it doesn't succeed in its mission. the `const char *cURLpp::Exception::what()` will return the human readable reason of failure.

## 5 Wrapping libcurl

As previously said, `cURLpp` is just a C++libcurl wrapper. It wouldn't be a good idea to reproduce here, in this project, all the libcurl documentation. This means that when you will be programming with `cURLpp`, you will refer more to libcurl's documentation than to `cURLpp`'s. We will explain here how `cURLpp` wraps libcurl, so you will be able to use it, without constantly referring to its manual.

First, you must know that there are two main things that constitute `cURLpp`: There are its options value setting/retrieving behavior and its utilities that help you to use libcurl's options more easily.

### 5.1 Option setting/retrieving

The main feature of `cURLpp` is that you can retrieve options previously set on handles. `cURLpp` gives you the opportunity to retrieve options values that were previously set on a specific handle and then use them again for other handles. But first, let's show you how to set an option on a handle, in comparison to libcurl's way of setting an option.

libcurl sets options on handles with this function:

```
curl_easy_setopt(CURL *handle, CURLOPToption option, parameter)
```

Here's a part of the documentation taken from the man pages:

`curl_easy_setopt()` is used to tell libcurl how to behave. By using the appropriate options to `curl_easy_setopt()`, you can change libcurl's behavior. All options are set with the option followed by a parameter. That parameter can be a long, a function pointer or an object pointer, all depending on what the specific option expects.

Lets code a simple example:

```
CURL *handle = curl_easy_init();
if(handle == NULL) {
    //something went wrong.
}

CURLcode code = curl_easy_setopt(handle,
    CURLOPT_URL, 'http://www.example.com');
if(code != CURLE_OK) {
    //something went wrong
}
```

The code below does the same thing but with cURLpp:

```
cURLpp::Easy handle;
handle.setOpt(cURLpp::Options::Url('http://www.example.com'));
```

If a problem occur, cURLpp will throw an exception, for more detail on this subject, see the next section named *Exception issues*. As you see, the equivalent of the `curl_easy_setopt` function is the `cURLpp::Easy::setOpt` member function.

### 5.1.1 cURLpp::Options

The question that you might ask you at this moment is: “what exactly is the `cURLpp::Options::Url` class mentioned in the previous example?” In fact, this class is used to store values of options, but also to retrieve them, as shown below:

```
cURLpp::Easy handle;
handle.setOpt(cURLpp::Options::Url('http://www.example.com'));

cURLpp::Options::Url myUrl;
handle.getOpt(myUrl);
std::cout << myUrl.getValue() << std::endl;
```

This piece of code should print the URL on the standard output. Here's the code of the `examples/example01.cpp` file.

```

#include <string>
#include <iostream>

#include <curlpp/cURLpp.hpp>
#include <curlpp/Options.hpp>

#define MY_PORT 8080

/**
 * This example is made to show you how you can use the Options.
 */
int main()
{
    try
    {
        // Creation of the URL option.
        cURLpp::Options::Url myUrl(
            std::string("http://example.com"));

        // Copy construct from the other URL.
        cURLpp::Options::Url myUrl2(myUrl);

        // Creation of the port option.
        cURLpp::Options::Port myPort(MY_PORT);

        // Creation of the request.
        cURLpp::Easy myRequest;

        // Creation of an option that contain a copy
        // of the URL option.
        cURLpp::OptionBase *mytest = myUrl.clone();
        myRequest.setOpt(*mytest);

        // You can reuse the base option for other type of option
        // and set the option to the request. but first, don't forget
        // to delete the previous memory. You can delete it since the
        // option is internally duplicated for the request.
    }
}

```

```

delete mytest;
mytest = myPort.clone();
myRequest.setOpt(*mytest);
delete mytest;

// You can clone an option directly to the same type of
// option.
cURLpp::Options::Url *myUrl3 = myUrl.clone();
myRequest.setOpt(myUrl3);
// Now myUrl3 is owned by the request we will NOT use
// it anymore.

// You don't need to declare an option if you just want
// to use it once.
myRequest.setOpt(cURLpp::Options::Url("example.com"));

// Note that the previous line wasn't really efficient
// because we create the option, this option is duplicated
// for the request and then the option destructor is called.
// You can use this instead:
myRequest.setOpt(new cURLpp::Options::Url("example.com"));
// Note that with this the request will use directly this
// instance we just created. Be aware that if you pass an
// Option pointer to the setOpt function, it will consider
// the instance has its own instance. The Option instance
// will be deleted when the request will be deleted, so
// don't use the instance further in your code.

// Doing the previous line is efficient as this:
myRequest.setOpt(myUrl.clone());

// You can retrieve the value of a specific option.
std::cout << myUrl2.getValue() << std::endl;

```



```

        // You can see what are the options set for a request
        // with this function. This function will print the option
        // to the stdout.
        myRequest.print();

        // Perform the transaction with the options set.
        myRequest.perform();
    }
    catch( cURLpp::RuntimeError &e )
    {
        std::cout << e.what() << std::endl;
    }
    catch( cURLpp::LogicError &e )
    {
        std::cout << e.what() << std::endl;
    }

    return 0;
}

```

## 5.2 cURLpp::Option types

This section will explain the use of the types that we use for some options that differ from types that libcurl uses for the same option.

### 5.2.1 The bool type

A true value is binded to a non-zero long value, a false value is binded to a zero long value.

### 5.2.2 The std::string type

The `std::string::c_str()` function is passed to libcurl.

### 5.2.3 The std::list template of std::string type

This type is used when libcurl's option need a `curl_slist`. Instead of using this homemade struct, cURLpp allow you to use a known type, the

`std::list`. `cURLpp` this transform internally the `std::list` to a `curl_slist`.

## 5.3 cURLpp::Options

This section just list how each libcurl's option is binded by `cURLpp`. Some options might not be there, if it's the case and you want to use them, see the "how to enhance `cURLpp`" section.

### 5.3.1 Behavior options

```
typedef cURLpp::OptionTrait< bool,  
    CURLOPT_VERBOSE > Verbose;  
typedef cURLpp::OptionTrait< bool,  
    CURLOPT_HEADER > Header;  
typedef cURLpp::OptionTrait< bool,  
    CURLOPT_NOSIGNAL > NoSignal;  
typedef cURLpp::OptionTrait< bool,  
    CURLOPT_NOPROGRESS > NoProgress;
```

### 5.3.2 Callback options

```
typedef cURLpp::OptionTrait< cURL::curl_write_callback,  
    CURLOPT_WRITEFUNCTION > WriteFunction;  
typedef cURLpp::OptionTrait< void *,  
    CURLOPT_WRITEDATA > WriteData;  
typedef cURLpp::OptionTrait< cURL::curl_read_callback,  
    CURLOPT_READFUNCTION > ReadFunction;  
typedef cURLpp::OptionTrait< void *,  
    CURLOPT_READDATA > ReadData;  
typedef cURLpp::OptionTrait< cURL::curl_progress_callback,  
    CURLOPT_PROGRESSFUNCTION > ProgressFunction;  
typedef cURLpp::OptionTrait< void *,  
    CURLOPT_PROGRESSDATA > ProgressData;  
typedef cURLpp::OptionTrait< cURL::curl_write_callback,  
    CURLOPT_HEADERFUNCTION > HeaderFunction;  
typedef cURLpp::OptionTrait< void *,  
    CURLOPT_HEADERDATA > HeaderData;  
typedef cURLpp::OptionTrait< cURL::curl_debug_callback,
```

```

        CURLOPT_DEBUGFUNCTION > DebugFunction;
typedef curlpp::OptionTrait< void *,
        CURLOPT_DEBUGDATA > DebugData;
#ifdef CURLOPT_SSL_CTX_FUNCTION
typedef curlpp::OptionTrait< curl::curl_ssl_ctx_callback,
        CURLOPT_SSL_CTX_FUNCTION > SslCtxFunction;
typedef curlpp::OptionTrait< void *,
        curl::CURLOPT_SSL_CTX_DATA > SslCtxData;
#endif

```

### 5.3.3 Error options

```

typedef curlpp::OptionTrait< char *,
        curl::CURLOPT_ERRORBUFFER > ErrorBuffer;
typedef curlpp::OptionTrait< FILE *,
        curl::CURLOPT_STDERR > StdErr;
typedef curlpp::OptionTrait< bool,
        curl::CURLOPT_FAILONERROR > FailOnError;

```

### 5.3.4 Network options

```

typedef curlpp::OptionTrait< std::string,
        curl::CURLOPT_URL > Url;
typedef curlpp::OptionTrait< std::string,
        curl::CURLOPT_PROXY > Proxy;
typedef curlpp::OptionTrait< long,
        curl::CURLOPT_PROXYPORT > ProxyPort;
typedef curlpp::OptionTrait< curl_proxytype,
        curl::CURLOPT_PROXYTYPE > ProxyType;
typedef curlpp::OptionTrait< bool,
        curl::CURLOPT_HTTPPROXYTUNNEL > HttpProxyTunnel;
typedef curlpp::OptionTrait< std::string,
        curl::CURLOPT_INTERFACE > Interface;
typedef curlpp::OptionTrait< long,
        curl::CURLOPT_DNS_CACHE_TIMEOUT > DnsCacheTimeout;
typedef curlpp::OptionTrait< bool,
        curl::CURLOPT_DNS_USE_GLOBAL_CACHE > DnsUseGlobalCache;
typedef curlpp::OptionTrait< long,

```

```

        cURL::CURLOPT_BUFFERSIZE > BufferSize;
    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_PORT > Port;
#ifdef cURL::CURLOPT_TCP_NODELAY
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_TCP_NODELAY > TcpNoDelay;
#endif

```

### 5.3.5 Names and passwords options

```

    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_NETRC > Netrc;
#ifdef cURL::CURLOPT_NETRC_FILE
    typedef cURLpp::OptionTrait< std::string,
        cURL::CURLOPT_NETRC_FILE > NetrcFile;
#endif
    typedef cURLpp::OptionTrait< std::string,
        cURL::CURLOPT_USERPWD > UserPwd;
    typedef cURLpp::OptionTrait< std::string,
        cURL::CURLOPT_PROXYUSERPWD > ProxyUserPwd;
    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_HTTPAUTH > HttpAuth;
    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_PROXYAUTH > ProxyAuth;

```

### 5.3.6 HTTP options

```

    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_AUTOREFERER > AutoReferer;
    typedef cURLpp::OptionTrait< std::string,
        cURL::CURLOPT_ENCODING > Encoding;
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_FOLLOWLOCATION > FollowLocation;
#ifdef cURL::CURLOPT_UNRESTRICTED_AUTH
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_UNRESTRICTED_AUTH > UnrestrictedAuth;
#endif
    typedef cURLpp::OptionTrait< long,

```

```

        cURL::CURLOPT_MAXREDIRS > MaxRedirs;
#ifdef cURL::CURLOPT_UPLOAD
        typedef cURLpp::OptionTrait< bool,
            cURL::CURLOPT_PUT > Put;
#else
        typedef cURLpp::OptionTrait< bool,
            cURL::CURLOPT_UPLOAD > Put;
#endif
        typedef cURLpp::OptionTrait< bool,
            cURL::CURLOPT_POST > Post;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_POSTFIELDS > PostFields;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_POSTFIELDSIZE > PostFieldSize;
#ifdef cURL::CURLOPT_POSTFIELDSIZE_LARGE
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_POSTFIELDSIZE_LARGE > PostFieldSizeLarge;
#endif
        typedef cURLpp::OptionTrait< struct cURLpp::cURL::HttpPost *,
            cURL::CURLOPT_HTTPPOST > HttpPost;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_REFERER > Referer;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_USERAGENT > UserAgent;
        typedef cURLpp::OptionTrait< std::list< std::string >,
            cURL::CURLOPT_HTTPHEADER > HttpHeaders;
        typedef cURLpp::OptionTrait< std::list< std::string >,
            cURL::CURLOPT_HTTP200ALIASES > Http200Aliases;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_COOKIE > Cookie;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_COOKIEFILE > CookieFile;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_COOKIEJAR > CookieJar;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_COOKIESESSION > CookieSession;
        typedef cURLpp::OptionTrait< std::string,
            cURL::CURLOPT_HTTPGET > HttpGet;

```

```
typedef cURLpp::OptionTrait< long,
    cURL::CURLOPT_HTTP_VERSION > HttpVersion;
```

### 5.3.7 FTP options

```
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_FTPPORT > FtpPort;
typedef cURLpp::OptionTrait< long,
    cURL::CURLOPT_QUOTE > Quote;
typedef cURLpp::OptionTrait< long,
    cURL::CURLOPT_POSTQUOTE > PostQuote;
typedef cURLpp::OptionTrait< long,
    cURL::CURLOPT_PREQUOTE > PreQuote;
typedef cURLpp::OptionTrait< bool,
    cURL::CURLOPT_FTPLISTONLY > FtpListOnly;
typedef cURLpp::OptionTrait< bool,
    cURL::CURLOPT_FTPAPPEND > FtpAppend;
typedef cURLpp::OptionTrait< bool,
    cURL::CURLOPT_FTP_USE_EPSV > FtpUseEpsv;
#ifdef cURL::CURLOPT_FTP_CREATE_MISSING_DIRS
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_FTP_CREATE_MISSING_DIRS > FtpCreateMissingDirs;
#endif
#ifdef cURL::CURLOPT_FTP_RESPONSE_TIMEOUT
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_FTP_RESPONSE_TIMEOUT > FtpResponseTimeout;
#endif
#ifdef cURL::CURLOPT_FTP_SSL
    typedef cURLpp::OptionTrait< cURLpp::curl_ftpssl,
        cURL::CURLOPT_FTP_SSL > FtpSsl;
#endif
#ifdef cURL::CURLOPT_FTP_AUTH
    typedef cURLpp::OptionTrait< cURLpp::curl_ftpauth,
        cURL::CURLOPT_FTP_AUTH > FtpAuth;
#endif
```

### 5.3.8 Protocol options

```
typedef cURLpp::OptionTrait< bool,  
    cURL::CURLOPT_TRANSFERTEXT > TransferText;  
typedef cURLpp::OptionTrait< bool,  
    cURL::CURLOPT_CRLF > Crlf;  
typedef cURLpp::OptionTrait< std::string,  
    cURL::CURLOPT_RANGE > Range;  
typedef cURLpp::OptionTrait< long,  
    cURL::CURLOPT_RESUME_FROM > ResumeFrom;  
#ifdef cURL::CURLOPT_RESUME_FROM_LARGE  
    typedef cURLpp::OptionTrait< curl_off_t,  
        cURL::CURLOPT_RESUME_FROM_LARGE > ResumeFromLarge;  
#endif  
    typedef cURLpp::OptionTrait< std::string,  
        cURL::CURLOPT_CUSTOMREQUEST > CustomRequest;  
    typedef cURLpp::OptionTrait< bool,  
        cURL::CURLOPT_FILETIME > FileTime;  
    typedef cURLpp::OptionTrait< bool,  
        cURL::CURLOPT_NOBODY > NoBody;  
    typedef cURLpp::OptionTrait< long,  
        cURL::CURLOPT_INFILESIZE > InfileSize;  
#ifdef cURL::CURLOPT_INFILESIZE_LARGE  
    typedef cURLpp::OptionTrait< cURL::curl_off_t,  
        cURL::CURLOPT_INFILESIZE_LARGE > InfileSizeLarge;  
#endif  
#ifdef cURL::CURLOPT_UPLOAD  
    typedef cURLpp::OptionTrait< bool,  
        cURL::CURLOPT_UPLOAD > Upload;  
#else  
    typedef cURLpp::OptionTrait< bool,  
        cURL::CURLOPT_PUT > Upload;  
#endif  
#ifdef cURL::CURLOPT_MAXFILESIZE  
    typedef cURLpp::OptionTrait< long,  
        cURL::CURLOPT_MAXFILESIZE > MaxFileSize;  
#endif  
#ifdef cURL::CURLOPT_MAXFILESIZE_LARGE
```

```

        typedef cURLpp::OptionTrait< cURL::curl_off_t,
            cURL::CURLOPT_MAXFILESIZE_LARGE > MaxFileSizeLarge;
    #endif
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_TIMECONDITION > TimeCondition;
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_TIMECONDITION > TimeValue;

```

### 5.3.9 Connection options

```

    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_TIMEOUT > Timeout;
    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_LOW_SPEED_LIMIT > LowSpeedLimit;
    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_LOW_SPEED_TIME > LowSpeedTime;
    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_MAXCONNECTS > MaxConnects;
    typedef cURLpp::OptionTrait< cURL::curl_closepolicy,
        cURL::CURLOPT_CLOSEPOLICY > ClosePolicy;
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_FRESH_CONNECT > FreshConnect;
    typedef cURLpp::OptionTrait< bool,
        cURL::CURLOPT_FORBID_REUSE > ForbidReuse;
    typedef cURLpp::OptionTrait< long,
        cURL::CURLOPT_CONNECTTIMEOUT > ConnectTimeout;
    #ifdef cURL::CURLOPT_IPRESOLVE
        typedef cURLpp::OptionTrait< long,
            cURL::CURLOPT_IPRESOLVE > IpResolve;
    #endif

```

### 5.3.10 SSL and security options

```

    typedef cURLpp::OptionTrait< std::string,
        cURL::CURLOPT_SSLCERT > SslCert;
    typedef cURLpp::OptionTrait< std::string,
        cURL::CURLOPT_SSLCERTTYPE > SslCertType;
    typedef cURLpp::OptionTrait< std::string,

```



```

    cURL::CURLOPT_SSLCERTPASSWD > SslCertPasswd;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_SSLKEY > SslKey;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_SSLKEYTYPE > SslKeyType;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_SSLKEYPASSWD > SslKeyPasswd;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_SSLENGINE > SslEngine;
typedef cURLpp::OptionTrait< long,
    cURL::CURLOPT_SSLVERSION > SslVersion;
typedef cURLpp::OptionTrait< bool,
    cURL::CURLOPT_SSL_VERIFYPEER > SslVerifyPeer;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_CAINFO > CaInfo;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_CAPATH > CaPath;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_RANDOM_FILE > RandomFile;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_EGDSOCKET > EgdSocket;
typedef cURLpp::OptionTrait< long,
    cURL::CURLOPT_SSL_VERIFYHOST > SslVerifyHost;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_SSL_CIPHER_LIST > SslCipherList;
typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_KRB4LEVEL > Krb4Level;

```

### 5.3.11 Others options

```

typedef cURLpp::OptionTrait< std::string,
    cURL::CURLOPT_KRB4LEVEL > Krb4Level;

```

## 6 How the enhance cURLpp

Need to be written.

## 7 Exceptions issues

As previously said, cURLpp (libcurl in fact) offer the possibility to reimplement the data writing/reading functions. Those functions called from within libcurl might raise exceptions. Raising an exception in C code might cause problems. cURLpp protect you from doing so<sup>2</sup>. All exceptions are caught by cURLpp before they could cause damage to libcurl. If you want to raise an exception within traits, you need to do this:

```
cURLpp::raiseException(MyException('Exception Raised'));
```

Then, the `cURLpp::Easy::perform()` will raise your exception at the end of the process. If an exception is raised but not by this mechanism, a `cURLpp::UnknownExceptionError` will be raised.

---

<sup>2</sup>This feature will be added only in the 0.6.0 version