Mobile Location Protocol

User Guide

Petr VLFčIL

Mobile Location Protocol: User Guide

Petr VLFčIL

Revision History

Revision 1.1	25.11.2009	PV

Version for first auto tools release.

Table of Contents

Chapter 1. Document Scope

Goal of this document is to demonstrate possibilities of Mobile Location Protocol library and describe its usage.

Chapter 2. Preliminaries

2.1. Prerequisites

Mobile Location Protocol (MLP) library is written in C++. Provided source files and makefiles are prepared for compilation with GNU build tools (http://gcc.gnu.org/). Compilation was tested on Linux and Solaris. MLP library requires following additional software to be installed:

Xerces-c++, cURL	MLP is XML based protocol. Xerces SAX parser is used to process XML. MLP library was tested with Xerces-c 3.0.1. (http://xerces.apache.org/xerces-c/). cURL is needed for example client. It is used to send HTTP POST messages. It is needed only for curl-client example. MLP library was tested with curl 7.19.4. (http://curl.haxx.se/libcurl/).

2.2. Downloading

To obtain the actual source codes of the MLP library use subversion. Download the sources from the read-only repository at:

```
svn co https://libmlp-cpp.svn.sourceforge.net/svnroot/libmlp-cpp libmlp-cpp
```

Or download source file package at SourceForge web page of the project:

http://sourceforge.net/projects/libmlp-cpp/

2.3. Directory Architecture

When you download a package you have to unpack it with `tar -xzf libmlp-cpp-1.1.0.tar.gz` . After unpacking the archive, directory will look like this:

```
libmlp-cpp-1.1.0
    |
    +----config/
    +----doc/
    +----examples/
    +----lib/
    +----libmlp-cpp/
            |
            +----dtd/
    +----m4/
    +----test/
    |
    *----aclocal.m4
    *----AUTHORS
    *----bootstrap.sh
    *----ChangeLog
    *----config.hpp.in
    *----configure
    *----configure.ac
    *----COPYING
    *----doxyfile
    *----INSTALL
    *----libmlp-cpp-1.0.pc.in
```

```
    *----Makefile.am
    *----Makefile.in
    *----NEWS
    *----README
    *----TODO
```

Description follows:

doc/	Contains all documentation
examples/	Contains examples that demonstrate usage of MLP Library.
libmlp-cpp/	Contains source files
dtd/	Contain DTD files for MLP downloaded from: http://www.openmobilealliance.org/Technical/DTD.aspx.
AUTHORS	Contains information about authors.
COPYING	Contains BSD license and copyright.
README	Contains brief info about library and its installation.

2.4. Compilation

This project uses GNU auto tools tool-chain to build. If you are building form tar.gz package you should type *./configure*, *make*, *make install*. If you are building from SVN and you changed some configuration files, run *./boostrap.sh* to regenerate *configure* script.

Few configuration options are available:

--with-doxygen	Set path where doxygen is installed. Doxygen is used for documentation from source-code generation.
--enable-warnings	Turn on lots of compiler warnings.
--enable-examples	Compile examples together with library. If you enable this, you will need cURL library for example-client example.
--prefix	Installation directory. If not used, prefix set to default directory */usr/local*

Compilation produces libtool library that can be linked to your programs. Result of whole process is `libmlp-cpp.la` libtool library placed in `libmlp-cpp-1.1.0/libmlp-cpp`. Library provide pkg-config file `libmlp-cpp.pc` that stores information about includes and linker flags needed by other projects that want to use MLP library.

```
libmlp-cpp-1.1.0
          |
          +----libmlp-cpp/
                 |
                 *----libmlp-cpp.la
```

2.5. Installation

To install the library use *make install* command. Default directory to install is /usr/local. It can be changed by --prefix=[your path] parameter of configuration script. Headers are installed into $(pre-

fix)/include/libmlp-cpp. Library is installed into $(prefix)/lib. MLP DTDs are installed into $(prefix)/share/libmlp-cpp/dtd. Pkg-config file is installed into $(prefix)/lib/pkgconfig. Pkg-config file provides easy way to include library into other projects. Example usage of libmlp-cpp.pc file is described in section Compiling the Example. Directory structure after installation with *--prefix* set to *[your path]*:

```
--prefix=[your path]
        |
        +----include/
        |       |
        |       +----libmlp-cpp/
        |               |
        |               *---- ... header files ...
        +----lib/
        |       |
        |       +----pkgconfig/
        |       |       |
        |       |       *---- libmlp-cpp.pc
        |       |
        |       *---- ... library files ...
        +----share/
                |
                +----libmlp-cpp/
                        |
                        +----dtd/
                              |
                              *---- ... MLP DTD files ...
```

Chapter 3. Using MLP Library

3.1. Features

3.1.1. Supported Messages

MLP Library implements version 3.3 of MLP. Mobile Location Protocol specification includes many features and services. In this release there is only *Standard Location Immediate Service* supported.

Standard Location Immediate Service messages:	• Standard Location Immediate Request • Standard Location Immediate Answer • Standard Location Immediate Report
Other supported messages	• General error message

3.1.2. Supported Shapes

Location of the subscribers is returned in Shape XML element. MLP specification support several types of shapes. MLP Library supports only Point type. Other shapes are on our TODO list.

Point

3.2. Initialization and Termination

To use MLP Library you have to include main header file MlpLib.h into your source code. This will include necessary initialization macros and MlpParser object. Initialization macro `MLP_LIB_CPP_INIT` must be run before any other object from MLP Library is used. It should be run exactly once at the beginning of your code. There is also macro `MLP_LIB_CPP_TERMINATE` used to clean environment before exit. It should also be called exactly once at the and of your program. Be aware that all objects from MLP Library that were created should be deleted before `MLP_LIB_CPP_TERMINATE` macro call.

All magic that is done in the macros is wrapping around *Xerces-c* initialization a termination process. On the following code you can find sample initial program that is initializing a terminating environment.

```
#include <iostream>

#include <libmlp-cpp/MlpLib.hpp>


int main( int argc, char* argv[ ] )
{
    // Initialization of environment
    MLP_LIB_CPP_INIT;

    MlpParser * myMlpParser = new MlpParser();


    delete myMlpParser; // Delete before termination macro
                        // Use dynamically allocated MlpParser so you
                        // can manually delete it before termination
                        // macro.
```

```
    // Initialization of environment
    MLP_LIB_CPP_TERMINATE;

    return 0;
}
```

3.3. Using Parser and Results

After you initialize environment you can create MLP Parser object that can process MLP messages. *MlpParser* does not use any multi threading techniques so DO NOT USE one instance of one parser in more threads without your own synchronization. There can coexist more *MlpParser* in one program and they can work simultaneously. There can be for example one parser per each thread. After the *MlpParser* is created you can use it to process MLP message as many times as you want, without any reinitialization.

Create Parser

After the initialization of the environment you can create a *MlpParser*. Use dynamically allocated *MlpParser* so it can be deleted before `MLP_LIB_CPP_TERMINATE` macro is called. Ore use any other technique that can guarantee that *MlpParser* is deleted before terminating macro.

```
#include <libmlp-cpp/MlpLib.hpp>

... some code

    MlpParser * myMlpParser = new MlpParser();
```

Configuration and Initialization

MlpParser uses local DTD files to validate MLP messages. DTD files are not read from Internet for case where program that is using MLP Library does not have an access to the Internet. Now it is default and only behavior. To pass a file-path to DTD files to parser use *MlpConfig* object.

```
// Need MlpConfig.hpp
#include <libmlp-cpp/MlpConfig.hpp>

... some code

    MlpConfig config;
    config.setDTDpath( "../../dtd/" );
```

Now you can pass the *config* object to the parser. Than you have to initialize parser. This initializes some *Xerces-c* options.

```
// Need MlpException.hpp
#include <libmlp-cpp/MlpException.hpp>

 ... some code

    try {
        // Configure and initialize the parser
        myMlpParser->configure( &config );
        myMlpParser->init();
    }
    catch ( MlpException &e ) {
        // Print error message in case of parser initialisation error
```

```
        std::cout << e.getMessage() << std::endl;

        // Delete in case of exception
        delete myMlpParser;
        myMlpParser = NULL;
    }
```

Parsing the MLP Message

MlpParser takes two arguments. First one is object *MlpMessage*. It has to be empty. *MlpParser* puts information from MLP messages into it. The second one argument is a string containing some MLP message that is ready to be parsed. *MlpParser* reads `std::string` as source data. For future releases reading from streams is planned.

```
// Need MlpMessage.hpp
#include <libmlp-cpp/MlpMessage.hpp>

 ... some code

    // Create new clean MlpMessage
    MlpMessage parsedMsg;

    // Create sample MLP Message
    std::string myMsg =  "... Put here your MLP message ..."

    try {
        // Process myMsg
        myMlpParser->parse( &parsedMsg, myMsg );
    }
    catch ( MlpException &e ) {
        // Print error message in case of parsing error
        std::cout << e.getMessage() << std::endl;
    }
```

Using the Results

If everything goes well you have all information from MLP message `std::string myMsg` in `parsedMsg` object. Useful information can be accessed by simple calling of member functions of this object. Most important member functions are:

getMsgType()	Returns type of message. That meas either service initiation, service response or general error message.
getServiceType()	Returns type of service message. This determines exact type of service message: *Standard Location Immediate Request* (SLIR), *Standard Location Immediate Answer* (SLIA), etc...
getMlpHdr()	Returns pointer to the MLP message header. More detailed information about header can be retrieved using received pointer. For example You can retrieve a client list. Client list is `std::list` of `MlpClients` which are entities that requested the localization. In response messages header is not mandatory so this function can return *null pointer*.
getSubscriberList()	Returns pointer to the `std::list` of `MlpSubscribers`. `MlpSubscriber` contains subscribers that should be localized.

getPositionList()	Returns pointer to the `std::list` of `MlpPositions`. `MlpPosition` contains subscribers locations.

Information returned by the member functions of `MlpMessage` are either strings, pointers or constants. All constants important to user of *MLP Library* are in `MlpConstants.h` file.

Detailed information about member functions of different object can be found in API reference on *MLP Library* web-pages or documentation directory.

Reusing the Parser and Cleaning the Environment

You can reuse `MlpParser` object after parsing a message. Previous results do not affect future results. `MlpParser` is completely cleaned after the parsing is done.

```
    ... some code

    delete myMlpParser;

    MLP_LIB_CPP_TERMINATE;

    return 0;
}
```

3.4. Compiling the Example

To compile this example you have to compile your source code and link it with libmlp-cpp library file and xerces-c library. Here you see a simple `Makefile` (Too see compleete simple makefile look into examples/example_other/example1_simple.tar.gz).

```
CXXFLAGS= -g -Wall -c


#libraries to link with
LIBS=-lxerces-c -lmlp-cpp


all:
    gcc -MMD -MP $(CXXFLAGS) main.cpp -o main.o
    gcc -o example-guide.bin main.o ${LIBS}
```

This can only work if you have properly installed MLP library and Xerces-c library. You need to properly set environment variables so the linker knows where the libraries and headers are installed. On Linux libraries are searched according to `/etc/ld.so.cahe`. Default path is most often `/usr/lib`. If you install MLP library into the /usr, Makefile displayed above should work. If you plan to install MLP library in some nonstandard directory you need to properly set environment variables and paths to headers.

Name of the environment variable that specifies where linker look for libraries depends on your system. For most UNIX platforms it is *LD_LIBRARY_PATH*. On AIX it is *LIBPATH*, on Mac OS X it is *DYLD_LIBRARY_PATH*, and on HP-UX it is *SHLIB_PATH*. For Cigwin or MinGW Windows *PATH* variable is used.

Other way to use MLP library in your project is to use GNU auto tools. You can use pkg-config in your `configure.ac` file to configure your project to automatically read information from `libmlp-`

`cpp.pc` and use them to create `Makefile`. You need to properly set *$PKG_CONFIG_PATH* environment variable for pkg-config to find `libmlp-cpp.pc`. On the following program listing you can see the part of `configure.ac` that check availability of `libmlp-cpp.pc` and `xerces-c.pc` and reads information from them to variables. This variables are then substituted in `Makefile.am`. (Too see compleete autotools `configure.ac` and `Makefile.am` look into `examples/example_other/example1_autotools.tar.gz`).

Part of configure.ac:

```
PKG_CHECK_MODULES(LIBMLP, libmlp-cpp)
AC_SUBST(LIBMLP_CFLAGS)
AC_SUBST(LIBMLP_LIBS)

PKG_CHECK_MODULES(XERCESC, xerces-c)
AC_SUBST(XERCESC_CFLAGS)
AC_SUBST(XERCESC_LIBS)
```

Part of Makefile.am:

```
INCLUDES = -I@top_srcdir@ @XERCESC_CFLAGS@ @LIBMLP_CFLAGS@

noinst_PROGRAMS = example-guide.bin
example_guide_bin__SOURCES = main.cpp
example_guide_bin_LDADD = @XERCESC_LIBS@ @LIBMLP_LIBS@
```

For example if you installed MLP library into `/var/libmlp-cpp` your `libmlp-cpp.pc` file would be in `/var/libmlp-cpp/lib/pkgconfig`. Then to compile your program you would execute:

```
example $ export $PKG_CONFIG_PAHT=/var/libmlp-cpp/lib/pkgconfig; ./configure
example $ make
```

This way `libmlp-cpp.pc` file is found by pkg-config and LIBMLP_CFLAGS and LIBMLP_LIBS are filled with proper path to MLP library headers and libraries.

Chapter 4. Examples

This chapter describes examples available in `examples` subdirectory:

```
libmlp-cpp-1.1.0
          |
          +----examples/
                 |
                 *----curl-client/
                 *----example1/
                 *----example2/
                 *----example3/
                 *----example-guide/
                 *----example_other/
```

curl-client
: Curl-client example need *cURL* library. It is simple HTTP client that asks user for Mobile Station International Subscriber Directory Number (MSISDN) and send HTTP POST request containing MLP *SLIR* message to Location Server. Location server should answer with *SLIA* message. This message is processed with MlpParser object and acquired location information is displayed to a standard output.

example1-3
: This three examples are completely coincidental except each contains different MLP message. They read the MLP message from `message.xml` file into the `std::string` and pass it to the *MlpParser*. `Example1` contains *SLIR* message, `example2` contains *SLIA* message and `example3` contains *SLIREP* message.

example-guide
: In `example-guide` you can see simple usage of MLP Library. Following code listing is from `example-guide/main.cpp`.

example_other
: Directory `example_other` contains examples that are not part of library build system. You can copy them to arbitrary location, unpack them an try to compile them according to instructions in their READMEs. You can find there example with simple Makefile and example formated as GNU autotools package. They have the same source code as the `example1` but they have separated build system. This means you need to properly set paths to libraries, headers and pkg-config so your compiler finds MLP headers and your linker finds library to link.

  Examples from `examples_other` need to know the path to MLP DTDs. To successfully run them after compilation. You need to edit this path in example source codes before compilation!!!

All examples that are part of build system (curl-client, example1-3, example-guide) are set to reach DTD files located in libmlp-cpp-1.1.0/libmlp-cpp/dtd. They should run after the *make* command. Path to DTDs can be specified directly in source code for simplicity of examples.

Foloving code listing is source for `example-guide`.

```
#include <iostream>
#include <sstream>
#include <string>
#include <fstream>

#include <libmlp-cpp/MlpLib.hpp>
#include <libmlp-cpp/MlpMessage.hpp>
#include <libmlp-cpp/MlpException.hpp>
#include <libmlp-cpp/MlpSubscriber.hpp>
```

```
#include <libmlp-cpp/MlpShapePoint.hpp>


int main( int argc, char* argv[] )
{
    // Initialize
    MLP_LIB_CPP_INIT;

    // Message for parsing
    std::string myMsg =  "<?xml version = \"1.0\" ?> \
<!DOCTYPE svc_init PUBLIC \"-//OMA//DTD MLP 3.2.0//EN\" \
\"http://www.openmobilealliance.org/Tech/dtd/MLP_SVC_INIT_320.dtd\"> \
<svc_init ver=\"3.2.0\"> \
    <hdr ver=\"3.2.0\"> \
        <client> \
            <id>theasp</id> \
            <pwd>thepwd</pwd> \
            <serviceid>0005</serviceid> \
            <requestmode type=\"ACTIVE\"/> \
        </client> \
        <subclient last_client=\"YES\"> \
            <id>theSubAsp</id> \
            <pwd>thepwd2</pwd> \
            <serviceid>0005</serviceid> \
        </subclient> \
        <requestor type=\"MSISDN\"> \
            <id>461018765710</id> \
        </requestor> \
    </hdr> \
    <slir ver=\"3.2.0\" res_type=\"ASYNC\"> \
        <msids> \
            <msid enc=\"CRP\" type=\"IPV4\">93.10.0.250</msid> \
            <codeword>rtep1</codeword> \
            <session type=\"APN\">mySession1</session> \
            <msid_range> \
                <start_msid> \
                    <msid enc=\"CRP\" type=\"IPV4\">461018765710</msid> \
                </start_msid> \
                <stop_msid> \
                    <msid enc=\"CRP\" type=\"IPV4\">461018765712</msid> \
                </stop_msid> \
            </msid_range> \
            <msid type=\"ASID\">441728922342</msid> \
            <codeword>rtep2</codeword> \
            <msid_range> \
                <start_msid> \
                    <msid>461018765720</msid> \
                </start_msid> \
                <stop_msid> \
                    <msid>461018765728</msid> \
                </stop_msid> \
            </msid_range> \
            <codeword>ahoj1</codeword> \
            <codeword>ahoj2</codeword> \
            <codeword>ahoj3</codeword> \
            <codeword>ahoj4</codeword> \
        </msids> \
        <eqop> \
            <resp_req type=\"LOW_DELAY\" /> \
            <hor_acc qos_class=\"ASSURED\">1000</hor_acc> \
        </eqop> \
        <geo_info> \
            <CoordinateReferenceSystem> \
                <Identifier> \
```

```
                    <code>4004</code> \
                    <codeSpace>EPSG</codeSpace> \
                    <edition>6.1</edition> \
                </Identifier> \
            </CoordinateReferenceSystem> \
        </geo_info> \
        <loc_type type=\"CURRENT_OR_LAST\" /> \
        <prio type=\"HIGH\" /> \
    </slir> \
</svc_init>" ;

    MlpConfig config;
    config.setDTDpath ("../../dtd/" );

    // New MlpParser
    MlpParser * myMlpParser = new MlpParser();
    try
    {
        myMlpParser->configure( &config );
        myMlpParser->init();
    }
    catch (MlpException &e)
    {
        // Error in initialization
        std::cout << e.getMessage() << std::endl;
        delete myMlpParser;
        myMlpParser = NULL;
    }

    if( myMlpParser != NULL )
    {
        std::cout << "Press Enter to start parsing." << std::endl;
        std::cin.get();

            try{
                MlpMessage inMsg;
                MlpHdr* inMlpHdr;

                const MlpHdrClientList* inMlpHdrClientList;
                const MlpSubscriberList* inMlpSubscriberList;
                const MlpPositionList* inMlpPositionList;

                // Parsing
                myMlpParser->parse( &inMsg, myMsg );


                std::cout << "   Msg Type:         "
                          << inMsg.getMsgType() << std::endl;
                std::cout << "   Msg Service Type: "
                          << inMsg.getServiceType() << std::endl;

                inMlpHdr = inMsg.getMlpHdr();
                // If message contains header, print some information
                if (inMlpHdr != NULL)
                {
                    std::cout << "      Hdr Request Mode: "
                              << inMlpHdr->getRequestMode() << std::endl;

                    inMlpHdrClientList = (inMsg.getMlpHdr())->getHdrClientList();
                    for ( t_MlpHdrClientListIterator it = inMlpHdr->getBegin() ;
                            it != inMlpHdr->getEnd() ;
                            it++
                            )
                    {
```

```
                        std::cout << "        Klient type:  "
                                  << (*it)->getClientType() << std::endl;
                        std::cout << "          Client ID:    "
                                  << (*it)->getId() << std::endl;
                    }
                }


                // Present only if the message contains subscribers that should be
located.
                // Present only in Request messages.
                inMlpSubscriberList = inMsg.getSubscriberList();
                if ( inMlpSubscriberList != NULL)
                {

                    for ( t_MlpSubscriberListIterator
                            it  = (inMsg.getSubscriberList())->begin();
                            it != (inMsg.getSubscriberList())->end() ;
                            it++
                        )
                    {
                        if( (*it)->getSubscriberType() ==
MlpConstants::SUBSCRIBER_MSID )
                        {
                            std::cout << "      Subscriber type: MSID: "
                                      << (*it)->getMsid() << std::endl;
                        }
                        else if( (*it)->getSubscriberType()
                                == MlpConstants::SUBSCRIBER_MSID_RANGE )
                        {
                            std::cout << "      Subscriber type: MSID_RANGE" <<
std::endl;
                            std::cout << "          Start MSID:      "
                                      << (*it)->getStartMsid() << std::endl;
                            std::cout << "          Stop MSID:       "
                                      << (*it)->getStopMsid() << std::endl;
                        }
                    }
                }

                // Present only in response messages
                // Prints information oubt localized subscribers
                inMlpPositionList = inMsg.getPositionList();
                if ( inMlpPositionList != NULL)
                {

                    for ( t_MlpPositionListIterator
                            it  = (inMsg.getPositionList())->begin();
                            it != (inMsg.getPositionList())->end() ;
                            it++
                        )
                    {
                        std::cout << "      Position: " << std::endl;
                        std::cout << "          MSID:       "
                                  << (*it)->getMsid() << std::endl;
                        // error during localization
                        if( (*it)->wasError() )
                        {
                            std::cout << "          Error:       "
                                      << (*it)->getResult() << std::endl;
                            std::cout << "          Error Id:    "
                                      << (*it)->getResultId() << std::endl;
                        }
                        // localization was successful
```

```
                        else
                        {
                            std::cout << "          Shape type:  "
                                      << ( (*it)->getShape() )->getShapeType()
                                      << std::endl;
                            switch( ( (*it)->getShape() )->getShapeType() ){
                                case( MlpConstants::SHAPE_POINT ):
                                    std::cout << "          X:          "
                                              << ( (MlpShapePoint* )(*it)-
>getShape() )->getX()
                                              << std::endl;
                                    std::cout << "          Y:          "
                                              << ( (MlpShapePoint* )(*it)-
>getShape() )->getY()
                                              << std::endl;
                                    std::cout << "          Z:          "
                                              << ( (MlpShapePoint* )(*it)-
>getShape() )->getZ()
                                              << std::endl;
                                    break;
                                    // Other shapes not supported yet
                                case( MlpConstants::SHAPE_LINESTRING ):
                                    break;
                                case( MlpConstants::SHAPE_POLYGON ):
                                    break;
                                case( MlpConstants::SHAPE_BOX ):
                                    break;
                                case( MlpConstants::SHAPE_CIRCULAR_AREA ):
                                    break;
                                case( MlpConstants::SHAPE_CIRCULAR_ARC_AREA ):
                                    break;
                                case( MlpConstants::SHAPE_ELLIPTICAL_AREA ):
                                    break;
                                case( MlpConstants::SHAPE_MULTILINE_STRING ):
                                    break;
                                case( MlpConstants::SHAPE_MULTIPOINT ):
                                    break;
                                case( MlpConstants::SHAPE_MULTIPOLYGON ):
                                    break;
                                case( MlpConstants::SHAPE_LINEAR_RING ):
                                    break;
                            }
                        }
                    }
                }
            }
            catch ( MlpException &e ){
                std::cout << e.getMessage() << std::endl;
            }
    }

    std::cout << "Press Enter to stop." << std::endl;
    std::cin.get();

    // Delete the parser
    delete myMlpParser;

    // Terminate the environment
    MLP_LIB_CPP_TERMINATE;

    return 0;
}
```