# XMMEP 消息实现指导

## 一、名字空间规定

XMMEP 协议的所有消息定义在：namespace xmessenger 的名字空间。

## 二、消息基类

基类的定义如下，所有 XMMEP 消息类都派生自该类。

```
/**
 * Base class of all XMMEP Messages.
 */
class XMMessage {
public:
    virtual ~XMMessage(); // virtual destructor.

    /**
     * @brief Serializes  XMMessage to a byte array.
     * @param outputBuffer Byte array where XMMessage put to.
     *
     * @return Message length
     */
    size_t encode(char* outputBuffer) const;

    /**
     * @brief Get XMMessage from a byte array.
     * @param inputBuffer Byte array where XMMessage get from.
     *
     * @return true if decode operation success, false otherwise.
     */
    bool decode(const char* inputBuffer);

    /**
     * @brief Get XMMessage type.
     */
    virtual int code() const = 0;

    /**
     * @brief Put XMMessage content to output stream.
     */
    virtual std::ostream& output(std::ostream& os) const = 0;

    std::string toString() const;

protected:
    virtual size_t encodeBody(char* outputBuffer) const = 0;

    virtual const char* decodeBody(const char* inputBuffer) = 0;

};

std::ostream& operator<<(std::ostream& os, const XMMessage& msg);
```

```
} // end of namespace xmessenger
```

基类的部分函数实现：

```
size_t xmessenger::XMMessage::encode(char* outputBuffer) const {
    // Step1: encode body first
    size_t bodyLength = encodeBody(outputBuffer + HEAD_LENGTH);
    char c = outputBuffer[HEAD_LENGTH];

    // Step2: encode header
    std::sprintf(outputBuffer, "%4d%4d", bodyLength, code());
    outputBuffer[HEAD_LENGTH] = c;
    outputBuffer[bodyLength + HEAD_LENGTH] = '\0';
    return bodyLength + HEAD_LENGTH;
}

bool xmessenger::XMMessage::decode(const char* inputBuffer) {
    decodeBody(inputBuffer + HEAD_LENGTH);
    return true;
}

std::string xmessenger::XMMessage::toString() const {
    std::ostringstream os;
    output(os);
    return os.str();
}

std::ostream& xmessenger::operator<<(std::ostream& os,
        const xmessenger::XMMessage& msg) {
    return msg.output(os);
}
```

## 四、具体消息实现

具体消息类型都派生自基类 xmessenger::XMMessage，子类需要实现基类中的纯虚函数。下面以登录请求 MSG_LOGIN_REQ 为例：

消息定义，注意消息类型的枚举量放在每种消息的定义头文件中，比如 MSG_LOGIN_REQ：

```
namespace xmessenger {

enum {
    MSG_LOGIN_REQ = 112
};

class LoginRequest: public XMMessage {
public:
    LoginRequest();
    LoginRequest(const std::string& _userName, const std::string&
_password, const Presence& _loginStatus = ONLINE);

    virtual int code() const;
    virtual std::ostream& output(std::ostream& os) const;

    Presence getLoginStatus() const;
```

```
        std::string getPassword() const;
        std::string getUserName() const;

        void setLoginStatus(const Presence& loginStatus);
        void setPassword(const std::string& password);
        void setUserName(const std::string& userName);

    protected:
        virtual size_t encodeBody(char* outputBuffer) const;

        virtual const char* decodeBody(const char* inputBuffer);

    private:
        std::string userName;
        std::string password;
        Presence loginStatus;
    };

    } // end of namespace xmessenger
```

部分成员函数的实现：

```
    int xmessenger::LoginRequest::code() const {
        return MSG_LOGIN_REQ;
    }

    std::ostream& xmessenger::LoginRequest::output(std::ostream& os) const {
        os << "LoginRequest: userName[" << userName;
        os << "], password[***], loginStatus[";
        os << loginStatus << "]";

        return os;
    }

    size_t xmessenger::LoginRequest::encodeBody(char* outputBuffer) const {
        char* cur = encodeStringField(outputBuffer, userName);
        *cur = FIELDS_DELIM;
        cur = encodeStringField(++cur, password);
        *cur = FIELDS_DELIM;
        std::sprintf(++cur, "%1d", loginStatus);

        return std::strlen(outputBuffer);
    }

    const char* xmessenger::LoginRequest::decodeBody(const char*
inputBuffer) {
        const char* cur = decodeStringField(inputBuffer, userName);
        cur = decodeStringField(cur, password);

        int p;
        cur = decodeIntField(cur, p);
        loginStatus = Presence(p);

        return cur;
    }
```

成员函数 encodeBody()和 decodeBody()调用了一些辅助函数，比如 encodeIntField()和 decodeIntField()，以下是参考实现：

```
char* xmessenger::encodeIntField(char* outBuf, const int& field) {
    std::sprintf(outBuf, "%d", field);
    return outBuf + std::strlen(outBuf);
}

const char* xmessenger::decodeIntField(const char* inBuf, int& field) {
    const char* pos = std::strpbrk(inBuf, fieldDelim);
    if (0 != pos) {
        size_t len = pos - inBuf + 1;
        char tmp[12] = "";
        std::memcpy(tmp, inBuf, len - 1);
        field = std::atoi(tmp);
        return inBuf + len;
    }
    field = std::atoi(inBuf);
    return inBuf + (std::strlen(inBuf) + 1);
}
```

注：上述参考实现中，消息的 encode 操作并不安全，有造成字节数组溢出的隐患存在。

## 五、XMMEP 使用示例

消息的 encode：

```
// encode and send XMMEP message
xmessenger::LoginRequest req("kwarph", "123456", xmessenger::ONLINE);
char buf[xmessenger::MAX_MSG_SIZE + 1] = { };
size_t len = req.encode(buf);

sock.send(buf, len); // send message
```

消息的 decode：

```
// receive and decode XMMEP message
char recvbuf[xmessenger::MAX_MSG_SIZE + 1] = { };
sock.recv(recvbuf, sizeof recvbuf); // recv message

xmessenger::LoginRequest msg; // an empty message object
msg.decode(recvbuf);
cout << msg.code() << '\n'; // print message's code
cout << msg.getUserName() << '\n';
cout << msg << '\n'; // print message
```

## 六、命名的简要约定

1，名字尽量具有自描述性。

2，自定义类型名（包括但不限于：类、枚举）：首字大写，中间不带下划线等分隔字符，中间单词首字大写，举例说明：

```
class TheClassName;
```

```
enum MessageType { /* … */ };
```

3，函数名：首字小写，中间不带下划线等分隔字符，中间单词首字大写，举例说明：

```
int testFunc1();
```

4，对象名（变量名）：首字小写，中间不带下划线等分隔字符，中间单词首字大写，小范围使用的名字尽量简约，大范围使用的名字尽量具有自描述性，举例说明：

```
Message sendFileRequest;
for(int  i = 0; i < N; ++i) { /* … */ }
```

5，常量：全部字母大写，单词之间用下划线分隔，如：

```
enum {
    MSG_USER_ONLINE_NOTYFY = 100
};
const int MSG_COUNT = 12;
static const int FULL_CYCLES = 6;
```