qmsgpack Documentation

Release 1.0.0rc

Roman Isaikin

Dec 06, 2017

Contents

CHAPTER 1

Installation

Contents

qmsgpack is a pure Qt library (Qt4 and Qt5 supported), so you can build it for almost any platform Qt supports. There are two build methods:

- CMake
- qmake

And two ways of using it: build separately and include to your project, or build with your project (qmake subdirs)

1.1 Build

1.1.1 CMake

Get the latest qmsgpack version by grabbing the source code from GitHub:

```
$ git clone https://github.com/romixlab/qmsgpack.git
```

Now build and install it:

```
cd qmsgpack
mkdir build && cd build
cmake ..
make
sudo make install
```

There are several useful cmake options available:

-DBUILD_TESTS=True
: Build all the tests, run with `make tests`

-DCMAKE_INSTALL_PREFIX=/usr
: Change install location to */usr*

-DCMAKE_BUILD_TYPE=Debug
: Change build type to debug mode (default is *Release*), could be very useful if something goes wrong

-DWITH_GUI_TYPES=True
: Build with support for QtGui types (QColor)

-DWITH_LOCATION_TYPES=True
: Build with support for QtLocation types(QGeoCoordinate). Might not work, because CMake seems to be failing to find QtLocation, in this case you can try qmake instead.

Add options before `..` as follow:

```
cmake -DCMAKE_INSTALL_PREFIX=/usr -DBUILD_TESTS=True ..
```

Custom Qt installation

If you installed Qt with online installer, cmake will most likely not find it, in this case try adding following lines to CMakeLists.txt:

```
set(Qt5Core_DIR "/opt/Qt5.6.0/5.6/gcc_64/lib/cmake/Qt5Core")
set(Qt5Test_DIR "/opt/Qt5.6.0/5.6/gcc_64/lib/cmake/Qt5Test")
set(Qt5_DIR "/opt/Qt5.6.0/5.6/gcc_64/lib/cmake/Qt5Core")
set(QT_QMAKE_EXECUTABLE "/opt/Qt5.6.0/5.6/gcc_64/bin/qmake")
```

1.1.2 qmake

Get the latest qmsgpack version by grabbing the source code from GitHub:

```
$ git clone https://github.com/romixlab/qmsgpack.git
```

Now build and install it:

```
cd qmsgpack
qmake
make
sudo make install
```

Also you can just open `qmsgpack.pro` in Qt Creator and build it from there.

1.2 Use it

Just add following lines to your .pro file:

```
LIBS += -lqmsgpack
```

On Windows you may also set the `INCLUDEPATH` variable to appropriate location

CHAPTER 2

Basics

Contents

2.1 Standard types

Below are Qt equivalents to MessagePack types listed in spec:

MessagePack type	Qt or C++ type
positive fixint	quint8
fixmap, map	QMap, QHash
fixarray,array	QList
fixstr, str	QString
nil	QVariant()
false, true	bool
bin	QByteArray
float 32	float
float 64	double
uint 8	quint8
uint 16	quint16
uint 32	quint32
uint 64	quint64
int 8	qint8
int 16	qint16
int 32	qint32, int
int 64	qint64
negative fixint	qint8

You can pack and unpack any of those types right away:

2.1.1 Packing

Pass QVariant to `MsgPack::pack()` function:

```
QVariant v = 123;
QByteArray packed = MsgPack::pack(v);
qDebug() << packed.toHex();
```

Of course QVariant can contain a QVarianList or a QVariantMap:

```
QList<QVariant> list;
list << 123 << 4.56 << true;
QByteArray packed = MsgPack::pack(list);
qDebug() << packed.toHex();
```

Note: If you want to pack QList<int> for example see: REF TO MsgPackStream

2.1.2 Unpacking

Unpacking is handled by `MsgPack::unpack()` function:

```
QByteArray packed = MsgPack::pack("qwerty");
QVariant unpacked = MsgPack::unpack(packed);
qDebug() << unpacked.toString();
```

Tip: If packed data contains only one msgpack type (fixstr of fixmap for example), unpack will return it as `QVariant(QString())` and `QVariant(QMap())` respectively. But if there are several values packed,

`QVariant(QList())` will be returned (consider this 5 bool values packed without msgpack's list: [0xc3, 0xc3, 0xc3, 0xc3, 0xc3])

2.2 More types

There are built in packers and unpackers (basic and stream ones) for following types: `QPoint`, `QSize`, `QRect`, `QTime`, `QDate`, `QDateTime`, `QColor`, `QGeoCoordinate`. But since there is no such types in msgpack spec, ext type is used.

Example:

```
MsgPack::registerType(QMetaType::QPoint, 37); // 37 is msgpack user type id
QByteArray ba = MsgPack::pack(QPoint(12, 34));
qDebug() << MsgPack::unpack(ba).toPoint();
```

Note, that QColor and QGeoCoordinate is enabled by default only in qmake project.

2.3 Thread safety

All methods are thread safe, except `MsgPack::setCompatibilityModeEnabled` which is not.

`pack()` and `unpack()` do not use any global variables except user packers and unpackers, access to them is controlled via QReadLocker (and QWriteLocker when registering a new one), so readers do not block each other.

Warning: User packers and unpackers can break thread-safety! But in most cases they are so simple, so this is not a problem.

2.4 Compatibility mode

You can enable compatibility mode this way: `MsgPack::setCompatibilityModeEnabled(true)`, after that there will be no str8, and QByteArray will be packed to str.

CHAPTER 3

Streams

Contents

There are QDataStream analogue with almost identical API. Every basic type is supported as well as QList<T> and Qt types (QPoint, QSize, QRect, QTime, QDate, QDateTime, QColor, QGeoCoordinate). More types are on the way.

3.1 Packing

You can use any QIODevice derived class or QByteArray.

```
QByteArray ba;
MsgPackStream out(&ba, QIODevice::WriteOnly);
out << 1 << true << "Hello";
qDebug() << ba.toHex();
```

Of course you can unpack this byte array using `MsgPack::unpack`:

```
qDebug() << MsgPack::unpack(ba);
// output:
// QVariant(QVariantList, (QVariant(uint, 1), QVariant(bool, true), QVariant(QString,
↪"Hello")))
```

QList of any type are also supported:

```
QList<int> list;
list << 1 << 2 << 3;
QByteArray ba;
MsgPackStream out(&ba, QIODevice::WriteOnly);
out << list;
```

3.2 Unpacking

To unpack QByteArray just pass it by value, or use QIODevice::ReadOnly for other devices:

```
MsgPackStream in(ba);
QList<int> list2;
in >> list2;
qDebug() << list2; // (1, 2, 3)
```

3.3 More types

Include <qmsgpack/stream/geometry.h (or location.h or time.h) for additional types. And do not forget to register type, so that MsgPackStream will know which user type id to use.

```
#include <qmsgpack/stream/geometry.h>
// ...

MsgPack::registerType(QMetaType::QPoint, 3);
QByteArray ba;
MsgPackStream out(&ba, QIODevice::WriteOnly);
out << QPoint(1, 2) << QPoint();
qDebug() << MsgPack::unpack(ba);
// output:
// QVariant(QVariantList, (QVariant(QPoint, QPoint(1,2)), QVariant(QPoint, QPoint(0,
↪0))))
```

CHAPTER 4

Custom types

Contents

4.1 Custom QVariant

You can provide two functions for packing QVariant with custom type inside to QByteArray and vise versa. For example here is how QColor packer and unpacker looks like:

```
QByteArray pack_qcolor(const QVariant &variant)
{
    QColor color = variant.value<QColor>();
    if (!color.isValid())
        return QByteArray(1, 0);
    QByteArray data;
    data.resize(4);
    quint8 *p = (quint8 *)data.data();
    p[0] = color.red();
    p[1] = color.green();
    p[2] = color.blue();
    p[3] = color.alpha();
    return data;
}

QVariant unpack_qcolor(const QByteArray &data)
{
    if (data.length() == 1)
```

```
        return QColor();
    quint8 *p = (quint8 *)data.data();
    return QColor(p[0], p[1], p[2], p[3]);
}
```

And that's it! Now register this two functions:

```
MsgPack::registerPacker(QMetaType::QColor, 3, pack_qcolor); // 3 is msgpack ext type␣
↪id
MsgPack::registerUnpacker(3, unpack_qcolor);
```

After that `MsgPack::pack(QColor(127, 127, 127))` will start to work!

4.2 Custom stream

You can provide stream operators for any other type you might want to work with. But there are some pitfalls to consider if you want to unpack something with other MsgPack implementations.

Example:

```
class SomeType
{
    public:
        double x() const { return m_x; }
        void setX(double x) { m_x = x;}

        double y() const { return m_y; }
        void setY(double y) { m_y = y;}

        double z() const { return m_z; }
        void setZ(double z) { m_z = z;}

    private:
        double m_x, m_y, m_z;
};

MsgPackStream &operator<<(MsgPackStream &s, const SomeType &t)
{
    s.writeExtHeader(27, <msgpack user type id>); // size of packed double is 9 * 3 =␣
↪27
    s << t.x() << t.y() << t.z();
    return s;
}

MsgPackStream &operator>>(MsgPackStream &s, SomeType &t)
{
    quint32 len;
    s.readExtHeader(len);
    if (len != 27) {
        s.setStatus(MsgPackStream::ReadCorruptData);
        return s;
    }
    double x, y, z;
    s >> x >> y >> z;
    t.setX(x);
    t.setY(y);
```

```
    t.setZ(z);
    return s;
}
```

In this case size of data is known id advance, if this is not the case, then you can use QByteArray. Here is how QPoint operators are implemented:

```
MsgPackStream &operator<<(MsgPackStream &s, const QPoint &point)
{
    // we need to know user type id, that was registered with MsgPack::registerType
    qint8 msgpackType = MsgPack::msgpackType(QMetaType::QPoint);
    if (msgpackType == -1) {
        s.setStatus(MsgPackStream::WriteFailed);
        return s;
    }
    QByteArray ba;
    MsgPackStream out(&ba, QIODevice::WriteOnly); // stream inside stream ;)
    if (point.isNull()) { // save some bytes if point is invalid
        quint8 p[1] = {0};
        out.writeBytes((const char *)p, 1);
    } else {
        out << point.x() << point.y();
    }
    s.writeExtHeader(ba.length(), msgpackType); // only now write msgpack ext field
    s.writeBytes(ba.data(), ba.length()); // and variable length data
    return s;
}

MsgPackStream &operator>>(MsgPackStream &s, QPoint &point)
{
    quint32 len;
    s.readExtHeader(len); // read msgpack ext field
    if (len == 1) { // handle invalid QPoint
        point = QPoint();
        return s;
    }
    QByteArray ba;
    ba.resize(len);
    s.readBytes(ba.data(), len); // read len bytes to byte array
    MsgPackStream in(ba);
    int x, y;
    in >> x >> y;
    point = QPoint(x, y);
    return s;
}
```

Tip: Of course you can just stream out everything without any ext header and user type id's, like this: `s << point.x() << point.y(); return s;` but in that case you will not be able to unpack anything useful with MsgPack::unpack() or in other MsgPack implementations.

Contents:

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

Symbols

C