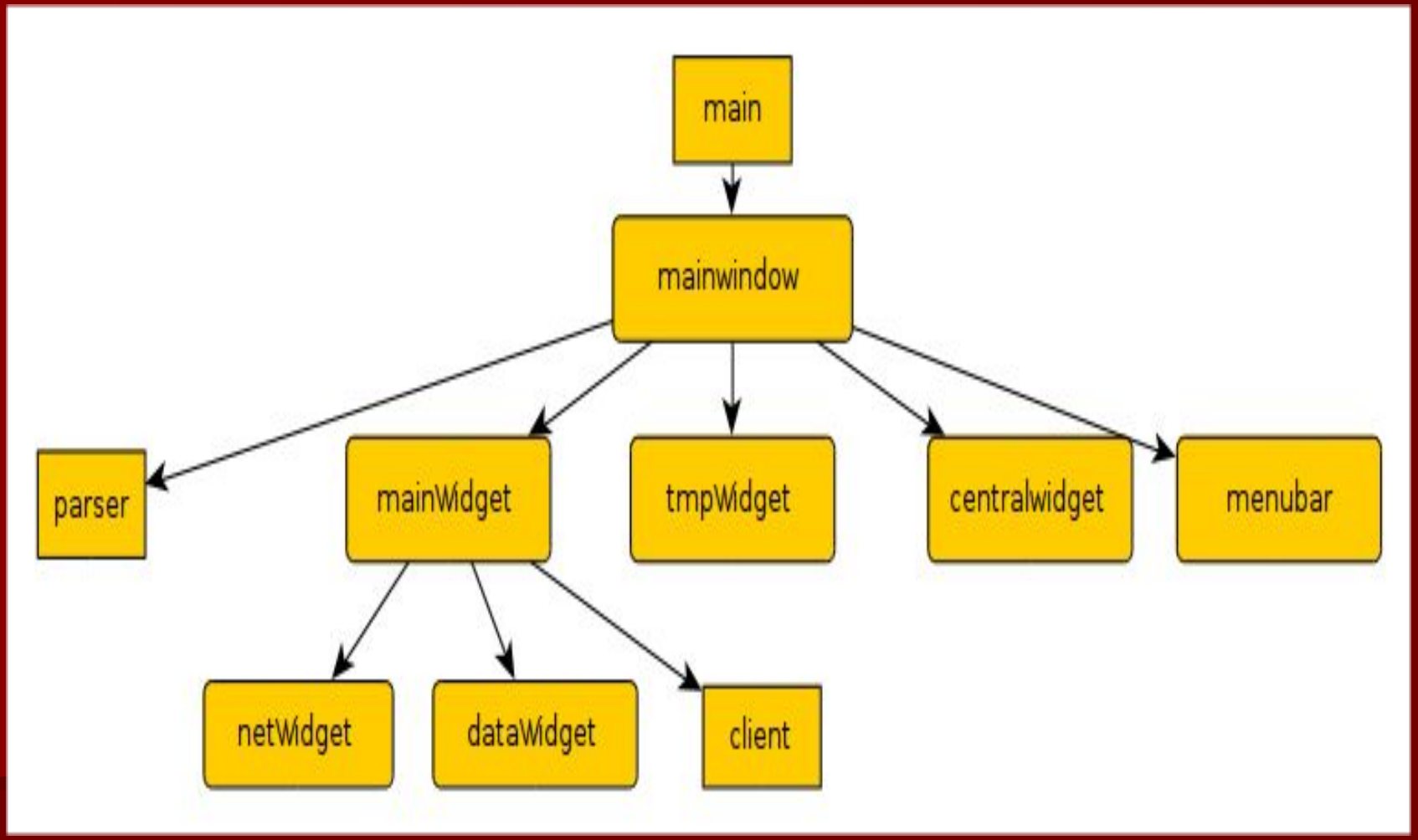
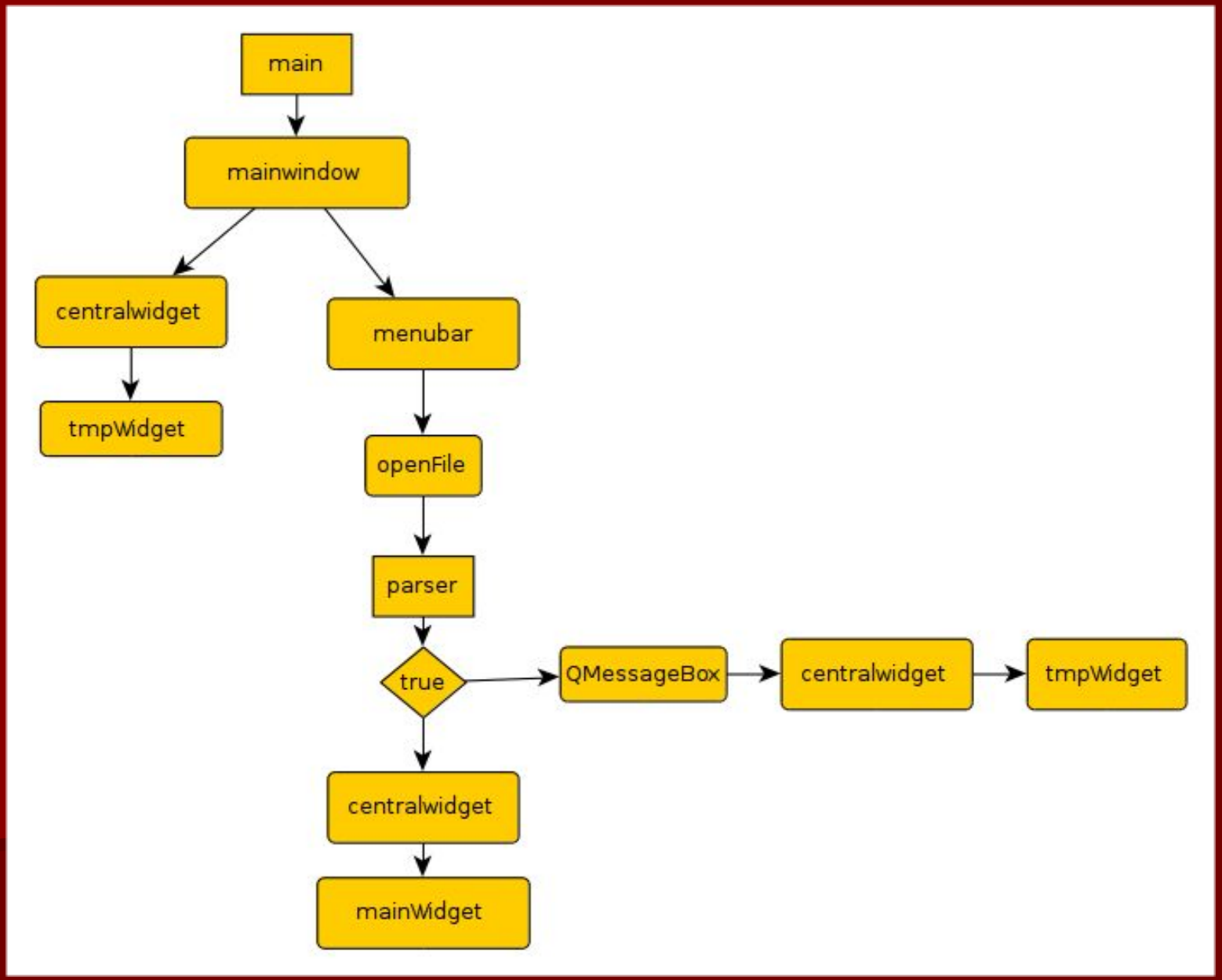
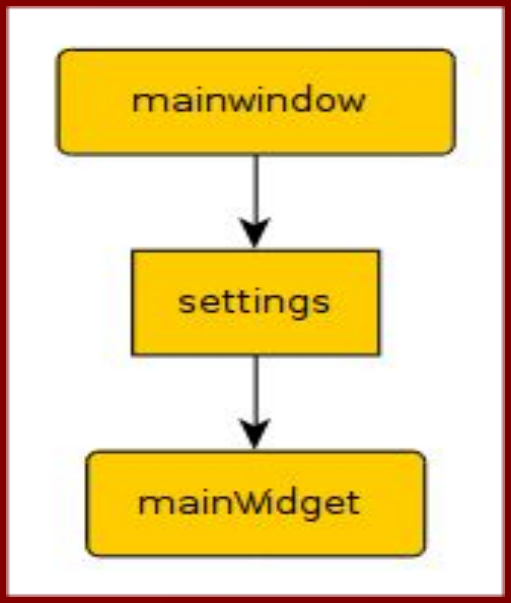
# Элементы приложения

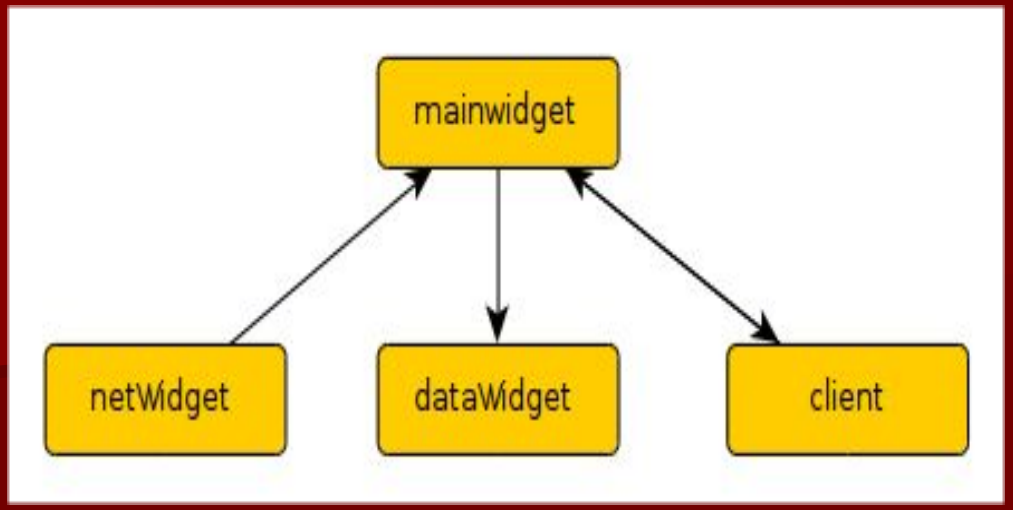


# Mainwindow

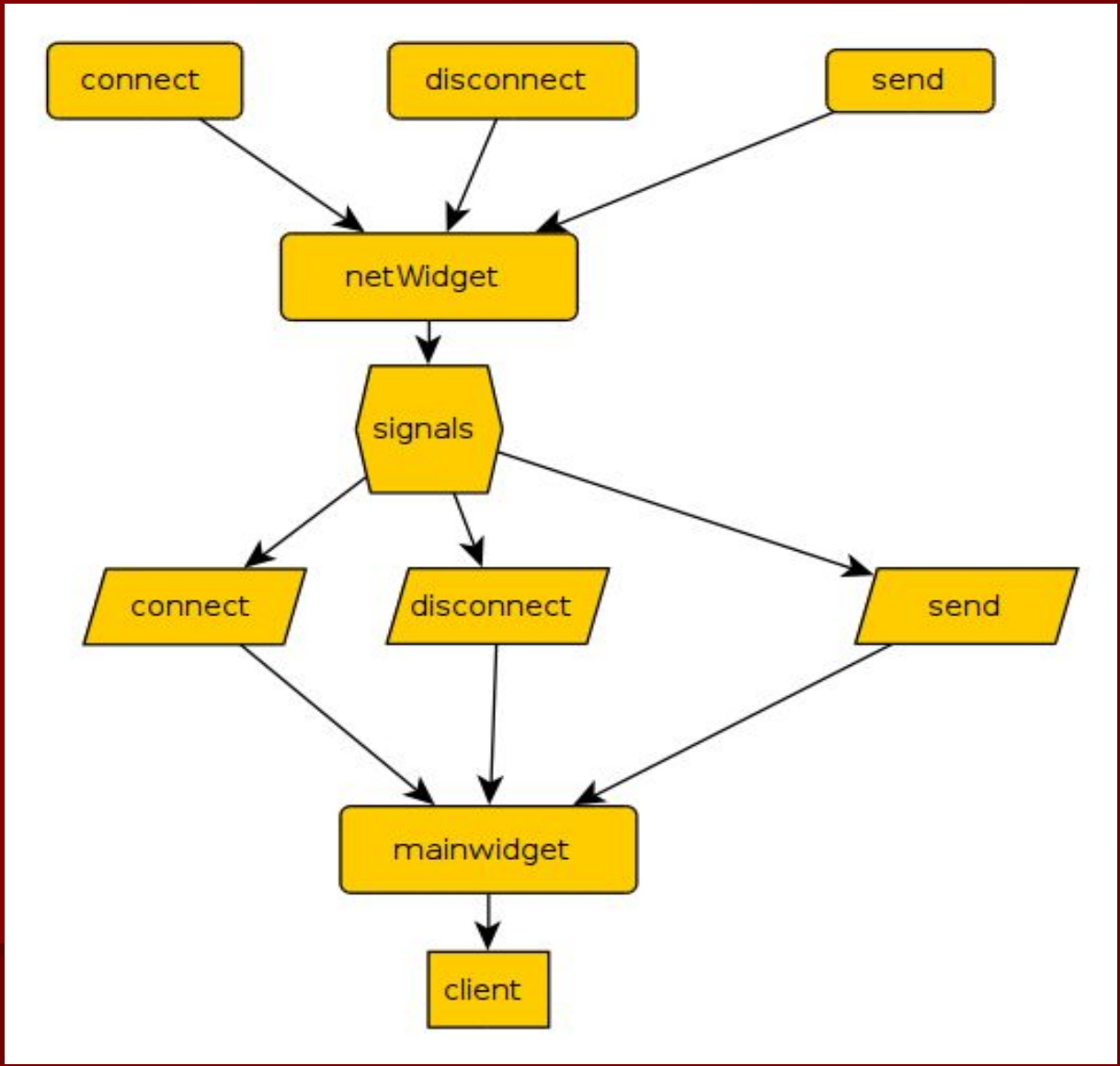

main
mainwindow
centralwidget
menubar
tmpWidget
openFile
parser
true
QMessageBox
centralwidget
tmpWidget
centralwidget
mainWidget

# Mainwidget

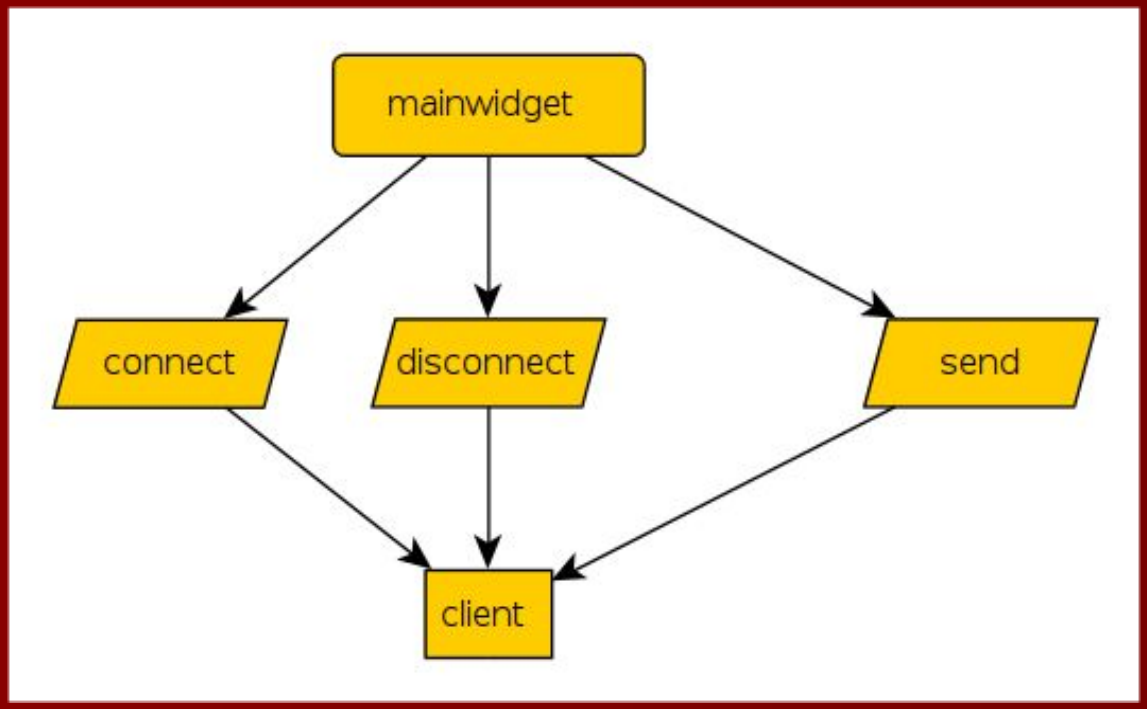

mainwindow
settings
mainWidget

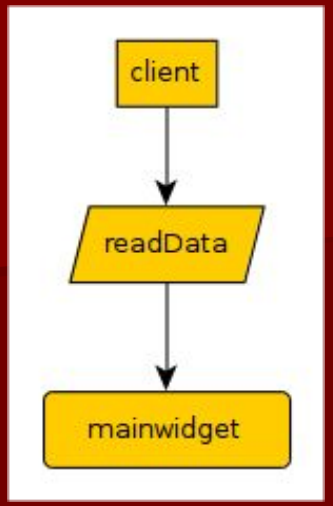

mainwidget
netWidget
dataWidget
client

# Взаимодействие netWidget и mainwidget

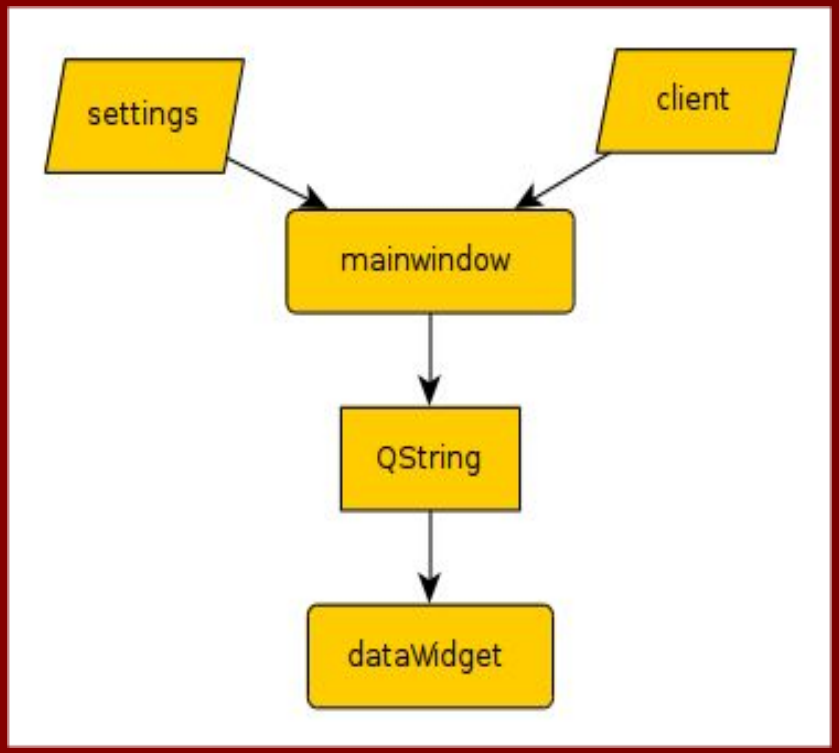


# Взаимодействие client и mainwidget

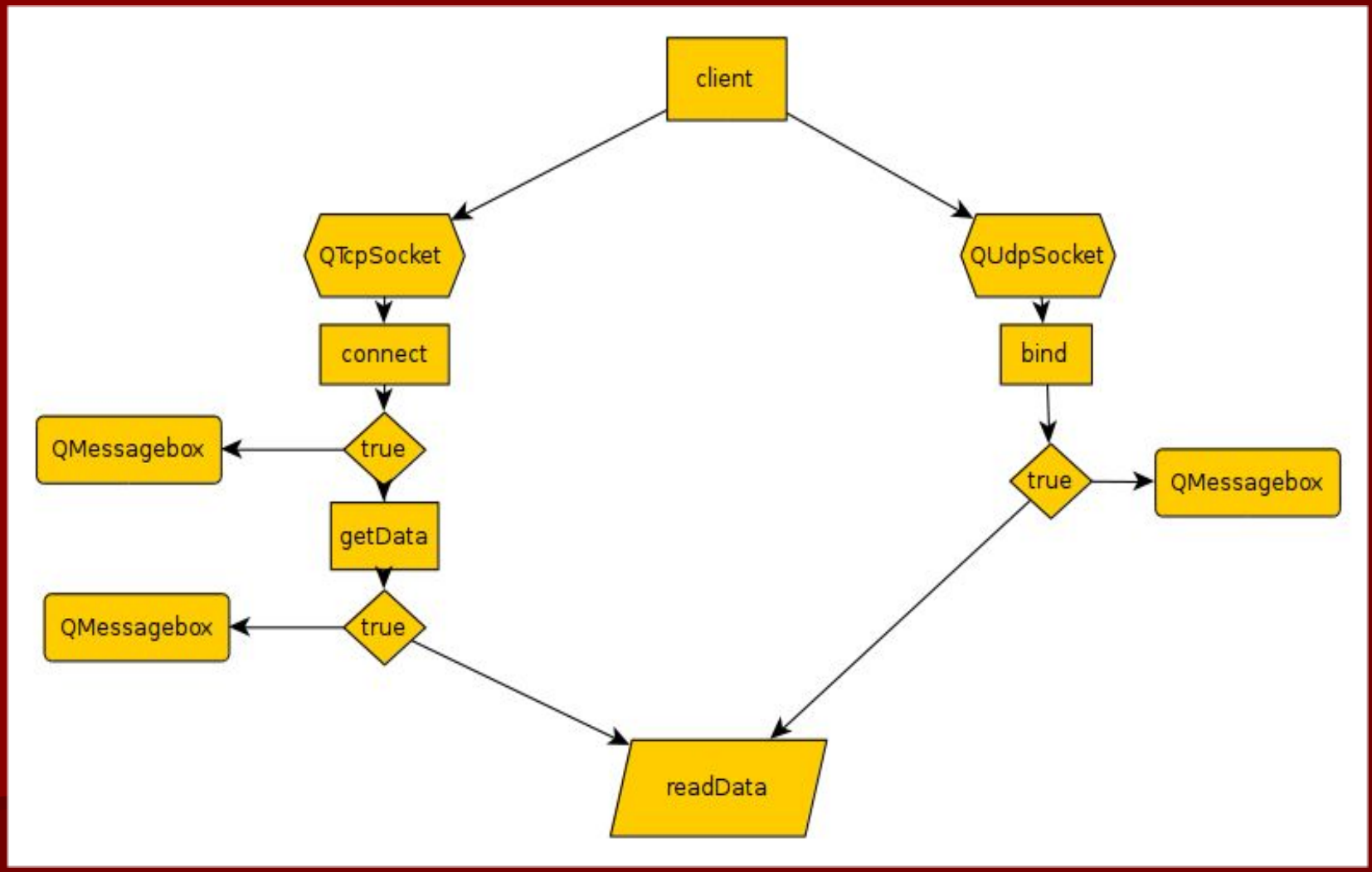


# Взаимодействие dataWidget и mainwidget



# Структура client



# QByteArray

Класс QByteArray предоставляет массив байт.
QByteArray может быть использован для хранения как сырых байт (включая и '\0'), так и традиционных 8-битных нуль-терминированых строк.
Использование QByteArray более удобно, чем использование const char *.
Здесь всегда гарантировано, что данные завершаются '\0' и используется неявное совместное использование (copy-on-write) для экономии памяти и избегания ненужного копирования данных.

```
QByteArray ba;
 ba.resize(5);
 ba[0] = 0x3c;
 ba[1] = 0xb8;
 ba[2] = 0x64;
 ba[3] = 0x18;
 ba[4] = 0xca;
```

# UdpClient

```
m_udpSocket = new QUdpSocket(this);

connect(m_udpSocket, SIGNAL(readyRead()),this, SLOT(readDataUdp()));
```

```
bool Client::setConnection(QString address, int port)
{
    m_address = address;
    m_port = port;
    m_udpSocket->bind(QHostAddress(m_address), m_port);
    return true;
}
```

# UdpClient

```
bool Client::readDataUdp()
{
    qDebug()<<"readDataUdp()";
    while (m_udpSocket->hasPendingDatagrams()) {
            QByteArray datagram;

            datagram.resize(m_udpSocket->pendingDatagramSize());
            qDebug()<<"datagram size=" <<datagram.size();
            QHostAddress sender;
            quint16 senderPort;

            m_udpSocket->readDatagram(datagram.data(), datagram.size(),
                                      &sender, &senderPort);

            processData(datagram);
        }

    return true;
}
```

Получает дейтаграмму не больше, чем datagram.size() байт и сохраняет его в данные. Определяет адрес хоста отправителя и порт.

# TcpClient

```
Client::Client()
{
    m_pTcpSocket = new QTcpSocket(this);


    connect(m_pTcpSocket, SIGNAL(connected()),this, SLOT(slotConnected()));
    connect(m_pTcpSocket, SIGNAL(readyRead()),this, SLOT(readData()));
    connect(m_pTcpSocket, SIGNAL(error(QAbstractSocket::SocketError)),
            this,         SLOT(slotError(QAbstractSocket::SocketError)));
}

bool Client::setConnection(QString address, int port)
{
    m_address = address;
    m_port = port;
    m_pTcpSocket->connectToHost(m_address, m_port);
    return true;
}
bool Client::slotConnected()
{
    qDebug()<<"connected()";
    return true;
}
```

# TcpClient

```
bool Client::readData()
{
    QDataStream in(m_pTcpSocket);
   in.setVersion(QDataStream::Qt_4_2);
   for (;;) {
       if (!m_nNextBlockSize) {
           if (m_pTcpSocket->bytesAvailable() < sizeof(quint16)) {
               break;
           }
           in >> m_nNextBlockSize;
       }

       if (m_pTcpSocket->bytesAvailable() < m_nNextBlockSize) {
           break;
       }
       QTime   time;
       QString str;
       in >> time >> str;
       m_nNextBlockSize = 0;
   }
    return true;
}
```

Слот r*eadData()*вызывается при поступлении данных от сервера. Цикл for нужен, так как не все данные с сервера могут прийти одновременно. Поэтому клиент должен быть в состоянии получить как весь блок целиком, так и только часть блока или даже все блоки сразу. Каждый переданный блок начинается полем, хранящим размер блока.

# TcpClient

```
bool Client::send()
{
    QByteArray  arrBlock;
    QDataStream out(&arrBlock, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_2);
    out << quint16(0) << QTime::currentTime() << m_ptxtInput->text();

    out.device()->seek(0);
    out << quint16(arrBlock.size() - sizeof(quint16));

    m_pTcpSocket->write(arrBlock);
    return true;
}
```

Мы не можем записывать данные сразу в *QTcpSocket*, потому что мы не знаем размер блока, который должен быть выслан в первую очередь. Поэтому мы должны сначала создать объект *QByteArray*, для того чтобы записывать все данные блока в него, записывая сначала размер равным 0. После того как все необходимые данные блока записаны, мы перемещаем указатель на начало блока и вызовом метода *seek()* записываем размер блока, который вычисляется как размер arrBlock с вычитанием из него sizeof(quint16). Это делается для исключения данных размера при подсчете байт.