qt c++ 5.14v using tcp/ip using multithreading and our application is server and as well as client, heartbeatmsg(.h,.cpp,.ui) have a structure like typedef struct{ QString ntype;,QString strSource,int n recevtTimestap ;}heartbeat;and . sourceID have enum struc{ SP=1,Dt,GT} in Globalvariable.h, and commonproperties .h and .cpp and call in DSC.properties in using Qmap<QString,QString>getproperties() const; .xml processor (.h,.cpp) generatexmlmessage to server using XSD.file(folder for all XSD files) by validating xml file using switch .heartbeatmsg use globalvariable,commonproperties ,it receive message from server or client in this for format heartbeartmessage::messagereceived(QString messageReceived,QStringMessageSource) and validate with commonproperties and xmlprocessor and also contain table in that void inserttotable(int nrow,QString strSource,QString strTimeStamp,bool status), heartbeatmsg object create in controlstatus(.h,cpp,.ui) ,heartbeart object call in controlstatus(connect(object,signal(sendstsof(QMap<int bool>),this,slot(controlstatusui(Qmap<int,bool>)))) it change the color if green is it means connected and if red is their not connected of three server by use this funcation(::contolstatusui(Qmap<int,bool>qmapconnectionstatus{ clear all(); Qmapiterator<int,bool>iterator(qmapconnectionstatus); Qmaplistofapp.clear(); Qmaplistofapp=qmapconnectionstatus; While(iterator.hasnext()) { Iterator.next(); Switch(iterator.key(); { Case sourceID::SP; controlstatus::spConn<iterator.value()>; Break; Case sourceID::DT; controlstatus::dpConn<iterator.value()>; Break; Case sourceID::GT; controlstatus::gtConn<iterator.value()>; Break;)) ,spconn,dpconn,gtconn is Boolean and it is connected to label of respected enitity label and it display on the .UI and we need to connect three different client(SP,DT and GT and ) and,tcp/ip of server and client class method call at mainwindow(.h,.cpp,.ui) using commonpropeties .,and ipaddress and port no place in the DCS.properties,, one server(SP) we need to send in the xml format of heartbeartmsg by creating .xsd file. remaining 2 server in the normal form using tcp/ip,provide server(simulator ) code also too

TCP/IP Heartbeat Monitoring System with Qt C++

Here's a comprehensive implementation of your heartbeat monitoring system using Qt 5.14 with multithreading capabilities, acting as both server and client.

1. Global Definitions (GlobalVariable.h)

cpp

Copy

Download

#ifndef GLOBALVARIABLE\_H

#define GLOBALVARIABLE\_H

#include <QString>

enum SourceID { SP = 1, DT, GT };

typedef struct {

QString nType;

QString strSource;

int nRecevtTimestamp;

} Heartbeat;

#endif *// GLOBALVARIABLE\_H*

2. Common Properties (CommonProperties.h)

cpp

Copy

Download

#ifndef COMMONPROPERTIES\_H

#define COMMONPROPERTIES\_H

#include <QMap>

#include <QString>

class CommonProperties {

public:

static CommonProperties& instance();

QMap<QString, QString> getProperties() const;

void loadProperties(const QString& filename);

QString getIP(SourceID source) const;

int getPort(SourceID source) const;

private:

CommonProperties();

QMap<QString, QString> m\_properties;

QMap<SourceID, QString> m\_sourceNames;

};

#endif *// COMMONPROPERTIES\_H*

3. Common Properties Implementation (CommonProperties.cpp)

cpp

Copy

Download

#include "CommonProperties.h"

#include "GlobalVariable.h"

#include <QFile>

#include <QTextStream>

CommonProperties& CommonProperties::instance() {

static CommonProperties instance;

return instance;

}

CommonProperties::CommonProperties() {

m\_sourceNames[SP] = "SP";

m\_sourceNames[DT] = "DT";

m\_sourceNames[GT] = "GT";

}

QMap<QString, QString> CommonProperties::getProperties() const {

return m\_properties;

}

void CommonProperties::loadProperties(const QString& filename) {

QFile file(filename);

if (!file.open(QIODevice::ReadOnly | QIODevice::Text))

return;

m\_properties.clear();

QTextStream in(&file);

while (!in.atEnd()) {

QString line = in.readLine().trimmed();

if (line.startsWith("#") || line.isEmpty()) continue;

QStringList parts = line.split("=");

if (parts.size() == 2) {

m\_properties[parts[0].trimmed()] = parts[1].trimmed();

}

}

file.close();

}

QString CommonProperties::getIP(SourceID source) const {

QString key = m\_sourceNames.value(source) + ".IP";

return m\_properties.value(key, "127.0.0.1");

}

int CommonProperties::getPort(SourceID source) const {

QString key = m\_sourceNames.value(source) + ".Port";

return m\_properties.value(key, "0").toInt();

}

4. XML Processor (XMLProcessor.h)

cpp

Copy

Download

#ifndef XMLPROCESSOR\_H

#define XMLPROCESSOR\_H

#include <QString>

#include <QDomDocument>

#include "GlobalVariable.h"

class XMLProcessor {

public:

XMLProcessor();

QString generateXMLMessage(const Heartbeat& heartbeat);

bool validateXML(const QString& xml, const QString& xsdPath);

Heartbeat parseHeartbeatXML(const QString& xml);

private:

QString getXSDFilePath(SourceID source) const;

};

#endif *// XMLPROCESSOR\_H*

5. XML Processor Implementation (XMLProcessor.cpp)

cpp

Copy

Download

#include "XMLProcessor.h"

#include "CommonProperties.h"

#include <QFile>

#include <QXmlSchema>

#include <QXmlSchemaValidator>

XMLProcessor::XMLProcessor() {}

QString XMLProcessor::generateXMLMessage(const Heartbeat& heartbeat) {

QDomDocument doc;

QDomElement root = doc.createElement("Heartbeat");

doc.appendChild(root);

QDomElement type = doc.createElement("Type");

type.appendChild(doc.createTextNode(heartbeat.nType));

root.appendChild(type);

QDomElement source = doc.createElement("Source");

source.appendChild(doc.createTextNode(heartbeat.strSource));

root.appendChild(source);

QDomElement timestamp = doc.createElement("Timestamp");

timestamp.appendChild(doc.createTextNode(QString::number(heartbeat.nRecevtTimestamp)));

root.appendChild(timestamp);

return doc.toString();

}

bool XMLProcessor::validateXML(const QString& xml, const QString& xsdPath) {

QFile xsdFile(xsdPath);

if (!xsdFile.open(QIODevice::ReadOnly))

return false;

QXmlSchema schema;

if (!schema.load(&xsdFile, QUrl::fromLocalFile(xsdFile.fileName()))) {

xsdFile.close();

return false;

}

xsdFile.close();

QXmlSchemaValidator validator(schema);

return validator.validate(xml.toUtf8());

}

Heartbeat XMLProcessor::parseHeartbeatXML(const QString& xml) {

Heartbeat hb;

QDomDocument doc;

if (!doc.setContent(xml))

return hb;

QDomElement root = doc.documentElement();

hb.nType = root.firstChildElement("Type").text();

hb.strSource = root.firstChildElement("Source").text();

hb.nRecevtTimestamp = root.firstChildElement("Timestamp").text().toInt();

return hb;

}

QString XMLProcessor::getXSDFilePath(SourceID source) const {

switch(source) {

case SP: return ":/xsd/SPHeartbeat.xsd";

case DT: return ":/xsd/DTHeartbeat.xsd";

case GT: return ":/xsd/GTHeartbeat.xsd";

default: return "";

}

}

6. Heartbeat Message (HeartbeatMsg.h)

cpp

Copy

Download

#ifndef HEARTBEATMSG\_H

#define HEARTBEATMSG\_H

#include <QObject>

#include <QMap>

#include <QTableWidget>

#include "GlobalVariable.h"

class HeartbeatMsg : public QObject {

Q\_OBJECT

public:

explicit HeartbeatMsg(QObject \*parent = nullptr);

void messageReceived(const QString &messageReceived, const QString &messageSource);

void insertToTable(int nRow, const QString &strSource, const QString &strTimeStamp, bool status);

signals:

void sendStatus(QMap<int, bool>);

private:

QTableWidget \*m\_table;

bool validateMessage(const QString &message, SourceID source);

};

#endif *// HEARTBEATMSG\_H*

7. Heartbeat Message Implementation (HeartbeatMsg.cpp)

cpp

Copy

Download

#include "HeartbeatMsg.h"

#include "CommonProperties.h"

#include "XMLProcessor.h"

#include <QDateTime>

HeartbeatMsg::HeartbeatMsg(QObject \*parent) : QObject(parent), m\_table(nullptr) {}

void HeartbeatMsg::messageReceived(const QString &messageReceived, const QString &messageSource) {

QMap<int, bool> statusMap;

SourceID source = static\_cast<SourceID>(messageSource.toInt());

bool isValid = false;

if (source == SP) {

XMLProcessor processor;

isValid = processor.validateXML(messageReceived, ":/xsd/SPHeartbeat.xsd");

if (isValid) {

Heartbeat hb = processor.parseHeartbeatXML(messageReceived);

insertToTable(0, hb.strSource, QString::number(hb.nRecevtTimestamp), isValid);

}

} else {

*// Simple validation for DT and GT*

isValid = !messageReceived.isEmpty() && messageReceived.contains("HEARTBEAT");

insertToTable(source == DT ? 1 : 2,

messageSource,

QDateTime::currentDateTime().toString("hh:mm:ss.zzz"),

isValid);

}

statusMap[source] = isValid;

emit sendStatus(statusMap);

}

void HeartbeatMsg::insertToTable(int nRow, const QString &strSource, const QString &strTimeStamp, bool status) {

if (!m\_table) return;

if (nRow >= m\_table->rowCount())

m\_table->insertRow(nRow);

m\_table->setItem(nRow, 0, new QTableWidgetItem(strSource));

m\_table->setItem(nRow, 1, new QTableWidgetItem(strTimeStamp));

m\_table->setItem(nRow, 2, new QTableWidgetItem(status ? "Connected" : "Disconnected"));

QColor color = status ? Qt::green : Qt::red;

for (int col = 0; col < m\_table->columnCount(); ++col) {

m\_table->item(nRow, col)->setBackground(color);

}

}

bool HeartbeatMsg::validateMessage(const QString &message, SourceID source) {

if (source == SP) {

XMLProcessor processor;

return processor.validateXML(message, ":/xsd/SPHeartbeat.xsd");

}

return !message.isEmpty() && message.contains("HEARTBEAT");

}

8. TCP Server (TCPServer.h)

cpp

Copy

Download

#ifndef TCPSERVER\_H

#define TCPSERVER\_H

#include <QTcpServer>

#include <QTcpSocket>

#include <QThread>

#include <QMap>

class TCPServer : public QTcpServer {

Q\_OBJECT

public:

explicit TCPServer(QObject \*parent = nullptr);

void startServer(quint16 port);

signals:

void newMessage(QString, QString);

protected:

void incomingConnection(qintptr socketDescriptor) override;

private slots:

void onReadyRead();

void onDisconnected();

private:

QMap<QTcpSocket\*, QThread\*> m\_clients;

};

#endif *// TCPSERVER\_H*

9. TCP Server Implementation (TCPServer.cpp)

cpp

Copy

Download

#include "TCPServer.h"

#include <QDebug>

TCPServer::TCPServer(QObject \*parent) : QTcpServer(parent) {}

void TCPServer::startServer(quint16 port) {

if (!this->listen(QHostAddress::Any, port)) {

qDebug() << "Could not start server";

} else {

qDebug() << "Listening to port" << port << "...";

}

}

void TCPServer::incomingConnection(qintptr socketDescriptor) {

QThread \*thread = new QThread;

QTcpSocket \*socket = new QTcpSocket;

socket->setSocketDescriptor(socketDescriptor);

socket->moveToThread(thread);

connect(thread, &QThread::started, [socket]() {

qDebug() << "New client connected in thread" << QThread::currentThreadId();

});

connect(socket, &QTcpSocket::readyRead, this, &TCPServer::onReadyRead);

connect(socket, &QTcpSocket::disconnected, this, &TCPServer::onDisconnected);

connect(socket, &QTcpSocket::disconnected, thread, &QThread::quit);

connect(thread, &QThread::finished, thread, &QThread::deleteLater);

m\_clients.insert(socket, thread);

thread->start();

}

void TCPServer::onReadyRead() {

QTcpSocket \*socket = qobject\_cast<QTcpSocket\*>(sender());

if (!socket) return;

QByteArray data = socket->readAll();

QString message = QString::fromUtf8(data);

QString source = QString::number(m\_clients.key(socket));

emit newMessage(message, source);

}

void TCPServer::onDisconnected() {

QTcpSocket \*socket = qobject\_cast<QTcpSocket\*>(sender());

if (socket) {

m\_clients.remove(socket);

socket->deleteLater();

}

}

10. TCP Client (TCPClient.h)

cpp

Copy

Download

#ifndef TCPCLIENT\_H

#define TCPCLIENT\_H

#include <QTcpSocket>

#include <QThread>

#include <QTimer>

class TCPClient : public QObject {

Q\_OBJECT

public:

explicit TCPClient(SourceID source, QObject \*parent = nullptr);

void connectToServer();

signals:

void newMessage(QString, QString);

private slots:

void onConnected();

void onReadyRead();

void onDisconnected();

void sendHeartbeat();

private:

SourceID m\_source;

QTcpSocket \*m\_socket;

QThread \*m\_thread;

QTimer \*m\_heartbeatTimer;

QString m\_serverIP;

int m\_serverPort;

};

#endif *// TCPCLIENT\_H*

11. TCP Client Implementation (TCPClient.cpp)

cpp

Copy

Download

#include "TCPClient.h"

#include "CommonProperties.h"

#include "XMLProcessor.h"

#include <QDebug>

TCPClient::TCPClient(SourceID source, QObject \*parent)

: QObject(parent), m\_source(source), m\_socket(nullptr), m\_thread(nullptr), m\_heartbeatTimer(nullptr) {

CommonProperties &props = CommonProperties::instance();

m\_serverIP = props.getIP(source);

m\_serverPort = props.getPort(source);

m\_thread = new QThread;

this->moveToThread(m\_thread);

m\_socket = new QTcpSocket;

m\_socket->moveToThread(m\_thread);

connect(m\_thread, &QThread::started, this, &TCPClient::connectToServer);

connect(m\_socket, &QTcpSocket::connected, this, &TCPClient::onConnected);

connect(m\_socket, &QTcpSocket::readyRead, this, &TCPClient::onReadyRead);

connect(m\_socket, &QTcpSocket::disconnected, this, &TCPClient::onDisconnected);

m\_heartbeatTimer = new QTimer;

m\_heartbeatTimer->moveToThread(m\_thread);

connect(m\_heartbeatTimer, &QTimer::timeout, this, &TCPClient::sendHeartbeat);

m\_thread->start();

}

void TCPClient::connectToServer() {

m\_socket->connectToHost(m\_serverIP, m\_serverPort);

}

void TCPClient::onConnected() {

qDebug() << "Connected to server" << m\_serverIP << ":" << m\_serverPort;

m\_heartbeatTimer->start(5000); *// Send heartbeat every 5 seconds*

}

void TCPClient::onReadyRead() {

QByteArray data = m\_socket->readAll();

QString message = QString::fromUtf8(data);

emit newMessage(message, QString::number(m\_source));

}

void TCPClient::onDisconnected() {

qDebug() << "Disconnected from server";

m\_heartbeatTimer->stop();

QTimer::singleShot(3000, this, &TCPClient::connectToServer); *// Reconnect after 3 seconds*

}

void TCPClient::sendHeartbeat() {

if (m\_source == SP) {

XMLProcessor processor;

Heartbeat hb;

hb.nType = "HEARTBEAT";

hb.strSource = QString::number(m\_source);

hb.nRecevtTimestamp = QDateTime::currentDateTime().toSecsSinceEpoch();

QString xml = processor.generateXMLMessage(hb);

m\_socket->write(xml.toUtf8());

} else {

QString message = QString("HEARTBEAT|%1|%2")

.arg(QString::number(m\_source))

.arg(QDateTime::currentDateTime().toString("yyyyMMddhhmmss"));

m\_socket->write(message.toUtf8());

}

}

12. Control Status (ControlStatus.h)

cpp

Copy

Download

#ifndef CONTROLSTATUS\_H

#define CONTROLSTATUS\_H

#include <QWidget>

#include <QMap>

#include "GlobalVariable.h"

namespace Ui {

class ControlStatus;

}

class ControlStatus : public QWidget {

Q\_OBJECT

public:

explicit ControlStatus(QWidget \*parent = nullptr);

~ControlStatus();

public slots:

void controlStatusUI(QMap<int, bool> qmapConnectionStatus);

private:

Ui::ControlStatus \*ui;

QMap<int, bool> m\_mapListOfApp;

bool m\_spConn, m\_dpConn, m\_gtConn;

};

#endif *// CONTROLSTATUS\_H*

13. Control Status Implementation (ControlStatus.cpp)

cpp

Copy

Download

#include "ControlStatus.h"

#include "ui\_ControlStatus.h"

ControlStatus::ControlStatus(QWidget \*parent)

: QWidget(parent), ui(new Ui::ControlStatus), m\_spConn(false), m\_dpConn(false), m\_gtConn(false) {

ui->setupUi(this);

ui->labelSP->setStyleSheet("background-color: red;");

ui->labelDT->setStyleSheet("background-color: red;");

ui->labelGT->setStyleSheet("background-color: red;");

}

ControlStatus::~ControlStatus() {

delete ui;

}

void ControlStatus::controlStatusUI(QMap<int, bool> qmapConnectionStatus) {

ui->labelSP->setStyleSheet("background-color: red;");

ui->labelDT->setStyleSheet("background-color: red;");

ui->labelGT->setStyleSheet("background-color: red;");

m\_mapListOfApp.clear();

m\_mapListOfApp = qmapConnectionStatus;

QMapIterator<int, bool> iterator(qmapConnectionStatus);

while (iterator.hasNext()) {

iterator.next();

switch (iterator.key()) {

case SP:

m\_spConn = iterator.value();

ui->labelSP->setStyleSheet(m\_spConn ? "background-color: green;" : "background-color: red;");

break;

case DT:

m\_dpConn = iterator.value();

ui->labelDT->setStyleSheet(m\_dpConn ? "background-color: green;" : "background-color: red;");

break;

case GT:

m\_gtConn = iterator.value();

ui->labelGT->setStyleSheet(m\_gtConn ? "background-color: green;" : "background-color: red;");

break;

}

}

}

14. Main Window (MainWindow.h)

cpp

Copy

Download

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "TCPServer.h"

#include "TCPClient.h"

#include "HeartbeatMsg.h"

#include "ControlStatus.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

Ui::MainWindow \*ui;

TCPServer \*m\_server;

QMap<SourceID, TCPClient\*> m\_clients;

HeartbeatMsg \*m\_heartbeatMsg;

ControlStatus \*m\_controlStatus;

};

#endif *// MAINWINDOW\_H*

15. Main Window Implementation (MainWindow.cpp)

cpp

Copy

Download

#include "MainWindow.h"

#include "ui\_MainWindow.h"

#include "CommonProperties.h"

#include <QMessageBox>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), ui(new Ui::MainWindow), m\_server(nullptr), m\_heartbeatMsg(nullptr) {

ui->setupUi(this);

*// Load properties*

CommonProperties::instance().loadProperties("DCS.properties");

*// Initialize server*

m\_server = new TCPServer(this);

m\_server->startServer(CommonProperties::instance().getPort(SP));

*// Initialize heartbeat message processor*

m\_heartbeatMsg = new HeartbeatMsg(this);

m\_heartbeatMsg->setTable(ui->tableWidget);

*// Initialize control status UI*

m\_controlStatus = new ControlStatus(this);

ui->statusLayout->addWidget(m\_controlStatus);

*// Connect signals and slots*

connect(m\_server, &TCPServer::newMessage, m\_heartbeatMsg, &HeartbeatMsg::messageReceived);

connect(m\_heartbeatMsg, &HeartbeatMsg::sendStatus, m\_controlStatus, &ControlStatus::controlStatusUI);

*// Initialize clients*

m\_clients[SP] = new TCPClient(SP, this);

m\_clients[DT] = new TCPClient(DT, this);

m\_clients[GT] = new TCPClient(GT, this);

connect(m\_clients[SP], &TCPClient::newMessage, m\_heartbeatMsg, &HeartbeatMsg::messageReceived);

connect(m\_clients[DT], &TCPClient::newMessage, m\_heartbeatMsg, &HeartbeatMsg::messageReceived);

connect(m\_clients[GT], &TCPClient::newMessage, m\_heartbeatMsg, &HeartbeatMsg::messageReceived);

}

MainWindow::~MainWindow() {

delete ui;

if (m\_server) m\_server->close();

qDeleteAll(m\_clients);

}

16. Server Simulator (ServerSimulator.h)

cpp

Copy

Download

#ifndef SERVERSIMULATOR\_H

#define SERVERSIMULATOR\_H

#include <QTcpServer>

#include <QTcpSocket>

#include <QTimer>

class ServerSimulator : public QObject {

Q\_OBJECT

public:

explicit ServerSimulator(quint16 port, QObject \*parent = nullptr);

void start();

private slots:

void newConnection();

void readyRead();

void sendSimulatedData();

private:

QTcpServer \*m\_server;

QList<QTcpSocket\*> m\_clients;

QTimer \*m\_simulationTimer;

quint16 m\_port;

};

#endif *// SERVERSIMULATOR\_H*

17. Server Simulator Implementation (ServerSimulator.cpp)

cpp

Copy

Download

#include "ServerSimulator.h"

#include <QDateTime>

#include <QDebug>

ServerSimulator::ServerSimulator(quint16 port, QObject \*parent)

: QObject(parent), m\_server(new QTcpServer(this)), m\_simulationTimer(new QTimer(this)), m\_port(port) {

connect(m\_server, &QTcpServer::newConnection, this, &ServerSimulator::newConnection);

connect(m\_simulationTimer, &QTimer::timeout, this, &ServerSimulator::sendSimulatedData);

}

void ServerSimulator::start() {

if (!m\_server->listen(QHostAddress::Any, m\_port)) {

qDebug() << "Server could not start on port" << m\_port;

} else {

qDebug() << "Simulator server listening on port" << m\_port;

m\_simulationTimer->start(3000); *// Send data every 3 seconds*

}

}

void ServerSimulator::newConnection() {

QTcpSocket \*socket = m\_server->nextPendingConnection();

connect(socket, &QTcpSocket::readyRead, this, &ServerSimulator::readyRead);

connect(socket, &QTcpSocket::disconnected, [this, socket]() {

m\_clients.removeOne(socket);

socket->deleteLater();

});

m\_clients.append(socket);

qDebug() << "New client connected to simulator";

}

void ServerSimulator::readyRead() {

QTcpSocket \*socket = qobject\_cast<QTcpSocket\*>(sender());

if (!socket) return;

QByteArray data = socket->readAll();

qDebug() << "Simulator received:" << data;

}

void ServerSimulator::sendSimulatedData() {

if (m\_clients.isEmpty()) return;

QString message;

if (m\_port == 1234) { *// SP server*

message = QString(

"<Heartbeat>"

"<Type>HEARTBEAT</Type>"

"<Source>SP</Source>"

"<Timestamp>%1</Timestamp>"

"</Heartbeat>"

).arg(QDateTime::currentDateTime().toSecsSinceEpoch());

} else {

message = QString("HEARTBEAT|%1|%2")

.arg(m\_port == 1235 ? "DT" : "GT")

.arg(QDateTime::currentDateTime().toString("yyyyMMddhhmmss"));

}

for (QTcpSocket \*client : m\_clients) {

client->write(message.toUtf8());

}

}

18. Sample DCS.properties

text

Copy

Download

# Server configuration

SP.IP=127.0.0.1

SP.Port=1234

DT.IP=127.0.0.1

DT.Port=1235

GT.IP=127.0.0.1

GT.Port=1236

# Other properties

Heartbeat.Interval=5000

Timeout=30000

19. Sample SPHeartbeat.xsd

xml

Copy

Download

Run

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="Heartbeat">

<xs:complexType>

<xs:sequence>

<xs:element name="Type" type="xs:string"/>

<xs:element name="Source" type="xs:string"/>

<xs:element name="Timestamp" type="xs:integer"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

How to Use

1. Create a Qt project with all these files
2. Add the XSD files to your resources (create an XSD folder in your resources)
3. Create the DCS.properties file in your build directory
4. Start the main application
5. Optionally, start the server simulators for testing:

cpp

Copy

Download

*// To test with simulators*

ServerSimulator spSimulator(1234);

ServerSimulator dtSimulator(1235);

ServerSimulator gtSimulator(1236);

spSimulator.start();

dtSimulator.start();

gtSimulator.start();

This implementation provides:

* A multithreaded TCP server that can handle multiple clients
* Three TCP clients (SP, DT, GT) that connect to their respective servers
* XML message processing for SP with XSD validation
* Simple message processing for DT and GT
* Status monitoring with color-coded UI indicators
* Heartbeat mechanism to monitor connection status
* Configuration through properties file

The system will show green indicators when connections are active and receiving valid heartbeats, and red when connections are lost or invalid messages are received.