ExpertElectronics ESDR3 integration series

Design options for 3rd party developers

TCI skimmer Lite integration with LOG

The purpose of this document is to describe in general how integration can be done between TCI skimmer (version Lite) and any integration capable 3rd party Ham Radio Log.

VK6NX

31 January 2023

Document Control

Version	Author	Comment	Date
0.1	VK6NX	Initial Draft	31 Jan 2023

Contents

1 Introduction

1.1 Document Scope

This document describes the option to integration between TCI skimmer Lite and any integration capable Ham radio Log.

This document is a living document and will need to be updated and validated once TCI skimmer Lite project moves from version to version.

1.2 Document Audience

Any interested 3rd party developer.

1.3 Integration prerequisites

Log must meet the following prerequisites, before proceeding integration:

1. Log must support TCI integration with ESDR3.
2. Log must have implemented separate Websockets connection to TCI skimmer.

1.4 Context

The integration methods described in this document should be taken within context of overall integration between Expert Electronics ESDR3-based (strictly) products and 3rd party software.

Integration options mentioned in this document are directly related to **TCI skimmer Lite** (later on - with **TCI skimmer**, once it is ready), **ESDR3** (only) and any Ham Radio Log, which meets above listed prerequisites.

2 Integration Overview

2.1 Preface

Term “integration” within the context of this document means adding on capabilities of exchanging bi-directional information between TCI skimmer Lite (“Skimmer” in the document below) and Ham radio Log (“Log” in the document below).

2.2 Log to Skimmer optional data exchange.

After integration is completed, Log will be able to send following data to Skimmer:

Configuration of “Skimmer mode” (options: "common", "dxpedition", "contest")

Configuration option of using Callsign (options: "on", “off”)

Configuration option of using RSTR (options: "on", “off”)

Configuration option of using SRX (options: "on", “off”)

Configuration option of using SRX_INFO (options: "on", “off”)

Configuration option of using Stream (options: "on", “off”)

For the environments when there is no implementation for the Log to Skimer data exchange, the following “default” Skimmer settings are in place:

Mode: "common"

Use RSTR: “off”

Use SRX: “off”

Use SRX_INFO: “off”

Use Stream: “off”

2.3 Skimmer to Log data exchange.

As minimum the Skimmer is capable to send decoded Call Sign to the log.

Additionally, as of current Skimmer version, it can send the decoding frequency associated with decoded Call Sign. However this data can also be obtained via TCI communication between Log and ESDR3, hence it is kind of “duplicate”, leaving 3rd party developer to select an integration option, which is more comfortable for its project.

Once additional functionality will be implemented in Skimmer, it will be capable to reflect the RSTR, SRX, SRX_INFO and Stream options (described in the above section) via sending relevant data to the Log.

2.4 Integration framework.

Integration examples provided within this document are based on Qt6.4 / Qt6.5, using both C++ libraries and QML.

3 Integration Approach

3.1 Skimmer side capabilities

Skimmer runs two streams of integration:

1. In-build Native EE's C++ TCI library, running as the TCI client, which interconnects with ESDR3 via Websockets (default) port 50001. This integration stream provides data exchange between ESDR3 and Skimmer.
2. In-build Websockets Server (Qt C++), running as the server via Websockets (default) port 42042. This integration stream enables data exchange between Skimmer and Log.

3.2 Dependencies

To be able to integration with Skimmer, the Log must run separate Websockets stream towards ws://localhost:42042 (assuming Log runs on the same PC with Skimmer) or to ws://ip_address:42042 (assuming log runs on separate PC apart from Skimmer).

Please see the example below, based on author's Log actual integration.

3.3 Caveats

RSTR, SRX, SRX_INFO and Stream options are not enabled as yet with current Skimmer development version.

3.4 Risks

No risks found at this stage of development.

3.5 Issues

No issues found at this stage of development.

4 Integration Examples

4.1 Log side

The following example is provided based on OClog v1.3.1 update 4

4.2 Log .pro file

```
QT += quick qml core quickcontrols2 websockets
CONFIG += c++latest

SOURCES += \
    main.cpp \
    ./staticLibs/wsskimmerclient/wsskimmerclient.cpp \
HEADERS += \
    ./staticLibs/wsskimmerclient/wsskimmerclient.h \
```

4.3 Log main.cpp file

```
#include "staticLibs/wsskimmerclient/wsskimmerclient.h"
int main(int argc, char *argv[])
{
   qmlRegisterType<WSskimmerClient>("WsSkimmerClient", 1, 0, "WsSkimmerClient");
}
```

4.4 Log wsskimmerclient.h file

```
#ifndef WSSKIMMERCLIENT_H
#define WSSKIMMERCLIENT_H

#include <QObject>
#include <QtWebSockets/QWebSocket>
#include <QDebug>
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>

class WSskimmerClient : public QObject
{
    Q_OBJECT
public:
    explicit WSskimmerClient(QObject *parent = nullptr);

signals:

public slots:
    Q_INVOKABLE void connectToHost(QString host);
    Q_INVOKABLE void disconnectFromHost();
    Q_INVOKABLE void onConnected();
    Q_INVOKABLE void onDisconnected();
    Q_INVOKABLE void onStateChanged(QAbstractSocket::SocketState socketState);
    Q_INVOKABLE void sendConfigToSkimmer(QString &msg);
```

```
Q_SIGNALS:
    void wsTciConnected(const QString &msg);
    void wsTciDisonnected(const QString &msg);
    void wsSocketState(QAbstractSocket::SocketState socketState);
    void wsMessageReceived(const QVariantMap &msg);
    void wsJsonMessageReceived(const QJsonDocument &msg);

    void wsSkimmerMessageReceived (const QString &msg);
    void skimmerDecodedCallSign(const QString &msg);
    void configConfirmationFromSkimmer(const QString &msg);
private slots:

    Q_INVOKABLE void onMessageReceived(QString message);

private:
    QWebSocket web_socket_;
};

#endif // WSSKIMMERCLIENT_H
```

4.5 Log wsskimmerclient.cpp file

```
#include "wsskimmerclient.h"

WSskimmerClient::WSskimmerClient(QObject *parent)
    : QObject{parent}
{
    connect(&web_socket_, &QWebSocket::connected, this,
&WSskimmerClient::onConnected);
    connect(&web_socket_, &QWebSocket::disconnected, this,
&WSskimmerClient::onDisconnected);
    connect(&web_socket_, &QWebSocket::textMessageReceived, this,
&WSskimmerClient::onMessageReceived);
    connect(&web_socket_, &QWebSocket::stateChanged, this,
&WSskimmerClient::onStateChanged);
}

void WSskimmerClient::connectToHost(QString host)
{
    if(web_socket_.state() == QAbstractSocket::ConnectedState){
        disconnectFromHost();
    }
    else{
        qInfo() << "Connecting to: " << host << " (msg from C++ WSskimmerClient.cpp)";
        web_socket_.open(QUrl(host));
    }
}

void WSskimmerClient::disconnectFromHost()
{
    web_socket_.close();
}
```

```
void WSskimmerClient::onStateChanged(QAbstractSocket::SocketState socketState)
{
    emit wsSocketState(socketState);
}

void WSskimmerClient::sendConfigToSkimmer(QString &msg)
{
    web_socket_.sendTextMessage(msg);
}

void WSskimmerClient::onMessageReceived(QString message)
{
    emit wsSkimmerMessageReceived(message); // for QML in-band logview

    QByteArray jsonSkimmerData = message.toUtf8();
    QJsonDocument jsonSkimmerDocument =
QJsonDocument::fromJson(jsonSkimmerData);
    QJsonObject object = jsonSkimmerDocument.object();
    QJsonValue callsign = object.value("callsign");
    if (callsign.toString() != ""){
        emit skimmerDecodedCallSign(callsign.toString());
    }
    QJsonValue config = object.value("config");
    if (config.toString() != ""){
        emit configConfirmationFromSkimmer(config.toString());
    }
}
```

4.6 Log main.qml file

```
import WsSkimmerClient 1.0
ApplicationWindow {
property string _skimmerSocketState
property string wsSkimmerDecodedCallSign: ""
property string wsSkimmerConfigConfirm: ""

WsSkimmerClient {
id: skimmerWSclient
  onWsSocketState: (data) => {_skimmerSocketState = data;}
  onSkimmerDecodedCallSign: (data) =>{wsSkimmerDecodedCallSign = data;}
  onConfigConfirmationFromSkimmer: (data) =>{wsSkimmerConfigConfirm = data; }
            }
}
```

4.7 Skimmer side

The following parts are provided on “for your reference” basis.

The following part of websockerserver.cpp related to receiving config from the Log:

```
void WSocketsServer::processTextMessage(const QString message) {
    emit wsConfigMessageReceived(message);
    QWebSocket *pClient = qobject_cast<QWebSocket *>(sender());

    QByteArray jsoncConfigData = message.toUtf8();
    QJsonDocument jsonConfigDocument = QJsonDocument::fromJson(jsoncConfigData);
    if(jsonConfigDocument.isObject() == false) {
        if (pClient) {
            pClient->sendTextMessage("Message received by Skimmer:" + message);
        }
    }
    if(jsonConfigDocument.isObject() == true) {
        QJsonObject object = jsonConfigDocument.object();
        QJsonValue mode = object.value("mode");
        qDebug() << "C++ JSON Skimmer mode: " << mode.toString();
        QJsonArray array = object["modeconfig"].toArray();
        QJsonObject ar0 = array.at(0).toObject();
        QJsonObject ar1 = array.at(1).toObject();
        QJsonObject ar2 = array.at(2).toObject();
        QJsonObject ar3 = array.at(3).toObject();
        QJsonObject ar4 = array.at(4).toObject();
        emit skimmerMode(mode.toString());
        emit useCallsign(ar0["callsign"].toString());
        emit useRSTR(ar1["rstr"].toString());
        emit useSRX(ar2["srx"].toString());
        emit useSRXinfo(ar3["srxinfo"].toString());
        emit useStream(ar4["stream"].toString());
        qDebug() << "C++ JSON Skimmer use Callsign: " << ar0["callsign"].toString();
        qDebug() << "C++ JSON Skimmer use RSTR: " << ar1["rstr"].toString();
        qDebug() << "C++ JSON Skimmer use SRX: " << ar2["srx"].toString();
        qDebug() << "C++ JSON Skimmer use SRX_INFO: " << ar3["srxinfo"].toString();
        qDebug() << "C++ JSON Skimmer use Stream: " << ar4["stream"].toString();

        if (pClient) {
            pClient->sendTextMessage("{\"config\":\"received\"}");
        }
    }
}
```

The following part of websockerserver.cpp is related to sending decoded Call Sign to Log:

```
void WSocketsServer::sendCallSign(const QString &message) {
    for (QWebSocket *pClient : qAsConst(m_clients)) {
            pClient->sendTextMessage(message);
    }
    qInfo() << "C++ Skimmer - Callsign sent:" << message;
}
```

The following part of main.qml are providing the overview of QML integration of the Websockets server in relation to Log data exchange.

The following part is related to receiving config from Log:

```
property string _skimmerMode: ""
WSocketsServer {
    id: websocketServer

    onSkimmerMode: (data) => {
                       _skimmerMode = data;
                   }
}
```

The following part is related to pushing decoded Call Sign to Log:

```
property string decodedCallSign: "callsign"
function sendTestCallsignToLog(){
    websocketServer.sendCallSign(JSON.stringify({"callsign":decodedCallSign}))
}
```