EIPScanner Documentation

Release 1.0.0

Aleksey Timin, Adam Roth

Jun 22, 2023

CONTENTS:

CHAPTER

ONE

ABOUT

EIPScanner is a free implementation of Ethernet/IP scanner in C++.

1.1 Getting Started

1.1.1 Installing

EIPScanner provides only installing from the sources. To compile the sources, your system must meet the following requirements:

- Linux or MacOS operation system
- CMake 3.5 or higher
- C++ compiler supporting C++20 standard
- Gtest 1.8.0 or higher (optional)

In order to compile and install the library, type from the project's root directory:

```
mkdir build
cd build
cmake ..
cmake --build . --target install
```

Optionally, you can build the usage examples and the unit tests by adding the following CMake options:

```
cmake -DTEST_ENABLED=ON -DEXAMPLE_ENABLED=ON ..
```

For successful usage of the library, it will be very helpful if you remember where **EIPScanner** have been installed.

1.1.2 Usage

Here we will show how you can use the library in your CMake project. For that, let's make a simple project.

First of all, we should create *CMakeLists.txt* with the following content:

```
cmake_minimum_required(VERSION 3.5)
project(hi_eip)

set(CMAKE_CXX_STANDARD 14)

add_executable(hi_eip main.cpp)
```

(continues on next page)

(continued from previous page)

```
include_directories(/usr/local/include/EIPScanner)
target_link_libraries(hi_eip EIPScanner)
```

Pay attention to the last two lines. Currently, **EIPScanner** doesn't provide a cmake module to help to find the library on your machine and we have to do all manually. First, we point on the include directory whose path should be *path/were/eipscanner/is/installed/ + EIPScanner*. Second, we link our executable file with the library *EIPScanner*. If you'd like to use the static library instead, use *EIPScannerS* name.

Okay, we have *CMakeLists.txt*. Now we should create *main.cpp* and place there this code:

```
#include <EIPScanner/MessageRouter.h>
#include <EIPScanner/utils/Logger.h>
#include <EIPScanner/utils/Buffer.h>

using eipScanner::SessionInfo;
using eipScanner::MessageRouter;
using namespace eipScanner::cip;
using namespace eipScanner::utils;

int main() {
        Logger::setLogLevel(LogLevel::DEBUG);
        auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);
        auto messageRouter = std::make_shared<MessageRouter>();

        // Read attribute
        auto response = messageRouter->sendRequest(si, ServiceCodes::GET_ATTRIBUTE_
↪SINGLE,
                                                   EPath(0x01, 1, 1));

        if (response.getGeneralStatusCode() == GeneralStatusCodes::SUCCESS) {
                Buffer buffer(response.getData());
                CipUint vendorId;
                buffer >> vendorId;

                Logger(LogLevel::INFO) << "Vendor ID is " << vendorId;
        }

        return 0;
}
```

If you are familiar with **EtherNet/IP** protocol you should understand that the code is doing. If not, it doesn't matter, we will discuss this later.

Let's build the example and run it:

```
mkdir build && cd build
cmake ..
./hi_eip
```

It might happen you become the error:

```
libEIPScanner.so.1: cannot open shared object file: No such file or directory
```

It means, your host system didn't manage to find **EIPScanner**'s shared library. We can help it:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/where/eipscanner/is/installed
```

Try again and it must work. If you failed again, then a bug report will be welcomed.

1.2 Explicit Messaging

EtherNet/IP protocol provides the explicit messaging that is a RPC protocol working via TCPIP. To make a "call" we have to know the code of the service and the address of the instance or the class that provides the wanted service.

CIP protocol uses *EPATH* to address objects in the object model. It contains ClassID of the object, InstanceID of the instance and, optionally, AttributeID of the instance attribute. If we want to call a class service we should use InstanceID=0.

Let's say we have a *Widget Object* with ClassID=0xf0 and this object has service *PlusOne* with code 0x1 which receives a integer as an argument and returns its increment. The service belongs to instance level, so we have some instance of *Widget Object* with InstanceID=0x5.

So we have this request:

	Type	Value
Service Code	USINT	0x1
Address	EPATH	ClassID=0xf0, InstanceID=0x5
Argument	INT	5

And the response should be:

	Type	Value
Service Code	USINT	0x81 (response has code = service code | 0x80)
General Status	USINT	0 (SUCCESS)
Result	INT	6

1.2.1 Message Router

But whom should we send the request? In any EIP device, there is a special object for this. It is *Message Router*. The router is responsible for receiving explicit requests, routing them to **CIP** objects and handling errors and results.

This sounds a bit abstract. Let's see how it will be implemented in code:

```
#include <EIPScanner/MessageRouter.h>
#include <EIPScanner/utils/Logger.h>
#include <EIPScanner/utils/Buffer.h>

using eipScanner::SessionInfo;
using eipScanner::MessageRouter;
using namespace eipScanner::cip;
using namespace eipScanner::utils;

int main() {
        try {
                // Open EIP session with the adapter
                auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);
```

(continues on next page)

(continued from previous page)

```
                // Send Message Router Request
                MessageRouter messageRouter;

                const CipUsint PLUS_ON_SERVICE = 0x05;
                const EPath EPATH_TO_WIDGET_INSTANCE(0xf0, 0x5);
                Buffer buffer;
                CipInt arg = 5;
                buffer << arg;

                auto response = messageRouter.sendRequest(si,
                                        PLUS_ON_SERVICE,
                                        EPATH_TO_WIDGET_INSTANCE,
                                        buffer.data());

                if (response.getGeneralStatusCode() == GeneralStatusCodes::SUCCESS) {
                        Buffer buffer(response.getData());
                        CipInt result;
                        buffer >> result;

                        Logger(LogLevel::INFO) << result;
                }
        } catch (std::exception &e) {
                Logger(LogLevel::ERROR) << e.what();
                return -1;
        }

        return 0;
}
```

First of all, we have to connect with the EIP adapter and establish EIP session with it. We do it by using *SessionInfo* object:

```
auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);
```

Here we're connecting with the EIP adapter by IP address 172.28.1.3 and port 0xAF12 (default port for the explicit messaging).

Then we should form and send a request to the *Message Router*:

```
MessageRouter messageRouter;

const CipUsint PLUS_ON_SERVICE = 0x05;
const EPath EPATH_TO_WIDGET_INSTANCE(0xf0, 0x5);
Buffer buffer;
CipInt arg = 5;
buffer << arg;

auto response = messageRouter.sendRequest(si,
                     PLUS_ON_SERVICE,
                     EPATH_TO_WIDGET_INSTANCE,
                     buffer.data());
```

Unfortunately, we can't send the service arguments "as is". Instead we should encode them and decode the result according **CIP** specification. To make it easer, **EIPScanner** provides a special class *utils::Buffer*. In this example, we encode 5 as INT type and get the result as a byte vector with method *utils::Buffer::data()*.

The result of the request is stroed in *Message Router Response*:

```
if (response.getGeneralStatusCode() == GeneralStatusCodes::SUCCESS) {
      Buffer buffer(response.getData());
      CipInt result;
      buffer >> result;

      Logger(LogLevel::INFO) << result;
 }
```

First of all, we should check if the request is successful and only then we decode its data by using *Buffer*. In our example, this is only one number of type INT.

1.2.2 Reading\Writing CIP Attributes

The most typical operations in the explicit communication are reading and writing **CIP** attributes. The example that we used above is suitable, but we should keep in mind 2 things:

1. Use *cip::Epath* with *Attribute ID* which you're going to read or write an attribute. For an example EPath(1,2,3), where ClassId=1, InstanceId=2, AttributeId=3
2. Use *cip::ServiceCodes* enum with the common service codes

1.3 Implicit Messaging

EtherNet/IP protocol allows to connect the scanner and the adapter by using IO connections, so that they can send data each other periodically or then the data have changed.

In order to establish and handle IO connections through UDP, **EIPScanner** provides *ConnectionManager* class, that has *forwardOpen* method where we can pass all the parameters of the connections. The method returns an instance of *IOConnection* class, which we can use to handle the received data from the adapter and send the our data to it.

```
#include <sstream>
#include <cip/connectionManager/NetworkConnectionParams.h>
#include "SessionInfo.h"
#include "ConnectionManager.h"
#include "utils/Logger.h"
#include "utils/Buffer.h"

using namespace eipScanner::cip;
using eipScanner::SessionInfo;
using eipScanner::MessageRouter;
using eipScanner::ConnectionManager;
using eipScanner::cip::connectionManager::ConnectionParameters;
using eipScanner::cip::connectionManager::NetworkConnectionParams;
using eipScanner::utils::Buffer;
using eipScanner::utils::Logger;
using eipScanner::utils::LogLevel;

int main() {
        Logger::setLogLevel(LogLevel::DEBUG);
        auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);

        // Implicit messaging
        ConnectionManager connectionManager;
```

(continues on next page)

(continued from previous page)

```
        ConnectionParameters parameters;
        parameters.connectionPath = {0x20, 0x04,0x24, 151, 0x2C, 150, 0x2C, 100};  //␣
↪config Assm151, output Assm150, intput Assm100
        parameters.o2tRealTimeFormat = true;
        parameters.originatorVendorId = 342;
        parameters.originatorSerialNumber = 32423;
        parameters.t2oNetworkConnectionParams |= NetworkConnectionParams::P2P;
        parameters.t2oNetworkConnectionParams |= NetworkConnectionParams::SCHEDULED_
↪PRIORITY;
        parameters.t2oNetworkConnectionParams |= 32; //size of Assm100 =32
        parameters.o2tNetworkConnectionParams |= NetworkConnectionParams::P2P;
        parameters.o2tNetworkConnectionParams |= NetworkConnectionParams::SCHEDULED_
↪PRIORITY;
        parameters.o2tNetworkConnectionParams |= 32; //size of Assm150 = 32

        parameters.originatorSerialNumber = 0x12345;
        parameters.o2tRPI = 1000000;
        parameters.t2oRPI = 1000000;
        parameters.transportTypeTrigger |= NetworkConnectionParams::CLASS1;

        auto io = connectionManager.forwardOpen(si, parameters);
        if (auto ptr = io.lock()) {
                ptr->setDataToSend(std::vector<uint8_t>(32));

                ptr->setReceiveDataListener([](auto realTimeHeader, auto sequence,␣
↪auto data) {
                        std::ostringstream ss;
                        ss << "secNum=" << sequence << " data=";
                        for (auto &byte : data) {
                                ss << "[" << std::hex << (int) byte << "]";
                        }

                        Logger(LogLevel::INFO) << "Received: " << ss.str();
                });

                ptr->setCloseListener([]() {
                        Logger(LogLevel::INFO) << "Closed";
                });
        }

        int count = 200;
        while (connectionManager.hasOpenConnections() && count-- > 0) {
                connectionManager.handleConnections(std::chrono::milliseconds(100));
        }

        connectionManager.forwardClose(si, io);

        return 0;
}
```

There are many lines of code here. But let's go through it step by step.

```
auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);
```

ConnectionManager uses *Explicit Messaging*, so we need to establish **EIP** session before to call service *Forward_Open*:

```
ConnectionManager connectionManager;

ConnectionParameters parameters;
parameters.connectionPath = {0x20, 0x04,0x24, 151, 0x2C, 150, 0x2C, 100};  // config␣
↪Assm151, output Assm150, intput Assm100
parameters.o2tRealTimeFormat = true;
parameters.originatorVendorId = 342;
parameters.originatorSerialNumber = 32423;
parameters.t2oNetworkConnectionParams |= NetworkConnectionParams::P2P;
parameters.t2oNetworkConnectionParams |= NetworkConnectionParams::SCHEDULED_PRIORITY;
parameters.t2oNetworkConnectionParams |= 32; //size of Assm100 =32
parameters.o2tNetworkConnectionParams |= NetworkConnectionParams::P2P;
parameters.o2tNetworkConnectionParams |= NetworkConnectionParams::SCHEDULED_PRIORITY;
parameters.o2tNetworkConnectionParams |= 32; //size of Assm150 = 32

parameters.originatorSerialNumber = 0x12345;
parameters.o2tRPI = 1000000;
parameters.t2oRPI = 1000000;
parameters.transportTypeTrigger |= NetworkConnectionParams::CLASS1;

auto io = connectionManager.forwardOpen(si, parameters);
```

As you can see, IO connection has a lot of parameters. This tutorial doesn't aim to give the whole information about all the options and parameters of the implicit messaging. Use please **CIP** specification for details. Moreover each **EIP** device can have its own set of parameters which it uses to establish the IO connection. Always see documentation or\and EDS files to figure out how to tune the parameters.

However, there are some things that need clarifying:

1. Service *Forward_Open* opens two connections: Originator (Scanner) => Target (Adapter) and Target => Originator. Parameters that start with **o2t** defined for direction Originator => Target, **t2o** for Originator => Target.
2. *t2oNetworkConnectionParams* and *t2oNetworkConnectionParams* has last 9 bits for connection size. Use operator **|=** to set them
3. IO connection path must be a vector of byte (as you see it in EDS file or specification): 0x20 0x04 CONFIG_ASSEMBLY_ID 0x2C OUTPUT_ASSEMBLY_ID 0x2C INPUT_ASSEMBLY
4. *RPI* and *API* in microseconds

If method *ConnectionManager::forwardOpen* has managed to open the connection it returns a weak pointer to it else null pointer:

```
if (auto ptr = io.lock()) {
        // Set data to send
        ptr->setDataToSend(std::vector<uint8_t>(32));

        // Set callback for received data
        ptr->setReceiveDataListener([](auto realTimeHeader, auto sequence, auto data)
↪{
                std::ostringstream ss;
                ss << "secNum=" << sequence << " data=";
                for (auto &byte : data) {
                        ss << "[" << std::hex << (int) byte << "]";
                }

                Logger(LogLevel::INFO) << "Received: " << ss.str();
        });
```

(continues on next page)

(continued from previous page)

```
        // Set callback to no
        ptr->setCloseListener([]() {
                Logger(LogLevel::INFO) << "Closed";
        });
}
```

In this snippet, we set the data to send and subscribe on the two events: the data is received and the connection is closed.

Note: Pay attention, that the size of the data is the same as the O=>T connection size if the connection has the fixed size. Some device can ignore this data and close the connection by timeout.

To open a connection and set the listeners are not enough to make it work. **EIPScanner** is a single thread library and we need periodically to handle these connections:

```
int count = 200;
while (connectionManager.hasOpenConnections() && count-- > 0) {
        connectionManager.handleConnections(std::chrono::milliseconds(100));
}
```

This loop executes 200 times and while there are the open connections to handle. Method *ConnectionManager::handleConnections* does several things for each IO connection:

1. Checks if the new data received via UDP and calls the corresponding handler that has been set by *IOConnection::setReceiveDataListener*.
2. Closes a connection if it hasn't been receiving new data during *IOConnection::t2oAPI* x *ConnectionParameters::connectionTimeoutMultiplier* x 4 and calls the corresponding handler.
3. Sends the data each *IOConnection::o2tAPI*.

Note: You should call method *ConnectionManager::handleConnections* more often than the least API of opened connections.

The last thing, we should do, is close the connection politely:

```
connectionManager.forwardClose(si, io)
```

1.4 Discovery

EtherNet/IP allows to discover **EIP** devices in the network by using UDP broadcast messages.

EIPScanner provides *DiscoveryManager* class for this purpose:

```
//
// Created by Aleksey Timin on 12/17/19.
//

#if defined(_WIN32) || defined(WIN32) || defined(_WIN64)
#include <winsock2.h>
#define OS_Windows (1)
#endif
```

(continues on next page)

(continued from previous page)

```
#include <DiscoveryManager.h>
#include <utils/Logger.h>

using eipScanner::DiscoveryManager;
using eipScanner::utils::Logger;
using eipScanner::utils::LogLevel;

int main() {
  Logger::setLogLevel(LogLevel::DEBUG);

#if OS_Windows
  WSADATA wsaData;
  int winsockStart = WSAStartup(MAKEWORD(2, 2), &wsaData);
  if (winsockStart != 0) {
    Logger(LogLevel::ERROR) << "Failed to start WinSock - error code: " <<␣
↪winsockStart;
    return EXIT_FAILURE;
  }
#endif

  DiscoveryManager discoveryManager("172.28.255.255", 0xAF12,␣
↪std::chrono::seconds(1));
  auto devices = discoveryManager.discover();

  for (auto& device : devices) {
    Logger(LogLevel::INFO) << "Discovered device: "
      << device.identityObject.getProductName()
      << " with address " << device.socketAddress.toString();
  }

#if OS_Windows
  WSACleanup();
#endif

  return EXIT_SUCCESS;
}
```

Method *DiscoveryManager::discover* sends broadcast UDP request in the network and waits for the responses from the devices. It returns a vector of structures for each discovered device that contain the IP addresses, ports and identity_objects. If there is no device in the network it returns an empty vector.

1.5 Standard CIP Objects

EIPScanner provides some classes, that help the users to work with standard **CIP** objects without knowing their specifications.

1.5.1 Identity Object (0x01)

Identity Object provides identification and general information about the *CIP* devices. It presents in all *CIP* products.

You can read this information with *IdentityObject* class:

```
//
// Created by Aleksey Timin on 12/19/19.
//

#if defined(_WIN32) || defined(WIN32) || defined(_WIN64)
#include <winsock2.h>
#define OS_Windows (1)
#endif

#include "IdentityObject.h"
#include "utils/Logger.h"

using eipScanner::IdentityObject;
using eipScanner::SessionInfo;
using eipScanner::utils::Logger;
using eipScanner::utils::LogLevel;

int main() {
  Logger::setLogLevel(LogLevel::DEBUG);

#if OS_Windows
  WSADATA wsaData;
  int winsockStart = WSAStartup(MAKEWORD(2, 2), &wsaData);
  if (winsockStart != 0) {
    Logger(LogLevel::ERROR) << "Failed to start WinSock - error code: " <<␣
↪winsockStart;
    return EXIT_FAILURE;
  }
#endif

  auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);
  IdentityObject identityObject(1, si);

  Logger(LogLevel::INFO) << identityObject.getVendorId()
      << identityObject.getDeviceType()
      << identityObject.getProductCode()
      << identityObject.getRevision().toString()
      << identityObject.getStatus()
      << identityObject.getSerialNumber()
      << identityObject.getProductName();

#if OS_Windows
  WSACleanup();
#endif

  return EXIT_SUCCESS;
}
```

The constructor takes *instanceID* and *SessionInfo* as arguments to read data via *EtherNet/IP*.

1.5.2 Parameter Object (0x0f)

Parameter Object provides a standard way to access to data and configuration of the **CIP** device.

EIPScanner has special class *ParameterObject* to read an parameter, but before use it you should know:

1. *Instance ID* of the parameter or how many parameters the device has to read all of them

2. If the device supports full attributes of Paramter Object (sting descriptions, scaling, etc.) or not

The following example shows how you can get the necessary information from Parameter Object class and read all the parameters from **EIP** device:

```
//
// Created by Aleksey Timin on 12/4/19.
//

#if defined(_WIN32) || defined(WIN32) || defined(_WIN64)
#include <winsock2.h>
#define OS_Windows (1)
#endif

#include "ParameterObject.h"
#include "utils/Logger.h"
#include "utils/Buffer.h"

using namespace eipScanner::cip;
using eipScanner::SessionInfo;
using eipScanner::MessageRouter;
using eipScanner::utils::Logger;
using eipScanner::utils::LogLevel;
using eipScanner::ParameterObject;
using eipScanner::utils::Buffer;

const CipUint MAX_INSTANCE = 2;
const CipUint CLASS_DESCRIPTOR = 8;
const CipUint SUPPORTS_FULL_ATTRIBUTES = 0x2;

int main() {
  Logger::setLogLevel(LogLevel::DEBUG);

#if OS_Windows
  WSADATA wsaData;
  int winsockStart = WSAStartup(MAKEWORD(2, 2), &wsaData);
  if (winsockStart != 0) {
    Logger(LogLevel::ERROR) << "Failed to start WinSock - error code: " <<␣
↪winsockStart;
    return EXIT_FAILURE;
  }
#endif

  auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);

  // Read the number of the parameters
  MessageRouter messageRouter;
  auto response = messageRouter.sendRequest(si
      , ServiceCodes::GET_ATTRIBUTE_SINGLE
      , EPath(ParameterObject::CLASS_ID, 0, MAX_INSTANCE));

  if (response.getGeneralStatusCode() != GeneralStatusCodes::SUCCESS) {
    Logger(LogLevel::ERROR) << "Failed to read the count of the parameters";
    logGeneralAndAdditionalStatus(response);
    return -1;
  }

  Buffer buffer(response.getData());
```

(continues on next page)

(continued from previous page)

```
  CipUint paramsCount;
  buffer >> paramsCount;

  Logger(LogLevel::INFO) << "The device has " << paramsCount << "parameters";

  // Read Parameter Class Descriptor
  response = messageRouter.sendRequest(si
      , ServiceCodes::GET_ATTRIBUTE_SINGLE
      , EPath(ParameterObject::CLASS_ID, 0, CLASS_DESCRIPTOR));

  if (response.getGeneralStatusCode() != GeneralStatusCodes::SUCCESS) {
    Logger(LogLevel::ERROR) << "Failed to read the class descriptor";
    logGeneralAndAdditionalStatus(response);
    return -1;
  }

  buffer = Buffer(response.getData());
  CipUint descriptor;
  buffer >> descriptor;

  Logger(LogLevel::INFO) << "Read the class descriptor=0x" << std::hex <<␣
↪(int)descriptor;
  bool allAttributes = descriptor & SUPPORTS_FULL_ATTRIBUTES;

  // Read and save parameters in a vector
  std::vector<ParameterObject> parameters;
  parameters.reserve(paramsCount);
  for (int i = 0; i < paramsCount; ++i) {
    parameters.emplace_back(i+1, allAttributes, si);
  }

  if (!parameters.empty()) {
    parameters[0].getType(); // Read type
    parameters[0].getActualValue<CipUint>(); // 2040
    parameters[0].getEngValue<CipUint>(); // 20.4
    parameters[0].getName();  // Freq
    parameters[0].getUnits();   // Hz
    // .. etc

    parameters[0].updateValue(si);
    parameters[0].getActualValue<CipUint>(); // updated value
  }

#if OS_Windows
  WSACleanup();
#endif

  return EXIT_SUCCESS;
}
```

The example is pretty big and we need to look at it closer:

Below we read the number of parameters in the device by reading *Max Instance* attribute of *Parameter Object* class. For an example, if the number equals 5, we have 5 *Parameter Object* instances with IDs from 1 to 5:

```
MessageRouter messageRouter;
auto response = messageRouter.sendRequest(si
```

(continues on next page)

(continued from previous page)

```
        , ServiceCodes::GET_ATTRIBUTE_SINGLE
        , EPath(ParameterObject::CLASS_ID, 0, MAX_INSTANCE));

if (response.getGeneralStatusCode() != GeneralStatusCodes::SUCCESS) {
    Logger(LogLevel::ERROR) << "Failed to read the count of the parameters";
    logGeneralAndAdditionalStatus(response);
    return -1;
}

Buffer buffer(response.getData());
CipUint paramsCount;
buffer >> paramsCount;
```

But to know the number of the parameters is not enough. We need to figure out if the parameters support full attributes. This information is stored in the second bit of *Parameter Classe Descriptor* and we have to read it:

```
response = messageRouter.sendRequest(si
        , ServiceCodes::GET_ATTRIBUTE_SINGLE
        , EPath(ParameterObject::CLASS_ID, 0, CLASS_DESCRIPTOR));

if (response.getGeneralStatusCode() != GeneralStatusCodes::SUCCESS) {
    Logger(LogLevel::ERROR) << "Failed to read the class descriptor";
    logGeneralAndAdditionalStatus(response);
    return -1;
}

buffer = Buffer(response.getData());
CipUint descriptor;
buffer >> descriptor;

Logger(LogLevel::INFO) << "Read the class descriptor=0x" << std::hex <<␣
↪(int)descriptor;
bool allAttributes = descriptor & SUPPORTS_FULL_ATTRIBUTES;
```

Now we know all that we need and we are able to read the parameters and save them in a vector:

```
std::vector<ParameterObject> parameters;
parameters.reserve(paramsCount);
for (int i = 0; i < paramsCount; ++i) {
    parameters.emplace_back(i+1, allAttributes, si);
}
```

When the parameters are read in its constructors, you can access to its attributes by using the getters:

```
if (!parameters.empty()) {
    parameters[0].getType(); // Read type
    parameters[0].getActualValue<CipUint>(); // 2040
    parameters[0].getEngValue<CipUint>(); // 20.4
    parameters[0].getName();        // Freq
    parameters[0].getUnits();       // Hz
    // .. etc

    parameters[0].updateValue(si);
    parameters[0].getActualValue<CipUint>(); // updated value
}
```

Note: After an instance of *ParamterObject* is created it doesn't update its attributes. You can update only the actual value with method *ParamterObject::updateValue*

1.5.3 File Object (0x37)

File Object allows to transfer files between a scanner and an adapter. It might be helpful if you want to read *EDS* file from the **EIP** device.

EIPScanner implements only reading files with *FileObject* class. Below you can see how to read *EDS* file from the device:

```
//
// Created by Aleksey Timin on 11/24/19.
//

#if defined(_WIN32) || defined(WIN32) || defined(_WIN64)
#include <winsock2.h>
#define OS_Windows (1)
#endif

#include <fstream>
#include "FileObject.h"
#include "utils/Logger.h"
#include "fileObject/FileObjectState.h"

using namespace eipScanner::cip;
using eipScanner::SessionInfo;
using eipScanner::utils::Logger;
using eipScanner::utils::LogLevel;
using eipScanner::FileObject;

int main() {
  Logger::setLogLevel(LogLevel::DEBUG);

#if OS_Windows
  WSADATA wsaData;
  int winsockStart = WSAStartup(MAKEWORD(2, 2), &wsaData);
  if (winsockStart != 0) {
    Logger(LogLevel::ERROR) << "Failed to start WinSock - error code: " <<␣
↪winsockStart;
    return EXIT_FAILURE;
  }
#endif

  auto si = std::make_shared<SessionInfo>("172.28.1.3", 0xAF12);

  FileObject edsFile(0xC8, si);
  edsFile.beginUpload(si, [](GeneralStatusCodes status, const std::vector<uint8_t> &
↪fileContent) {
    if (status == GeneralStatusCodes::SUCCESS) {
      std::ofstream outFile("Device.eds", std::ios::out | std::ios::trunc |␣
↪std::ios::binary);
      outFile.write(reinterpret_cast<const char *>(fileContent.data()), fileContent.
↪size());
    }
```

(continues on next page)

(continued from previous page)

```
  });

  while (edsFile.handleTransfers(si)) {
    continue;
  };

#if OS_Windows
  WSACleanup();
#endif

  return EXIT_SUCCESS;
}
```

File Object sends data split into chunks (max 255), so we need to receive all of them after we've begun uploading. Method *FileObject::handleTransfers* receives all the chunks and calls a handler, which we set in method *FileObject::beginUpload*, where we save the received data as a file.

1.6 Vendor Specific CIP Objects

: .. toctree:

```
:maxdepth: 2
:caption: Contents:

ra/_index.rst
```

1.7 Miscellaneous

1.7.1 Byte Buffer

EtherNet/IP has a type system and specifies how the types must be sends over the network. So we have to decode and encode our C++ types. To make it easy, **EIPScanner** have class **utils::Buffer**.

To decode the data that we've received from the network use operator >>:

```
Buffer buffer(dataFromNetwork);
cip::CipUint var1;
cip::CipReal var2;
buffer >> var1 >> var2;
```

To encode the data that we are going to send, use operator <<:

```
Buffer buffer;
cip::CipUint var1 = 10;
cip::CipReal var2 = 2.4;
buffer << var1 << var2;

buffer.data(); // => data to send
```

1.7.2 Logging

EIPScanner has a very simple embedded logger *utils::Logger* which prints the messages into stdout. This is a simple usage example:

```
#include "utils/Logger.h"

using eipScanner::utils::Logger;
using eipScanner::utils::LogLevel;

int main() {
    Logger::setLogLevel(LogLevel::INFO);

    Logger(LogLevel::INFO) << "You will see this message.";
    Logger(LogLevel::DEBUG) << "You won't see this message.";
    return 0;
}
```

You can set the lowest log level for all your application by method *Logger::setLogLevel*. Here it is *INFO*, so that DEBUG level isn't printed.

Perhaps, the embedded logger isn't suitable for your application or you use another one. No problems. Implement interface *utils::LogAppenderIf* and set it with method *Logger::setAppender*.

Note: The printing messages happens in the destructor of *utils::Logger*. It means if you want to see message at once, don't save the logger in a variable or a field.

1.7.3 8-bit Path Segments

Some devices only support 8-bit path segments. In order to set up **EIPScanner** to use 8-bit path segments, a *MessageRouter* with the **USE_8_BIT_PATH_SEGMENTS** flag set should be passed to the *ConnectionManager* upon construction:

```
#include "MessageRouter.h"
#include "ConnectionManager.h"

using eipScanner::ConnectionManager;
using eipScanner::MessageRouter;

int main()
{
    MessageRouter::SPtr mr_ptr = std::make_shared<MessageRouter>(MessageRouter::USE_8_
↪BIT_PATH_SEGMENTS);
    ConnectionManager _connectionManager = ConnectionManager(mr_ptr);

    /* ConnectionManager now uses 8-bit path segments */

    return 0;
}
```

1.8 Library API

1.8.1 Full API

Namespaces

Namespace eipScanner

Contents

Namespaces

Classes

Enums

- *Enum ConnectionManagerServiceCodes*
- *Enum DescriptorAttributeBits*
- *Enum FileObjectStateCodes*
- *Enum ParameterObjectAttributeIds*

Namespace eipScanner::cip

Contents

Namespaces

- *Namespace eipScanner::cip::connectionManager*

Classes

- *Template Class CipBaseString*
- *Class CipRevision*
- *Class EPath*
- *Class MessageRouterRequest*
- *Class MessageRouterResponse*
- *Class Yaskawa_EPath*
- *Class Yaskawa_MessageRouterRequest*

Enums

- *Enum CipDataTypes*
- *Enum EPathSegmentTypes*
- *Enum EPathSegmentTypes*
- *Enum GeneralStatusCodes*
- *Enum ServiceCodes*

Functions

- *Function eipScanner::cip::logGeneralAndAdditionalStatus*

Typedefs

- *Typedef eipScanner::cip::CipBool*
- *Typedef eipScanner::cip::CipByte*
- *Typedef eipScanner::cip::CipDint*
- *Typedef eipScanner::cip::CipDword*
- *Typedef eipScanner::cip::CipInt*
- *Typedef eipScanner::cip::CipLint*
- *Typedef eipScanner::cip::CipLreal*
- *Typedef eipScanner::cip::CipLword*
- *Typedef eipScanner::cip::CipOctet*
- *Typedef eipScanner::cip::CipReal*
- *Typedef eipScanner::cip::CipShortString*
- *Typedef eipScanner::cip::CipSint*
- *Typedef eipScanner::cip::CipString*
- *Typedef eipScanner::cip::CipUdint*
- *Typedef eipScanner::cip::CipUint*
- *Typedef eipScanner::cip::CipUlint*
- *Typedef eipScanner::cip::CipUsint*
- *Typedef eipScanner::cip::CipWord*

Namespace eipScanner::cip::connectionManager

Contents

Classes

- *Struct ConnectionParameters*
- *Class ForwardCloseRequest*
- *Class ForwardOpenRequest*
- *Class ForwardOpenResponse*

- *Class LargeForwardOpenRequest*
- *Class NetworkConnectionParametersBuilder*

Enums

- *Enum NetworkConnectionParams*

Namespace eipScanner::eip

Contents

- *Classes*
- *Enums*

Classes

- *Class CommonPacket*
- *Class CommonPacketItem*
- *Class CommonPacketItemFactory*
- *Class EncapsPacket*
- *Class EncapsPacketFactory*

Enums

- *Enum CommonPacketItemIds*
- *Enum EncapsCommands*
- *Enum EncapsStatusCodes*

Namespace eipScanner::fileObject

Contents

- *Classes*
- *Enums*
- *Typedefs*
- *Variables*

Classes

- *Class FileObjectEmptyState*
- *Class FileObjectLoadedState*
- *Class FileObjectNonExistentState*
- *Class FileObjectState*
- *Class FileObjectUploadInProgressState*

Enums

- *Enum FileObjectAttributesCodes*
- *Enum FileObjectServiceCodes*
- *Enum TransferPacketTypeCodes*

Typedefs

- *Typedef eipScanner::fileObject::EndUploadHandler*

Variables

- *Variable eipScanner::fileObject::FILE_OBJECT_CLASS_ID*
- *Variable eipScanner::fileObject::MAX_TRANSFER_SIZE*

Namespace eipScanner::sockets

Contents

Classes

- *Class BaseSocket*
- *Class EndPoint*
- *Class TCPSocket*
- *Class UDPBoundSocket*
- *Class UDPSocket*

Namespace eipScanner::utils

Contents

Classes

Enums

Namespace eipScanner::vendor

Contents

Namespaces

Namespace eipScanner::vendor::ra

Contents

Namespaces

Namespace eipScanner::vendor::ra::powerFlex525

Contents

Classes

Enums

Functions

Variables

- *Variable eipScanner::vendor::ra::powerFlex525::MAX_FAULT_PARAMETER_NUMBER*

Namespace std

Classes and Structs

Struct ConnectionParameters

- Defined in file_src_cip_connectionManager_ConnectionParameters.h

Struct Documentation

struct ConnectionParameters

Public Members

CipUsint **priorityTimeTick** = 0

CipUsint **timeoutTicks** = 0

CipUdint **o2tNetworkConnectionId** = 0

CipUdint **t2oNetworkConnectionId** = 0

CipUint **connectionSerialNumber** = 0

CipUint **originatorVendorId** = 0

CipUdint **originatorSerialNumber** = 0

CipUsint **connectionTimeoutMultiplier** = 0

CipUdint **o2tRPI** = 0

CipUdint **o2tNetworkConnectionParams** = 0

CipUdint **t2oRPI** = 0

CipUdint **t2oNetworkConnectionParams** = 0

CipUsint **transportTypeTrigger** = 0

CipUsint **connectionPathSize** = 0

CipBool **o2tRealTimeFormat** = 0

CipBool **t2oRealTimeFormat** = 0

std::vector<uint8_t> **connectionPath** = { }

Struct IdentityItem

- Defined in file_src_DiscoveryManager.h

Struct Documentation

struct IdentityItem
Contains information about EIP device and its address in the network.

Public Types

using Vec = std::vector<*IdentityItem*>

Public Members

IdentityObject **identityObject**

sockets::*EndPoint* **socketAddress**

Struct DPIFaultCode::FaultDescriptions

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultCode.h

Nested Relationships

This struct is a nested type of *Class DPIFaultCode*.

Struct Documentation

struct FaultDescriptions

Public Members

int **faultType**

string **faultText**

string **faultDescription**

Struct DPIFaultObject::FullInformation

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultObject.h

Nested Relationships

This struct is a nested type of *Class DPIFaultObject*.

Struct Documentation

struct FullInformation
Informaion about the fault

Public Members

cip::*CipUint* **faultCode**
: the code of the fault (0 is no fault)

cip::*CipUsint* **dsiPort**
: DSI port.

cip::*CipUsint* **dsiDeviceObject**
: DSI Device Object.

cip::*CipString* **faultText**
: the text of the fault

cip::*CipLword* **timerValue**
: timer value

bool **isValidData**
: true if the timer value valid

bool **isRealTime**
: true if the time is real else it is elapsed

Struct DPIFaultParameter::FaultDetails

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.h

Nested Relationships

This struct is a nested type of *Class DPIFaultParameter*.

Struct Documentation

struct FaultDetails

Public Members

int **faultNumber**

cip::*CipUint* **faultCode**

cip::*CipLreal* **busVoltage**

cip::*CipLreal* **current**

cip::*CipLreal* **frequency**

Struct DPIFaultParameter::FullInformation

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.h

Nested Relationships

This struct is a nested type of *Class DPIFaultParameter*.

Struct Documentation

struct FullInformation

Public Members

FaultDetails **faultDetails**

DPIFaultCode::*FaultDescriptions* **faultDescription**

Class BaseObject

- Defined in file_src_BaseObject.h

Inheritance Relationships

Derived Types

- `public eipScanner::FileObject` (*Class FileObject*)
- `public eipScanner::IdentityObject` (*Class IdentityObject*)
- `public eipScanner::ParameterObject` (*Class ParameterObject*)
- `public eipScanner::vendor::ra::powerFlex525::DPIFaultObject` (*Class DPIFaultObject*)

Class Documentation

class BaseObject

Base class for all the CIP Objects.

Subclassed by *eipScanner::FileObject*, *eipScanner::IdentityObject*, *eipScanner::ParameterObject*, *eipScanner::vendor::ra::powerFlex525::DPIFaultObject*

Public Functions

BaseObject(cip::*CipUint* *classId*, cip::*CipUint* *instanceId*)

Creates an CIP instance.

Parameters

- `classId`: the class ID of the the new instance
- `instanceId`: the instance ID of the the new instanc

cip::*CipUint* **getClassId**() **const**
Gets the class ID of the instance.

Return

cip::*CipUint* **getInstanceId**() **const**
Gets the instance ID of the instance.

Return

Template Class CipBaseString

- Defined in file_src_cip_CipString.h

Class Documentation

template<typename **T**>
class CipBaseString

Public Functions

CipBaseString()

CipBaseString(**const** std::string &*string*)

CipBaseString(**const** std::vector<uint8_t> &*data*)

~CipBaseString()

std::string **toStdString**() **const**

T **getLength**() **const**

const std::vector<uint8_t> &**getData**() **const**

Class CipRevision

- Defined in file_src_cip_CipRevision.h

Class Documentation

class CipRevision

Public Functions

CipRevision()

CipRevision(*CipUsint majorRevision*, *CipUsint minorRevision*)

bool **operator==**(**const** *CipRevision* &*other*) **const**

std::string **toString**() **const**

CipUsint **getMajorRevision**() **const**

CipUsint **getMinorRevision**() **const**

Class ForwardCloseRequest

- Defined in file_src_cip_connectionManager_ForwardCloseRequest.h

Class Documentation

class ForwardCloseRequest

Public Functions

ForwardCloseRequest()

~ForwardCloseRequest()

std::vector<uint8_t> **pack**() **const**

void **setConnectionSerialNumber**(*CipUint connectionSerialNumber*)

void **setOriginatorVendorId**(*CipUint originatorVendorId*)

void **setOriginatorSerialNumber**(*CipUdint originatorSerialNumber*)

void **setConnectionPath**(**const** std::vector<uint8_t> &*connectionPath*)

Class ForwardOpenRequest

- Defined in file_src_cip_connectionManager_ForwardOpenRequest.h

Class Documentation

class ForwardOpenRequest

Public Functions

ForwardOpenRequest(*ConnectionParameters params*)

~ForwardOpenRequest()

std::vector<uint8_t> **pack**() **const**

Class ForwardOpenResponse

- Defined in file_src_cip_connectionManager_ForwardOpenResponse.h

Class Documentation

class ForwardOpenResponse

Public Functions

ForwardOpenResponse()

~ForwardOpenResponse()

void **expand**(**const** std::vector<uint8_t> &*data*)

CipUdint **getO2TNetworkConnectionId**() **const**

CipUdint **getT2ONetworkConnectionId**() **const**

CipUint **getConnectionSerialNumber**() **const**

CipUint **getOriginatorVendorId**() **const**

CipUdint **getOriginatorSerialNumber**() **const**

CipUdint **getO2TApi**() **const**

CipUdint **getT2OApi**() **const**

CipUsint **getApplicationReplaySize**() **const**

const std::vector<uint8_t> &**getApplicationReplay**() **const**

Class LargeForwardOpenRequest

- Defined in file_src_cip_connectionManager_LargeForwardOpenRequest.h

Class Documentation

class LargeForwardOpenRequest

Public Functions

LargeForwardOpenRequest(*ConnectionParameters params*)

~LargeForwardOpenRequest()

std::vector<uint8_t> **pack**() **const**

Class NetworkConnectionParametersBuilder

- Defined in file_src_cip_connectionManager_NetworkConnectionParametersBuilder.h

Class Documentation

class NetworkConnectionParametersBuilder

Public Types

enum RedundantOwner
Values:

EXCLUSIVE

REDUNDANT

enum ConnectionType
Values:

NULL_TYPE

MULTICAST

P2P

RESERVED

enum Priority
Values:

LOW_PRIORITY

HIGH_PRIORITY

SCHEDULED

URGENT

enum Type
Values:

FIXED

VARIABLE

Public Functions

NetworkConnectionParametersBuilder(*CipUdint* *val* = 0, bool *lfo* = false)

virtual ~NetworkConnectionParametersBuilder()

NetworkConnectionParametersBuilder **setRedundantOwner**(*RedundantOwner* *val*)

NetworkConnectionParametersBuilder &**setConnectionType**(*ConnectionType* *val*)

NetworkConnectionParametersBuilder &**setPriority**(*Priority* *val*)

NetworkConnectionParametersBuilder &**setType**(*Type* *val*)

NetworkConnectionParametersBuilder &**setConnectionSize**(*CipUint* *val*)

CipUdint **build**()

NetworkConnectionParametersBuilder::*RedundantOwner* **getRedundantOwner**() **const**

NetworkConnectionParametersBuilder::*ConnectionType* **getConnectionType**() **const**

NetworkConnectionParametersBuilder::*Priority* **getPriority**() **const**

NetworkConnectionParametersBuilder::*Type* **getType**() **const**

CipUint **getConnectionSize**() **const**

Class EPath

- Defined in file_src_cip_EPath.h

Class Documentation

class EPath

Public Functions

EPath()

EPath(*CipUint classId*)

EPath(*CipUint classId*, *CipUint objectId*)

EPath(*CipUint classId*, *CipUint objectId*, *CipUint attributeId*)

std::vector<uint8_t> **packPaddedPath**(bool *use_8_bit_path_segments* = false) **const**

void **expandPaddedPath**(**const** std::vector<uint8_t> &*data*)

CipUint **getClassId**() **const**

CipUint **getObjectId**() **const**

CipUint **getAttributeId**() **const**

CipUsint **getSizeInWords**(bool *use_8_bit_path_segments* = false) **const**

std::string **toString**() **const**

bool **operator==**(**const** *EPath* &*other*) **const**

Class MessageRouterRequest

- Defined in file_src_cip_MessageRouterRequest.h

Class Documentation

class MessageRouterRequest

Public Functions

MessageRouterRequest(*CipUsint serviceCode*, **const** *EPath &ePath*, **const** std::vector<uint8_t> *data*, bool *use_8_bit_path_segments* = false)

~MessageRouterRequest()

std::vector<uint8_t> **pack**() **const**

Class MessageRouterResponse

- Defined in file_src_cip_MessageRouterResponse.h

Class Documentation

class MessageRouterResponse

Public Functions

MessageRouterResponse()

~MessageRouterResponse()

void **expand**(**const** std::vector<uint8_t> &*data*)

GeneralStatusCodes **getGeneralStatusCode**() **const**

ServiceCodes **getServiceCode**() **const**

const std::vector<uint16_t> &**getAdditionalStatus**() **const**

const std::vector<uint8_t> &**getData**() **const**

const std::vector<eip::*CommonPacketItem*> &**getAdditionalPacketItems**() **const**

void **setGeneralStatusCode**(*GeneralStatusCodes generalStatusCode*)

void **setData**(**const** std::vector<uint8_t> &*data*)

void **setAdditionalPacketItems**(**const** std::vector<eip::*CommonPacketItem*> &*_additionalPacketItems*)

Class Yaskawa_EPath

- Defined in file_src_vendor_yaskawa_mp3300iec_Yaskawa_EPath.h

Class Documentation

class Yaskawa_EPath

Public Functions

Yaskawa_EPath()

Yaskawa_EPath(*CipUsint classId*)

Yaskawa_EPath(*CipUsint classId*, *CipUsint objectId*)

Yaskawa_EPath(*CipUsint classId*, *CipUsint objectId*, *CipUsint attributeId*)

std::vector<uint8_t> **packPaddedPath**() **const**

void **expandPaddedPath**(**const** std::vector<uint8_t> &*data*)

CipUsint **getClassId**() **const**

CipUsint **getObjectId**() **const**

CipUsint **getAttributeId**() **const**

CipUsint **getSizeInWords**() **const**

std::string **toString**() **const**

bool **operator==**(**const** *Yaskawa_EPath* &*other*) **const**

Class Yaskawa_MessageRouterRequest

- Defined in file_src_vendor_yaskawa_mp3300iec_Yaskawa_MessageRouterRequest.h

Class Documentation

class Yaskawa_MessageRouterRequest

Public Functions

Yaskawa_MessageRouterRequest(*CipUsint serviceCode*, **const** *Yaskawa_EPath* &*ePath*, **const** std::vector<uint8_t> *data*)

~Yaskawa_MessageRouterRequest()

std::vector<uint8_t> **pack**() **const**

Class ConnectionManager

- Defined in file_src_ConnectionManager.h

Class Documentation

class ConnectionManager

Implements the implicit messaging with EIP adapter.

Public Functions

ConnectionManager()
Default constructor.

ConnectionManager(**const** *MessageRouter*::*SPtr* &*messageRouter*)

Note used fot testing

Parameters

- `messageRouter`:

~ConnectionManager()
Default destructor.

IOConnection::*WPtr* **forwardOpen**(**const** *SessionInfoIf*::*SPtr* &*si*, cip::connectionManager::*ConnectionParameters* *connectionParameters*, bool *isLarge* = false)
Opens an EIP IO connection with the EIP adapter.

Return weak pointer to the created connection or nullptr if got an error

Parameters

- `si`: the EIP session for explicit messaging
- `connectionParameters`: the parameters of the connection
- `isLarge`: use large forward open if true

IOConnection::*WPtr* **largeForwardOpen**(**const** *SessionInfoIf*::*SPtr* &*si*, cip::connectionManager::*ConnectionParameters* *connectionParameters*)
Opens an EIP IO connection with the EIP adapter.

Return weak pointer to the created connection or nullptr if got an error

Parameters

- `si`: the EIP session for explicit messaging
- `connectionParameters`: the parameters of the connection

void **forwardClose**(**const** *SessionInfoIf*::*SPtr* &*si*, **const** *IOConnection*::*WPtr* &*ioConnection*)
Closes an EIP IO connection.

Parameters

- `si`: the EIP session for explicit messaging
- `ioConnection`: the connection to close

void **handleConnections**(std::chrono::milliseconds *timeout*)
Handles active connections.

Parameters

- `timeout`: the timeout of receiving the data by select function

bool **hasOpenConnections**() **const**

Return true if there are some opened IO connections

Class DiscoveryManager

- Defined in file_src_DiscoveryManager.h

Class Documentation

class DiscoveryManager

Implements the discovery of EIP devices in the network.

Public Functions

DiscoveryManager(**const** std::string &*broadCastAddress*, int *port*, std::chrono::milliseconds *receiveTimout*)

Constructor.

Parameters

- `broadCastAddress`: the broadcast address to send a request (e.g. 172.28.255.255 or 192.168.1.255)
- `port`: the port of the discovery (default 0xAF12)
- `receiveTimout`: the timeout to wait for responses from the EIP devices

~DiscoveryManager()

default destructor

IdentityItem::*Vec* **discover**() **const**

Discovers the EIP network Sends the broadcast request through UDP and collects all the response.

Return the vector of *IdentityItem* for each discovered device.

Protected Functions

sockets::*BaseSocket*::*SPtr* **makeSocket**() **const**

Class CommonPacket

- Defined in file_src_eip_CommonPacket.h

Class Documentation

class CommonPacket

Public Functions

CommonPacket()

~CommonPacket()

CommonPacket &**operator<<**(**const** *CommonPacketItem* &*item*)

std::vector<uint8_t> **pack**() **const**

void **expand**(**const** std::vector<uint8_t> &*data*)

const *CommonPacketItem*::*Vec* &**getItems**()

Class CommonPacketItem

- Defined in file_src_eip_CommonPacketItem.h

Class Documentation

class CommonPacketItem

Public Types

using Vec = std::vector<*CommonPacketItem*>

Public Functions

CommonPacketItem()

CommonPacketItem(*CommonPacketItemIds typeId*, **const** std::vector<uint8_t> &*data*)

~CommonPacketItem()

std::vector<uint8_t> **pack**() **const**

CommonPacketItemIds **getTypeId**() **const**

cip::*CipUint* **getLength**() **const**

const std::vector<uint8_t> &**getData**() **const**

bool **operator==**(**const** *CommonPacketItem* &*rhs*) **const**

bool **operator!=**(**const** *CommonPacketItem* &*rhs*) **const**

Class CommonPacketItemFactory

- Defined in file_src_eip_CommonPacketItemFactory.h

Class Documentation

class CommonPacketItemFactory

Public Functions

CommonPacketItem **createNullAddressItem**() **const**

CommonPacketItem **createUnconnectedDataItem**(**const** std::vector<uint8_t> &*data*) **const**

CommonPacketItem **createSequenceAddressItem**(cip::*CipUdint* *connectionId*, cip::*CipUdint* *seqNumber*) **const**

CommonPacketItem **createConnectedDataItem**(**const** std::vector<uint8_t> &*data*) **const**

Class EncapsPacket

- Defined in file_src_eip_EncapsPacket.h

Class Documentation

class EncapsPacket

Public Functions

EncapsPacket()

~EncapsPacket()

std::vector<uint8_t> **pack**() **const**

void **expand**(**const** std::vector<uint8_t> &*data*)

EncapsCommands **getCommand**() **const**

void **setCommand**(*EncapsCommands* *command*)

cip::*CipUint* **getLength**() **const**

cip::*CipUdint* **getSessionHandle**() **const**

void **setSessionHandle**(cip::*CipUdint* *sessionHandle*)

EncapsStatusCodes **getStatusCode**() **const**

void **setStatusCode**(*EncapsStatusCodes* *statusCode*)

const std::vector<uint8_t> &**getData**() **const**

void **setData**(**const** std::vector<uint8_t> &*data*)

bool **operator==**(**const** *EncapsPacket* &*rhs*) **const**

bool **operator!=**(**const** *EncapsPacket* &*rhs*) **const**

Public Static Functions

size_t **getLengthFromHeader**(**const** std::vector<uint8_t> &*data*)

Public Static Attributes

const size_t **HEADER_SIZE** = 24

Class EncapsPacketFactory

- Defined in file_src_eip_EncapsPacketFactory.h

Class Documentation

class EncapsPacketFactory

Public Functions

EncapsPacket **createRegisterSessionPacket**() **const**

EncapsPacket **createUnRegisterSessionPacket**(cip::*CipUdint* *sessionHandle*) **const**

EncapsPacket **createSendRRDataPacket**(cip::*CipUdint* *sessionHandle*, cip::*CipUint* *timeout*, std::vector<uint8_t> *data*) **const**

EncapsPacket **createListIdentityPacket**() **const**

Class FileObject

- Defined in file_src_FileObject.h

Inheritance Relationships

Base Type

- `public eipScanner::BaseObject` (*Class BaseObject*)

Class Documentation

class FileObject : **public** eipScanner::*BaseObject*
Implements interface to File Object (0x37).

Public Types

using UPtr = std::unique_ptr<*FileObject*>

Public Functions

FileObject(cip::*CipUint* *instanceId*, **const** *SessionInfoIf*::*SPtr* &*si*)
Creates an instance and read the files object state.

Parameters

- `instanceId`: the ID of the CIP instance
- `si`: the EIP session for explicit messaging

Exceptions

- `std::runtime_error`:
- `std::system_error`:

FileObject(cip::*CipUint* *instanceId*, **const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*)

Note used for testing

Parameters

- `instanceId`:
- `si`:
- `messageRouter`:

~FileObject()
Default destructor.

fileObject::*FileObjectState*::*UPtr* &**getState**()
Gets the current state of the file.

Return

void **beginUpload**(*SessionInfoIf*::*SPtr* *si*, fileObject::*EndUploadHandler* *handle*)
Initiates uploading the file from the EIP adapter.

Parameters

- `si`: the EIP session for explicit messaging
- `handle`: a callback that called when the uploading finishes with an error or not

bool **handleTransfers**(*SessionInfoIf*::*SPtr* *si*)
Handle upload transfers.

Return true if uploading is in progress

Friends

friend eipScanner::FileObject::fileObject::FileObjectState

Class FileObjectEmptyState

- Defined in file_src_fileObject_FileObjectEmptyState.h

Inheritance Relationships

Base Type

- `public eipScanner::fileObject::FileObjectState` (*Class FileObjectState*)

Class Documentation

class FileObjectEmptyState : **public** eipScanner::fileObject::*FileObjectState*

Public Functions

FileObjectEmptyState(*FileObject* &*owner*, cip::*CipUint* *objectId*, *MessageRouter*::*SPtr* *messageRouter*)

void **initiateUpload**(*SessionInfoIf*::*SPtr* *si*, *EndUploadHandler* *handle*)

bool **transfer**(*SessionInfoIf*::*SPtr* *si*)

Class FileObjectLoadedState

- Defined in file_src_fileObject_FileObjectLoadedState.h

Inheritance Relationships

Base Type

- `public eipScanner::fileObject::FileObjectState` (*Class FileObjectState*)

Class Documentation

class FileObjectLoadedState : **public** eipScanner::fileObject::*FileObjectState*

Public Functions

FileObjectLoadedState(*FileObject* &*owner*, cip::*CipUint* *objectId*, *MessageRouter*::*SPtr* *messageRouter*)

void **initiateUpload**(*SessionInfoIf*::*SPtr* *si*, *EndUploadHandler* *handler*)

bool **transfer**(*SessionInfoIf*::*SPtr* *si*)

Class FileObjectNonExistentState

- Defined in file_src_fileObject_FileObjectNonExistentState.h

Inheritance Relationships

Base Type

- `public eipScanner::fileObject::FileObjectState` (*Class FileObjectState*)

Class Documentation

`class FileObjectNonExistentState` : **`public`** eipScanner::fileObject::*FileObjectState*

Public Functions

`FileObjectNonExistentState`(*FileObject* &*owner*, cip::*CipUint* *objectId*, *MessageRouter*::*SPtr* *messageRouter*)

void **`initiateUpload`**(*SessionInfoIf*::*SPtr* *si*, *EndUploadHandler* *handle*)

bool **`transfer`**(*SessionInfoIf*::*SPtr* *si*)

Class FileObjectState

- Defined in file_src_fileObject_FileObjectState.h

Inheritance Relationships

Derived Types

- `public eipScanner::fileObject::FileObjectEmptyState` (*Class FileObjectEmptyState*)
- `public eipScanner::fileObject::FileObjectLoadedState` (*Class FileObjectLoadedState*)
- `public eipScanner::fileObject::FileObjectNonExistentState` (*Class FileObjectNonExistentState*)
- `public eipScanner::fileObject::FileObjectUploadInProgressState` (*Class FileObjectUploadInProgressState*)

Class Documentation

`class FileObjectState`
Subclassed by *eipScanner::fileObject::FileObjectEmptyState*, *eipScanner::fileObject::FileObjectLoadedState*, *eipScanner::fileObject::FileObjectNonExistentState*, *eipScanner::fileObject::FileObjectUploadInProgressState*

Public Types

using UPtr = std::unique_ptr<*FileObjectState*>

Public Functions

FileObjectState(*FileObjectStateCodes state*, *FileObject &owner*, cip::*CipUint objectId*, *MessageRouter*::*SPtr messageRouter*)

~FileObjectState()

void **initiateUpload**(*SessionInfoIf*::*SPtr si*, *EndUploadHandler handle*)

bool **transfer**(*SessionInfoIf*::*SPtr si*)

FileObjectStateCodes **getStateCode**() **const**

void **SyncState**(*SessionInfoIf*::*SPtr si*)

Protected Functions

void **logWithStateName**(utils::*LogLevel logLevel*, **const** std::string &*message*) **const**

std::string **getStateName**() **const**

template<class **State**, class ...**Types**>
void **setState**(*Types*... *args*)

Protected Attributes

MessageRouter::*SPtr* **_messageRouter**

cip::*CipUint* **_objectId**

FileObjectStateCodes **_stateCode**

FileObject &**_owner**

Class FileObjectUploadInProgressState

- Defined in file_src_fileObject_FileObjectUploadInProgressState.h

Inheritance Relationships

Base Type

- `public eipScanner::fileObject::FileObjectState` (*Class FileObjectState*)

Class Documentation

class FileObjectUploadInProgressState : **public** eipScanner::fileObject::*FileObjectState*

Public Functions

FileObjectUploadInProgressState(*FileObject* &*owner*, cip::*CipUint* *objectId*, *MessageRouter*::*SPtr* *messageRouter*, cip::*CipUdint* *fileSize*, cip::*CipUsint* *transferSize*, *EndUploadHandler* *handler*)

void **initiateUpload**(*SessionInfoIf*::*SPtr* *si*, *EndUploadHandler* *handle*)

bool **transfer**(*SessionInfoIf*::*SPtr* *si*)

Class IdentityObject

- Defined in file_src_IdentityObject.h

Inheritance Relationships

Base Type

- `public eipScanner::BaseObject` (*Class BaseObject*)

Class Documentation

class IdentityObject : **public** eipScanner::*BaseObject*

Implements interface to Identity Object (0x01).

It reads all data from the CIP instance in the constructor

Public Functions

IdentityObject(cip::*CipUint* *instanceId*)

Creates an empty instance without any EIP requests.

Parameters

- `instanceId`: the ID of the CIP instance

IdentityObject(cip::*CipUint* *instanceId*, **const** *SessionInfoIf*::*SPtr* &*si*)

Creates an instance and reads all its data via EIP.

Parameters

- `instanceId`: the ID of the CIP instance
- `si`: the EIP session for explicit messaging

Exceptions

- `std::runtime_error`:
- `std::system_error`:

IdentityObject(cip::*CipUint* *instanceId*, **const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*)

Creates an instance and reads all its data via EIP.

Note Used for testing

Parameters

- `instanceId`:
- `si`:
- `messageRouter`:

CipUint **getVendorId**() **const**
Gets Vendor ID [AttrID=1].

Return

CipUint **getDeviceType**() **const**
Gets Device Type [AttrID=2].

Return

CipUint **getProductCode**() **const**
Gets Product Code [AttrID=3].

Return

const CipRevision &**getRevision**() **const**
Gets Revision [AttrID=4].

Return

CipWord **getStatus**() **const**
Gets Status [AttrID=5].

Return

CipUdint **getSerialNumber**() **const**
Gets Serial Number [AttrID=6].

Return

std::string **getProductName**() **const**
Gets Product Name [AttrID=7].

Return

void **setVendorId**(cip::*CipUint vendorId*)
Sets Vendor ID [AttrID=1].

Parameters

- `vendorId`:

void **setDeviceType**(cip::*CipUint deviceType*)
Sets Device Type [AttrID=2].

Parameters

- `deviceType`:

void **setProductCode**(cip::*CipUint* *productCode*)
Sets Product Code [AttrID=3].

Parameters

- `productCode`:

void **setRevision**(**const** cip::*CipRevision* &*revision*)
Sets Revision [AttrID=4].

Parameters

- `revision`:

void **setStatus**(cip::*CipWord* *status*)
Sets Status [AttrID=5].

Parameters

- `status`:

void **setSerialNumber**(cip::*CipUdint* *serialNumber*)
Sets Serial Number [AttrID=6].

Parameters

- `serialNumber`:

void **setProductName**(**const** std::string &*productName*)
Sets Product Name [AttrID=7].

Parameters

- `productName`:

Public Static Attributes

const cip::*CipUint* **CLASS_ID** = 0x01

Class IOConnection

- Defined in file_src_IOConnection.h

Class Documentation

class IOConnection
Implements an implicit EIP connection.

See *eipScanner::ConnectionManager*

Public Types

using ReceiveDataHandle = std::function<void(cip::*CipUdint*, cip::*CipUint*, **const** std::vector<uint8_t>&)>

using SendDataHandle = std::function<void(std::vector<uint8_t>&)>

using CloseHandle = std::function<void()>

using WPtr = std::weak_ptr<*IOConnection*>

using SPtr = std::shared_ptr<*IOConnection*>

Public Functions

~IOConnection()
Default destructor

void **setDataToSend**(**const** std::vector<uint8_t> &*data*)
Sets data to send via the connection each API period.

Note Set only data. The sequence counter and the real time format header are append automatically

Parameters

- `data`: the dat to send

void **setReceiveDataListener**(*ReceiveDataHandle* *handle*)
Sets a callback to handle received data.

Parameters

- `handle`:

void **setCloseListener**(*CloseHandle* *handle*)
Sets a callback to notify that the connection was closed.

Parameters

- `handle`:

void **setSendDataListener**(*SendDataHandle* *handle*)
Sets a callback to handle data to send.

Parameters

- `handle`:

Class MessageRouter

- Defined in file_src_MessageRouter.h

Class Documentation

class MessageRouter
Implements the explicit messaging with EIP adapter.

Public Types

using SPtr = std::shared_ptr<*MessageRouter*>

Public Functions

MessageRouter(bool *use_8_bit_path_segments* = false)
Default constructor.

~MessageRouter()
Default destructor.

MessageRouterResponse **sendRequest**(*SessionInfoIf*::*SPtr* *si*, cip::*CipUsint* *service*, **const** cip::*EPath* &*path*, **const** std::vector<uint8_t> &*data*, **const** std::vector<eip::*CommonPacketItem*> &*additionalPacketItems*) **const**
Sends an explicit requests to the EIP adapter by calling a CIP service.

Return the received response from the EIP adapter

Parameters

- `si`: the EIP session with the adapter
- `service`: the service code (for standard codes see eipScanner::cip::ServiceCodes)
- `path`: the path to an element in Object Model that provides the called service
- `data`: the encoded arguments of the service
- `additionalPacketItems`: (needed only for *eipScanner::ConnectionManager*)

Exceptions

- `std::runtime_error`:
- `std::system_error`:

MessageRouterResponse **sendRequest**(*SessionInfoIf*::*SPtr* *si*, cip::*CipUsint* *service*, **const** cip::*EPath* &*path*, **const** std::vector<uint8_t> &*data*) **const**
Sends an explicit requests to the EIP adapter by calling a CIP service.

Return the received response from the EIP adapter

Parameters

- `si`: the EIP session with the adapter
- `service`: the service code (for standard codes see eipScanner::cip::ServiceCodes)
- `path`: the path to an element in Object Model that provides the called service
- `data`: the encoded arguments of the service

Exceptions

- `std::runtime_error`:
- `std::system_error`:

MessageRouterResponse **sendRequest**(*SessionInfoIf*::*SPtr* *si*, cip::*CipUsint* *service*, **const** cip::*EPath* &*path*) **const**

Sends an explicit requests to the EIP adapter by calling a CIP service.

Return the received response from the EIP adapter

Parameters

- `si`: the EIP session with the adapter
- `service`: the service code (for standard codes see eipScanner::cip::ServiceCodes)
- `path`: the path to an element in Object Model that provides the called service

Exceptions

- `std::runtime_error`:
- `std::system_error`:

Public Static Attributes

constexpr bool **USE_8_BIT_PATH_SEGMENTS** = true

Class ParameterObject

- Defined in file_src_ParameterObject.h

Inheritance Relationships

Base Type

- `public eipScanner::BaseObject` (*Class BaseObject*)

Class Documentation

class ParameterObject : **public** eipScanner::*BaseObject*

Implements interface to Parameter Object (0x0F).

It reads all data from the CIP instance in the constructor

Public Functions

ParameterObject(cip::*CipUint* *instanceId*, bool *fullAttributes*, **const** *SessionInfoIf*::*SPtr* &*si*)

Creates an instance and reads all its data via EIP.

Parameters

- `instanceId`: the ID of the CIP instance
- `fullAttributes`: if true, then read all the attributes including scaling attributes and text descriptions
- `si`: the EIP session for explicit messaging

Exceptions

- `std::runtime_error`:
- `std::system_error`:

ParameterObject(cip::*CipUint instanceId*, bool *fullAttributes*, size_t *typeSize*)
Creates an empty instance without any EIP requests.

Parameters

- `instanceId`: the ID of the CIP instance
- `fullAttributes`: if true, then read all the attributes including scaling attributes and text descriptions
- `typeSize`: the size of the value in bytes

Exceptions

- `std::runtime_error`:
- `std::system_error`:

ParameterObject(cip::*CipUint instanceId*, bool *fullAttributes*, **const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*)
Creates an instance and reads all its data via EIP.

Note Used for testing

Parameters

- `instanceId`:
- `fullAttributes`:
- `si`:

~ParameterObject()
Default destructor.

void **updateValue**(**const** *SessionInfoIf*::*SPtr* &*si*)
Updates the parameter value from the instance.

Parameters

- `si`: the Session Info for explicit messaging

Exceptions

- `std::runtime_error`:
- `std::system_error`:

bool **isScalable**() **const**

Return true if the parameter supports scaling

void **setScalable**(bool *isScalable*)

Parameters

- `isScalable`: true if the parameter supports scaling

bool **isReadOnly**() **const**

Return true if the parameter value is read only

void **setReadOnly**(bool *isReadOnly*)

Parameters

- `isReadOnly`: true if the parameter value is read only

template<typename **T**>
T **getActualValue**() **const**

Gets an actual value [AttrID=1] of the parameter.

Note This is just a getter. To read value from EIP device, use *ParameterObject::updateValue*

Return the value of type T

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
cip::*CipLreal* **getEngValue**() **const**

Gets a value of the parameter in EU For scaling the method uses scalling attributes (Multiplier [AttrID=13], Divisor [AttrID=14], Base [AttrID=15], Offset[ID=16] and Precision [AttrID=21].

The formula is: Value in EU = ((Actual Value + Offset)*Multiplier*Base)/(Divisor*10^Precision)

Note it scales the actual value if *ParameterObject::isScalable* is true, else it returns the original actual value

Return the value in EU

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
T **getMinValue**() **const**

Gets a minimal value [AttrID=10] of the parameter.

Note The behavior is the same as *ParameterObject::getActualValue*

Return the value of type T

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
cip::*CipLreal* **getEngMinValue**() **const**

Gets a minimal value of the parameter in EU.

Note The behavior is the same as *ParameterObject::getEngValue*

Return the value in EU

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
void **setEngMinValue**(cip::*CipLreal value*)
Sets a minimal value of the parameter in EU.

Template Parameters

- `T`: the type of the parameter

Parameters

- `value`:

template<typename **T**>
T **getMaxValue**() **const**
Gets a maximal value [AttrID=11] of the parameter.

Note The behavior is the same as *ParameterObject::getActualValue*

Return the value of type T

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
cip::*CipLreal* **getEngMaxValue**() **const**
Gets a maximal value of the parameter in EU.

Note The behavior is the same as *ParameterObject::getEngValue*

Return the value in EU

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
void **setEngMaxValue**(cip::*CipLreal value*)
Sets a maximal value of the parameter in EU.

Template Parameters

- `T`: the type of the parameter

Parameters

- `value`:

template<typename **T**>
T **getDefaultValue**() **const**
Gets an default value [AttrID=12] of the parameter.

Note The behavior is the same as *ParameterObject::getActualValue*

Return the value of type T

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
cip::*CipLreal* **getEngDefaultValue**() **const**
Gets a default value of the parameter in EU.

Note The behavior is the same as *ParameterObject::getEngValue*

Return the value in EU

Template Parameters

- `T`: the type of the parameter

template<typename **T**>
void **setEngDefaultValue**(cip::*CipLreal* *value*)
Sets a default value of the parameter in EU.

Template Parameters

- `T`: the type of the parameter

Parameters

- `value`:

bool **hasFullAttributes**() **const**

Return true if the parameter supports full data including scaling attributes and text descriptions

cip::*CipDataTypes* **getType**() **const**
Gets the type code [AttrID=5] of the parameter.

Return

void **setType**(cip::*CipDataTypes* *type*)
Sets the type code [AttrID=5] of the parameter.

Parameters

- `type`:

const std::string &**getName**() **const**
Gets the name [AttrID=7] of the parameter.

Return

const std::string &**getUnits**() **const**
Gets the units [AttrID=8] of the parameter.

Return

const std::string &**getHelp**() **const**
Gets the help string [AttrID=9] of the parameter.

Return

const cip::*CipUint* &**getParameter**() **const**
Gets the number of parameter (instance ID)

Return

void **setName**(**const** std::string &*name*)
Sets the name [AttrID=7] of the parameter.

Parameters

- name:

void **setUnits**(**const** std::string &*units*)
Sets the units [AttrID=8] of the parameter.

Parameters

- units:

void **setHelp**(**const** std::string &*help*)
Sets the help string [AttrID=9] of the parameter.

Parameters

- help:

CipUint **getScalingMultiplier**() **const**
Gets the multiplier [AttrID=13] of the parameter.

Return

CipUint **getScalingDivisor**() **const**
Gets the divisor [AttrID=14] of the parameter.

Return

CipUint **getScalingBase**() **const**
Gets the base [AttrID=15] of the parameter.

Return

CipInt **getScalingOffset**() **const**
Gets the offset [AttrID=16] of the parameter.

Return

CipUsint **getPrecision**() **const**
Gets the precision [AttrID=21] of the parameter.

Return

void **setScalingMultiplier**(cip::*CipUint scalingMultiplier*)
Sets the multiplier [AttrID=13] of the parameter.

Parameters

- scalingMultiplier:

void **setScalingDivisor**(cip::*CipUint scalingDivisor*)
Sets the divisor [AttrID=14] of the parameter.

Parameters

- `scalingDivisor`:

void **setScalingBase**(cip::*CipUint scalingBase*)
Sets the base [AttrID=15] of the parameter.

Parameters

- `scalingBase`:

void **setScalingOffset**(cip::*CipInt scalingOffset*)
Sets the offset [AttrID=16] of the parameter.

Parameters

- `scalingOffset`:

void **setPrecision**(cip::*CipUsint precision*)
Sets the precision [AttrID=21] of the parameter.

Parameters

- `precision`:

cip::*CipLreal* **actualToEngValue**(cip::*CipLreal actualValue*) **const**

cip::*CipLreal* **engToActualValue**(cip::*CipLreal engValue*) **const**

Public Static Attributes

const cip::*CipUint* **CLASS_ID** = 0x0f

Class SessionInfo

- Defined in file_src_SessionInfo.h

Inheritance Relationships

Base Type

- `public eipScanner::SessionInfoIf` (*Class SessionInfoIf*)

Class Documentation

class SessionInfo : **public** eipScanner::*SessionInfoIf*
Implementation of EIP session.

Public Types

using SPtr = std::shared_ptr<*SessionInfo*>

Public Functions

SessionInfo(**const** std::string &*host*, int *port*, **const** std::chrono::milliseconds &*timeout*)
Establishes an EIP session with an EIP adapter.

Parameters

- `host`: The IP address of the adapter
- `port`: The port of the adapter
- `timeout`: timout to connect and receive the response

Exceptions

- `std::runtime_error`:
- `std::system_error`:

SessionInfo(**const** std::string &*host*, int *port*)
Establishes an EIP session with an EIP adapter.

Parameters

- `host`: The IP address of the adapter
- `port`: The port of the adapter

Exceptions

- `std::runtime_error`:
- `std::system_error`:

~SessionInfo()
Default destructor.

EncapsPacket **sendAndReceive**(**const** eip::*EncapsPacket* &*packet*) **const**

See *SessionInfo::sendAndReceive*

Return

Parameters

- `packet`:

cip::*CipUdint* **getSessionHandle**() **const**

See *SessionInfo::getSessionHandle*

Return

sockets::*EndPoint* **getRemoteEndPoint**() **const**

See *SessionInfo::getRemoteEndPoint*

Return

Class SessionInfoIf

- Defined in file_src_SessionInfoIf.h

Inheritance Relationships

Derived Type

- `public eipScanner::SessionInfo` (*Class SessionInfo*)

Class Documentation

class SessionInfoIf

Abstract interface for EIP session.

Subclassed by *eipScanner::SessionInfo*

Public Types

using SPtr = std::shared_ptr<*SessionInfoIf*>

Public Functions

virtual eip::*EncapsPacket* **sendAndReceive**(**const** eip::*EncapsPacket* &*packet*) **const** = 0

Sends and receives EIP Encapsulation packet

Return the received EIP Encapsulation packet

Parameters

- `packet`: the EIP Encapsulation packet to send

virtual cip::*CipUdint* **getSessionHandle**() **const** = 0

Gets the handle of the current EIP session

Return

virtual sockets::*EndPoint* **getRemoteEndPoint**() **const** = 0

Gets the address of the EIP adapter which the session is established with

Return

Class BaseSocket

- Defined in file_src_sockets_BaseSocket.h

Inheritance Relationships

Derived Types

- `public eipScanner::sockets::TCPSocket` (*Class TCPSocket*)
- `public eipScanner::sockets::UDPSocket` (*Class UDPSocket*)

Class Documentation

class BaseSocket

Subclassed by *eipScanner::sockets::TCPSocket*, *eipScanner::sockets::UDPSocket*

Public Types

using BeginReceiveHandler = std::function<void (*BaseSocket*&)>

using SPtr = std::shared_ptr<*BaseSocket*>

using UPtr = std::unique_ptr<*BaseSocket*>

Public Functions

BaseSocket(*EndPoint endPoint*)

BaseSocket(std::string *host*, int *port*)

~BaseSocket()

virtual void **Send**(**const** std::vector<uint8_t> &*data*) **const** = 0

virtual std::vector<uint8_t> **Receive**(size_t *size*) **const** = 0

void **setBeginReceiveHandler**(*BeginReceiveHandler handler*)

const std::chrono::milliseconds &**getRecvTimeout**() **const**

void **setRecvTimeout**(**const** std::chrono::milliseconds &*recvTimeout*)

int **getSocketFd**() **const**

const *EndPoint* &**getRemoteEndPoint**() **const**

Public Static Functions

int **getLastError**()

const std::error_category &**getErrorCategory**()

void **select**(std::vector<*BaseSocket*::*SPtr*> *sockets*, std::chrono::milliseconds *timeout*)

Protected Functions

void **BeginReceive**()

void **Shutdown**()

void **Close**()

Protected Attributes

int **_sockedFd**

EndPoint **_remoteEndPoint**

std::chrono::milliseconds **_recvTimeout**

BeginReceiveHandler **_beginReceiveHandler**

Protected Static Functions

timeval **makePortableInterval**(**const** std::chrono::milliseconds &*recvTimeout*)

Class EndPoint

- Defined in file_src_sockets_EndPoint.h

Class Documentation

class EndPoint

Public Functions

EndPoint(std::string *host*, int *port*)

EndPoint(**struct** sockaddr_in &*addr*)

const std::string &**getHost**() **const**

int **getPort**() **const**

const sockaddr_in &**getAddr**() **const**

std::string **toString**() **const**

bool **operator==**(**const** EndPoint &*rhs*) **const**

bool **operator!=**(**const** EndPoint &*rhs*) **const**

bool **operator<**(**const** EndPoint &*rhs*) **const**

Class TCPSocket

- Defined in file_src_sockets_TCPSocket.h

Inheritance Relationships

Base Type

- `public eipScanner::sockets::BaseSocket` (*Class BaseSocket*)

Class Documentation

class TCPSocket : **public** eipScanner::sockets::*BaseSocket*

Public Functions

TCPSocket(*EndPoint endPoint*)

TCPSocket(*EndPoint endPoint*, std::chrono::milliseconds *connTimeout*)

TCPSocket(std::string *host*, int *port*)

~TCPSocket()

void **Send**(**const** std::vector<uint8_t> &*data*) **const**

std::vector<uint8_t> **Receive**(size_t *size*) **const**

Class UDPBoundSocket

- Defined in file_src_sockets_UDPBoundSocket.h

Inheritance Relationships

Base Type

- `public eipScanner::sockets::UDPSocket` (*Class UDPSocket*)

Class Documentation

class UDPBoundSocket : **public** eipScanner::sockets::*UDPSocket*

Public Types

using WPtr = std::weak_ptr<*UDPBoundSocket*>

using SPtr = std::shared_ptr<*UDPBoundSocket*>

Public Functions

UDPBoundSocket (*EndPoint endPoint*)

UDPBoundSocket (std::string *host*, int *port*)

~UDPBoundSocket ()

Class UDPSocket

- Defined in file_src_sockets_UDPSocket.h

Inheritance Relationships

Base Type

- `public eipScanner::sockets::BaseSocket` (*Class BaseSocket*)

Derived Type

- `public eipScanner::sockets::UDPBoundSocket` (*Class UDPBoundSocket*)

Class Documentation

class UDPSocket : **public** eipScanner::sockets::*BaseSocket*
Subclassed by *eipScanner::sockets::UDPBoundSocket*

Public Types

using WPtr = std::weak_ptr<*UDPSocket*>

using SPtr = std::shared_ptr<*UDPSocket*>

using UPtr = std::unique_ptr<*UDPSocket*>

Public Functions

UDPSocket (*EndPoint endPoint*)

UDPSocket (std::string *host*, int *port*)

~UDPSocket ()

void **Send** (**const** std::vector<uint8_t> &*data*) **const**

std::vector<uint8_t> **Receive** (size_t *size*) **const**

std::vector<uint8_t> **ReceiveFrom** (size_t *size*, *EndPoint* &*endPoint*) **const**

Class Buffer

- Defined in file_src_utils_Buffer.h

Class Documentation

class Buffer

Implements decode and encode data according CIP specification.

An example:

```
Buffer buffer1();
cip::CipUint var1 = 1;
cip::CipDint var2 = 0xaa00000;

buffer1 << var1 << var2;

buffer1.data(); # => {0x01, 0x0, 0x0 ,0x0, 0x0, 0xaa}
```

Public Functions

Buffer(size_t *capacity*)

Creates an empty buffer

Parameters

- `capacity`: the size that will be reserved in the buffer

Buffer(**const** std::vector<uint8_t> &*data*)

Creates a buffer that contains the given data

Parameters

- `data`: The data to encode

Buffer()

Creates an empty buffer

Buffer &**operator<<**(uint8_t *val*)

Buffer &**operator>>**(uint8_t &*val*)

Buffer &**operator<<**(int8_t *val*)

Buffer &**operator>>**(int8_t &*val*)

Buffer &**operator<<**(uint16_t *val*)

Buffer &**operator>>**(uint16_t &*val*)

Buffer &**operator<<**(int16_t *val*)

Buffer &**operator>>**(int16_t &*val*)

Buffer &**operator<<**(uint32_t *val*)

Buffer &**operator>>**(uint32_t &*val*)

Buffer &**operator<<**(int32_t *val*)

Buffer &**operator>>**(int32_t &*val*)

Buffer &**operator<<**(uint64_t *val*)

Buffer &**operator>>**(uint64_t &*val*)

Buffer &**operator<<**(int64_t *val*)

Buffer &**operator>>**(int64_t &*val*)

Buffer &**operator<<**(float *val*)

Buffer &**operator>>**(float &*val*)

Buffer &**operator<<**(double *val*)

Buffer &**operator>>**(double &*val*)

Buffer &**operator<<**(**const** std::vector<uint8_t> &*val*)

Buffer &**operator>>**(std::vector<uint8_t> &*val*)

Buffer &**operator<<**(**const** std::vector<uint16_t> &*val*)

Buffer &**operator>>**(std::vector<uint16_t> &*val*)

template<typename **T**>
utils::*Buffer* &**operator<<**(**const** cip::CipBaseString<*T*> &*cipSting*)

template<typename **T**>
utils::*Buffer* &**operator>>**(cip::CipBaseString<*T*> &*cipSting*)

Buffer &**operator<<**(cip::*CipRevision* *v*)

Buffer &**operator>>**(cip::*CipRevision* &*val*)

Buffer &**operator<<**(sockets::*EndPoint* *v*)

Buffer &**operator>>**(sockets::*EndPoint* &*val*)

std::vector<uint8_t> **data**() **const**

size_t **size**() **const**

size_t **pos**() **const**

bool **isValid**() **const**

bool **empty**() **const**

Class ConsoleAppender

- Defined in file_src_utils_Logger.h

Inheritance Relationships

Base Type

- `public eipScanner::utils::LogAppenderIf` (*Class LogAppenderIf*)

Class Documentation

class ConsoleAppender : **public** eipScanner::utils::*LogAppenderIf*

Implements out log messages to std::cout.

Public Types

using UPtr = std::unique_ptr<*LogAppenderIf*>

Public Functions

void **print**(*LogLevel logLevel*, **const** std::string &*msg*)

Class LogAppenderIf

- Defined in file_src_utils_Logger.h

Inheritance Relationships

Derived Type

- `public eipScanner::utils::ConsoleAppender` (*Class ConsoleAppender*)

Class Documentation

class LogAppenderIf

Interface to print message in the logger.

See *Logger*

Subclassed by *eipScanner::utils::ConsoleAppender*

Public Types

using UPtr = std::unique_ptr<*LogAppenderIf*>

Public Functions

virtual ~LogAppenderIf()

virtual void **print**(*LogLevel logLevel*, **const** std::string &*msg*) = 0

Class Logger

- Defined in file_src_utils_Logger.h

Class Documentation

class Logger

Public Functions

Logger(*LogLevel level*)

template<typename **T**>
std::ostringstream &**operator<<**(*T msg*)
Add message to the log.

Return

Template Parameters

- `T`: type of the data to print

Parameters

- `msg`: The message to print

~Logger()
Default destructor.

The destructor prints all messages, that were added by << operator, before the logger are destroyed

Public Static Functions

void **setLogLevel**(*LogLevel level*)
Sets the lowest log level for all log messages.

Note to set off all logs use LogLevel::OFF

Parameters

- `level`:

void **setAppender**(*LogAppenderIf*::*UPtr appender*)
Sets appender to print messages for all log messages.

The default appender is *ConsoleAppender*

Parameters

- `appender`:

Class DPIFaultCode

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultCode.h

Nested Relationships

Nested Types

- *Struct DPIFaultCode::FaultDescriptions*

Class Documentation

class DPIFaultCode

Public Functions

DPIFaultCode(int *faultCode*)

~DPIFaultCode()

DPIFaultCode::*FaultDescriptions* **getFaultDescription**()

struct FaultDescriptions

Public Members

int **faultType**

string **faultText**

string **faultDescription**

Class DPIFaultManager

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultManager.h

Class Documentation

class DPIFaultManager

Implements a manager to retrieve new faults and clean its queue.

It use PrarameterObejcts instead of FaultObject because it doesn't contain the needed information

Public Types

using NewFaultObjectHandler = std::function<void (**const** *DPIFaultObject* &fault) >

using NewFaultHandler = std::function<void (**const** *DPIFaultParameter* &fault) >

using TrippedDeviceHandler = std::function<void (bool) >

Public Functions

DPIFaultManager ()
Default constructor (clearFaults = true, resetDevice = false, getFaultDetails = false)

DPIFaultManager (bool *clearFaults*, bool *resetDevice*, bool *getFaultDetails*)
Constructor.

Parameters

- `clearFaults`: if true the manager clears the queue after it has retrieved a new fault
- `resetDevice`: isn't used yet
- `getFaultDetails`: if true the manager read all data from fault parameters

void **setNewFaultListener** (*NewFaultHandler handler*)
Sets a callback to receive a new fault.

Parameters

- `handler`:

void **setTrippedDeviceListener** (*TrippedDeviceHandler handler*)
Sets a callback if the device changed trip-state.

Parameters

- `handler`:

void **handleFaultParameters** (**const** *SessionInfoIf*::*SPtr* &*si*)
reads fault parameters and calls NewFaultHandler handler if it gets a new one

Parameters

- `si`:

void **handleFaultParameters** (**const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*)

void **writeCommand** (*DPIFaultManagerCommands command*, **const** *SessionInfoIf*::*SPtr* &*si*)
const
Writes a command to DPI Fault Manager (e.g. clean fault or reset device)

See DPIFaultManagerCommands

Parameters

- `command`: the command to send
- `si`: the EIP session for explicit messaging

void **writeCommand**(*DPIFaultManagerCommands command*, **const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*) **const**

Note used for testing

Parameters

- `command`:
- `si`:
- `messageRouter`:

Class DPIFaultObject

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultObject.h

Nested Relationships

Nested Types

- *Struct DPIFaultObject::FullInformation*

Inheritance Relationships

Base Type

- `public eipScanner::BaseObject` (*Class BaseObject*)

Class Documentation

class DPIFaultObject : **public** eipScanner::*BaseObject*

Implements interface to DPI Fault Object (0x97) of PowerFlex 525.

Public Functions

DPIFaultObject(cip::*CipUint instanceId*, **const** *SessionInfoIf*::*SPtr* &*si*)

Creates an instance and reads all its data via EIP.

Parameters

- `instanceId`:
- `fullAttributes`: if true, then read all the attributes
- `si`: the EIP session for explicit messaging

DPIFaultObject(cip::*CipUint instanceId*, **const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*)

Note used for testing

Parameters

- instanceId:
- si:
- messageRouter:

const *DPIFaultObject*::*FullInformation* &**getFullInformation**() **const**
Gets the full information [AttrID=1] of the fault.

Return

Public Static Attributes

const cip::*CipUint* **CLASS_ID** = 0x97

struct FullInformation
Informaion about the fault

Public Members

cip::*CipUint* **faultCode**
the code of the fault (0 is no fault)

cip::*CipUsint* **dsiPort**
DSI port.

cip::*CipUsint* **dsiDeviceObject**
DSI Device Object.

cip::*CipString* **faultText**
the text of the fault

cip::*CipLword* **timerValue**
timer value

bool **isValidData**
true if the timer value valid

bool **isRealTime**
true if the time is real else it is elapsed

Class DPIFaultParameter

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.h

Nested Relationships

Nested Types

- *Struct DPIFaultParameter::FaultDetails*
- *Struct DPIFaultParameter::FullInformation*

Class Documentation

class DPIFaultParameter

Public Functions

DPIFaultParameter(**const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*, int *faultNumber*, bool *getFaultDetails*)

DPIFaultParameter()

const *DPIFaultParameter*::*FullInformation* &**getFullInformation**() **const**

const *DPIFaultParameter*::*FaultDetails* &**getFaultDetails**() **const**

void **setFaultDetails**(*FaultDetails faultInfo*)

void **setFaultDescription**(*DPIFaultCode*::*FaultDescriptions faultDescriptions*)

struct FaultDetails

Public Members

int **faultNumber**

cip::*CipUint* **faultCode**

cip::*CipLreal* **busVoltage**

cip::*CipLreal* **current**

cip::*CipLreal* **frequency**

struct FullInformation

Public Members

FaultDetails **faultDetails**

DPIFaultCode::*FaultDescriptions* **faultDescription**

Class Yaskawa_MessageRouter

- Defined in file_src_vendor_yaskawa_mp3300iec_Yaskawa_MessageRouter.h

Class Documentation

class Yaskawa_MessageRouter

Public Types

using SPtr = std::shared_ptr<*Yaskawa_MessageRouter*>

Public Functions

Yaskawa_MessageRouter()
Default constructor.

~Yaskawa_MessageRouter()
Default destructor.

MessageRouterResponse **sendRequest**(*SessionInfoIf*::*SPtr* *si*, cip::*CipUsint* *service*, **const** cip::*Yaskawa_EPath* &*path*, **const** std::vector<uint8_t> &*data*, **const** std::vector<eip::*CommonPacketItem*> &*additionalPacketItems*) **const**
Sends an explicit requests to the EIP adapter by calling a CIP service.

Return the received response from the EIP adapter

Parameters

- `si`: the EIP session with the adapter
- `service`: the service code (for standard codes see eipScanner::cip::ServiceCodes)
- `path`: the path to an element in Object Model that provides the called service
- `data`: the encoded arguments of the service
- `additionalPacketItems`: (needed only for *eipScanner::ConnectionManager*)

Exceptions

- `std::runtime_error`:
- `std::system_error`:

MessageRouterResponse **sendRequest**(*SessionInfoIf*::*SPtr* *si*, cip::*CipUsint* *service*, **const** cip::*Yaskawa_EPath* &*path*, **const** std::vector<uint8_t> &*data*) **const**
Sends an explicit requests to the EIP adapter by calling a CIP service.

Return the received response from the EIP adapter

Parameters

- `si`: the EIP session with the adapter
- `service`: the service code (for standard codes see eipScanner::cip::ServiceCodes)
- `path`: the path to an element in Object Model that provides the called service
- `data`: the encoded arguments of the service

Exceptions

- `std::runtime_error`:
- `std::system_error`:

MessageRouterResponse **sendRequest**(*SessionInfoIf*::*SPtr* *si*, cip::*CipUsint* *service*, **const** cip::*Yaskawa_EPath* &*path*) **const**
Sends an explicit requests to the EIP adapter by calling a CIP service.

Return the received response from the EIP adapter

Parameters

- `si`: the EIP session with the adapter
- `service`: the service code (for standard codes see eipScanner::cip::ServiceCodes)
- `path`: the path to an element in Object Model that provides the called service

Exceptions

- `std::runtime_error`:
- `std::system_error`:

Enums

Enum CipDataTypes

- Defined in file_src_cip_Types.h

Enum Documentation

enum `eipScanner::cip::`**CipDataTypes**

Values:

ANY = 0x00
data type that can not be directly encoded

BOOL = 0xC1
boolean data type

SINT = 0xC2
8-bit signed integer

INT = 0xC3
16-bit signed integer

DINT = 0xC4
32-bit signed integer

LINT = 0xC5
64-bit signed integer

USINT = 0xC6
8-bit unsigned integer

UINT = 0xC7
16-bit unsigned integer

UDINT = 0xC8
32-bit unsigned integer

ULINT = 0xC9
64-bit unsigned integer

REAL = 0xCA
Single precision floating point

LREAL = 0xCB
Double precision floating point

STIME = 0xCC
Synchronous time information*, type of DINT

DATE = 0xCD
Date only

DATE_OF_DAY = 0xCE
Time of day

DATE_AND_TIME = 0xCF
Date and time of day

STRING = 0xD0
Character string, 1 byte per character

BYTE = 0xD1
8-bit bit string

WORD = 0xD2
16-bit bit string

DWORD = 0xD3
32-bit bit string

LWORD = 0xD4
64-bit bit string

STRING2 = 0xD5
Character string, 2 byte per character

FTIME = 0xD6
Duration in micro-seconds, high resolution; range of DINT

LTIME = 0xD7
Duration in micro-seconds, high resolution, range of LINT

ITIME = 0xD8
Duration in milli-seconds, short; range of INT

STRINGN = 0xD9
Character string, N byte per character

SHORT_STRING = 0xDA
Character string, 1 byte per character, 1 byte length indicator

TIME = 0xDB
Duration in milli-seconds; range of DINT

EPATH = 0xDC
CIP path segments

ENG_UNIT = 0xDD
Engineering Units

USINT_USINT = 0xA0
Used for CIP Identity attribute 4 Revision

USINT6 = 0xA2
Struct for MAC Address (six USINTs)

MEMBER_LIST = 0xA3

BYTE_ARRAY = 0xA4

Enum NetworkConnectionParams

- Defined in file_src_cip_connectionManager_NetworkConnectionParams.h

Enum Documentation

enum eipScanner::cip::connectionManager::**NetworkConnectionParams**

Values:

REDUNDANT = (1 << 15)

OWNED = 0

TYPE0 = 0

MULTICAST = (1 << 13)

P2P = (2 << 13)

LOW_PRIORITY = 0

HIGH_PRIORITY = (1 << 10)

SCHEDULED_PRIORITY = (2 << 10)

URGENT = (3 << 10)

FIXED = 0

VARIABLE = (1 << 9)

TRIG_CYCLIC = 0

TRIG_CHANGE = (1 << 4)

TRIG_APP = (2 << 4)

CLASS0 = 0

CLASS1 = 1

CLASS2 = 2

CLASS3 = 3

TRANSP_SERVER = 0x80

Enum EPathSegmentTypes

- Defined in file_src_cip_EPath.cpp

Enum Documentation

enum eipScanner::cip::**EPathSegmentTypes**

Values:

CLASS_8_BITS = 0x20

CLASS_16_BITS = 0x21

INSTANCE_8_BITS = 0x24

INSTANCE_16_BITS = 0x25

ATTRIBUTE_8_BITS = 0x30

ATTRIBUTE_16_BITS = 0x31

CLASS_8_BITS = 0x20

CLASS_16_BITS = 0x21

INSTANCE_8_BITS = 0x24

INSTANCE_16_BITS = 0x25

ATTRIBUTE_8_BITS = 0x30

ATTRIBUTE_16_BITS = 0x31

Enum EPathSegmentTypes

- Defined in file_src_vendor_yaskawa_mp3300iec_Yaskawa_EPath.cpp

Enum Documentation

enum eipScanner::cip::**EPathSegmentTypes**

Values:

CLASS_8_BITS = 0x20

CLASS_16_BITS = 0x21

INSTANCE_8_BITS = 0x24

INSTANCE_16_BITS = 0x25

ATTRIBUTE_8_BITS = 0x30

ATTRIBUTE_16_BITS = 0x31

CLASS_8_BITS = 0x20

CLASS_16_BITS = 0x21

INSTANCE_8_BITS = 0x24

INSTANCE_16_BITS = 0x25

ATTRIBUTE_8_BITS = 0x30

ATTRIBUTE_16_BITS = 0x31

Enum GeneralStatusCodes

- Defined in file_src_cip_GeneralStatusCodes.h

Enum Documentation

enum eipScanner::cip::**GeneralStatusCodes**

Values:

SUCCESS = 0x00
Service was successfully performed.

CONNECTION_FAILURE = 0x01
A connection related service failed along the connection path.

RESOURCE_UNAVAILABLE = 0x02
Resources needed for the object to perform the requested service were unavailable.

INVALID_PARAMETER_VALUE = 0x03
See CIPStatusCodes.InvalidParameter, which is the preferred value to use for this condition.

PATH_SEGMENT_ERROR = 0x04
The path segment identifier or the segment syntax was not understood by the processing node. Path processing shall stop when a path segment error is encountered.

PATH_DESTINATION_UNKNOWN = 0x05
The path is referencing an object class, instance, or structure element that is not known or is not contained in the processing node. Path processing shall stop when a path destination unknown error is encountered.

PARTIAL_TRANSFER = 0x06
Only part of the expected data was transferred.

CONNECTION_LOST = 0x07
The messaging connection was lost.

SERVICE_NOT_SUPPORTED = 0x08
The requested service was not implemented or was not defined for this object Class/Instance.

INVALID_ATTRIBUTE_VALUE = 0x09
Invalid attribute data detected.

ATTRIBUTE_LIST_ERROR = 0x0A
An attribute in the Get_Attribute_List or Set_Attribute_List response has a non-zero status.

ALREADY_IN_REQUESTED_MODE_OR_STATE = 0x0B
The object is already in the mode/state being requested by the service.

OBJECT_STATE_CONFLICT = 0x0C
The object cannot perform the requested service in its current state/mode.

OBJECT_ALREADY_EXISTS = 0x0D
The requested instance of object to be created already exists.

ATTRIBUTE_NOT_SETTABLE = 0x0E
A request to modify a non-modifiable attribute was received.

PRIVILEGE_VIOLATION = 0x0F
A permission/privilege check failed.

DEVICE_STATE_CONFLICT = 0x10
The device's current mode/state prohibits the execution of the requested service.

REPLY_DATA_TOO_LARGE = 0x11
The data to be transmitted in the response buffer is larger than the allocated response buffer.

FRAGMENTATION_OF_PRIMITIVE_VALUE = 0x12
The service specified an operation that is going to fragment a primitive data value, i.e. half a REAL data type.

NOT_ENOUGH_DATA = 0x13
The service did not supply enough data to perform the requested operation.

ATTRIBUTE_NOT_SUPPORTED = 0x14
The attribute specified in the request is not supported.

TOO_MUCH_DATA = 0x15
The service was supplied with more data than was expected.

OBJECT_DOES_NOT_EXIST = 0x16
The object specified does not exist on the device.

SVCFRAG_SEQNC_NOT_IN_PROGRESS = 0x17
The fragmentation sequence for this service is not currently active for this data.

NO_STORED_ATTRIBUTE_DATA = 0x18
The attribute data of this object was not saved prior to the requested service.

STORE_OPERATION_FAILURE = 0x19
The attribute data of this object was not saved due to a failure following the attempt.

ROUTING_FAILURE_REQUEST_SIZE = 0x1A
The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service.

ROUTING_FAILURE_RESPONSE_SIZE = 0x1B
The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to abort the service.

MISSING_ATTRIBUTE_LIST_ENTRY = 0x1C
The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behavior.

INVALID_ATTRIBUTE_LIST = 0x1D
The service is returning the list of attributes supplied with status information for those attributes that were invalid.

EMBEDDED_SERVICE_ERROR = 0x1E
An embedded service resulted in an error.

VENDOR_SPECIFIC = 0x1F
A vendor specific error has been encountered. The Additional Code Field of the Error Response defines the particular error encountered. Use of this General Error Code should only be performed when none of the Error Codes presented in this table or within an Object Class definition accurately reflect the error.

INVALID_PARAMETER = 0x20
A parameter associated with the request was invalid. This code is used when a parameter does not meet the requirements of this specification and/or the requirements defined in an Application Object Specification.

WRITE_ONCE_WRITTEN = 0x21
An attempt was made to write to a write-once medium (e.g. WORM drive, PROM) that has already been written, or to modify a value that cannot be changed once established.

INVALID_REPLY_RECEIVED = 0x22
An invalid reply is received (e.g. reply service code does not match the request service code, or reply message is shorter than the minimum expected reply size). This status code can serve for other causes of invalid replies.

KEY_FAILURE_IN_PATH = 0x25
The Key Segment that was included as the first segment in the path does not match the destination module. The object specific status shall indicate which part of the key check failed.

PATH_SIZE_INVALID = 0x26
The size of the path which was sent with the Service Request is either not large enough to allow the Request to be routed to an object or too much routing data was included.

UNEXPECTED_ATTRIBUTE = 0x27
An attempt was made to set an attribute that is not able to be set at this time.

INVALID_MEMBER_ID = 0x28
The Member ID specified in the request does not exist in the specified Class/Instance/Attribute

MEMBER_NOT_SETTABLE = 0x29
A request to modify a non-modifiable member was received.

Enum ServiceCodes

- Defined in file_src_cip_Services.h

Enum Documentation

enum eipScanner::cip::**ServiceCodes**
Values:

NONE = 0x00

GET_ATTRIBUTE_ALL = 0X01

SET_ATTRIBUTE_ALL = 0X02

GET_ATTRIBUTE_LIST = 0x03

SET_ATTRIBUTE_LIST = 0x04

RESET = 0x05

START = 0x06

STOP = 0x07

CREATE_OBJECT_INSTANCE = 0x08

DELETE_OBJECT_INSTANCE = 0x09

MULTIPLE_SERVICE_PACKET = 0x0A

APPLY_ATTRIBUTES = 0x0D

GET_ATTRIBUTE_SINGLE = 0X0E

SET_ATTRIBUTE_SINGLE = 0X10

FIND_NEXT_OBJECT_INSTANCE = 0x11

ERROR_RESPONSE = 0x14

RESTORE = 0x15

SAVE = 0x16

GET_MEMBER = 0x18

NO_OPERATION = 0x17

SET_MEMBER = 0x19

INSERT_MEMBER = 0x1A

REMOVE_MEMBER = 0x1B

GROUP_SYNC = 0x1C

Enum ConnectionManagerServiceCodes

- Defined in file_src_ConnectionManager.cpp

Enum Documentation

enum eipScanner::**ConnectionManagerServiceCodes**

Values:

FORWARD_OPEN = 0x54

LARGE_FORWARD_OPEN = 0x5B

FORWARD_CLOSE = 0x4E

Enum DescriptorAttributeBits

- Defined in file_src_ParameterObject.cpp

Enum Documentation

enum eipScanner::**DescriptorAttributeBits**

Values:

SUPPORTS_SCALING = 1 << 2

READ_ONLY = 1 << 4

Enum CommonPacketItemIds

- Defined in file_src_eip_CommonPacketItem.h

Enum Documentation

enum eipScanner::eip::**CommonPacketItemIds**

Values:

NULL_ADDR = 0x0000

LIST_IDENTITY = 0x000C

CONNECTION_ADDRESS_ITEM = 0x00A1

CONNECTED_TRANSPORT_PACKET = 0x00B1

UNCONNECTED_MESSAGE = 0x00B2

O2T_SOCKADDR_INFO = 0x8000

T2O_SOCKADDR_INFO = 0x8001

SEQUENCED_ADDRESS_ITEM = 0x8002

Enum EncapsCommands

- Defined in file_src_eip_EncapsPacket.h

Enum Documentation

enum eipScanner::eip::**EncapsCommands**

Values:

NOP = 0

LIST_SERVICES = 0x0004

LIST_IDENTITY = 0x0063

LIST_INTERFACES = 0x0064

REGISTER_SESSION = 0x0065

UN_REGISTER_SESSION = 0x0066

SEND_RR_DATA = 0x006F

SEND_UNIT_DATA = 0x0070

INDICATE_STATUS = 0x0072

CANCEL = 0x0073

Enum EncapsStatusCodes

- Defined in file_src_eip_EncapsPacket.h

Enum Documentation

enum eipScanner::eip::**EncapsStatusCodes**

Values:

SUCCESS = 0x0000

UNSUPPORTED_COMMAND = 0x0001

INSUFFICIENT_MEMORY = 0x0002

INVALID_FORMAT_OR_DATA = 0x0003

INVALID_SESSION_HANDLE = 0x0064

UNSUPPORTED_PROTOCOL_VERSION = 0x0069

Enum FileObjectAttributesCodes

- Defined in file_src_fileObject_FileObjectState.h

Enum Documentation

enum eipScanner::fileObject::**FileObjectAttributesCodes**

Values:

STATE = 1

Enum FileObjectServiceCodes

- Defined in file_src_fileObject_FileObjectState.h

Enum Documentation

enum eipScanner::fileObject::**FileObjectServiceCodes**

Values:

INITIATE_UPLOAD = 0x4B

UPLOAD_TRANSFER = 0x4F

Enum TransferPacketTypeCodes

- Defined in file_src_fileObject_FileObjectState.h

Enum Documentation

enum eipScanner::fileObject::**TransferPacketTypeCodes**

Values:

FIRST = 0

MIDDLE = 1

LAST = 2

ABORT = 3

FIRST_AND_LAST = 4

Enum FileObjectStateCodes

- Defined in file_src_FileObject.h

Enum Documentation

enum eipScanner::**FileObjectStateCodes**
the state codes of File Object

Values:

NONEXISTENT = 0

FILE_EMPTY = 1

FILE_LOADED = 2

TRANSFER_UPLOAD_INITIATED = 3

TRANSFER_DOWNLOAD_INITIATED = 4

TRANSFER_UPLOAD_IN_PROGRESS = 5

TRANSFER_DOWNLOAD_IN_PROGRESS = 6

UNKNOWN = 255

Enum ParameterObjectAttributeIds

- Defined in file_src_ParameterObject.cpp

Enum Documentation

enum eipScanner::**ParameterObjectAttributeIds**
Values:

VALUE = 1

LINK_PATH_SIZE = 2

DESCRIPTOR = 4

DATA_TYPE = 5

DATA_SIZE = 6

NAME_STRING = 7

UNIT_STRING = 8

HELP_STRING = 9

MIN_VALUE = 10

MAX_VALUE = 11

DEFAULT_VALUE = 12

SCALING_MULTIPLIER = 13

SCALING_DIVISOR = 14

SCALING_BASE = 15

SCALING_OFFSET = 16

Enum LogLevel

- Defined in file_src_utils_Logger.h

Enum Documentation

enum eipScanner::utils::**LogLevel**

Values:

OFF = 0

ERROR

WARNING

INFO

DEBUG

TRACE

Enum DPIFaultClassAttributeIds

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultManager.cpp

Enum Documentation

enum eipScanner::vendor::ra::powerFlex525::**DPIFaultClassAttributeIds**

Values:

CLASS_REVISION = 1

NUMBER_OF_INSTANCE = 2

FAULT_COMMAND_WRITE = 3

FAULT_TRIP_INSTANCE_READ = 4

FAULT_DATA_LIST = 5

NUMBER_OF_RECORDED_FAULTS = 6

Enum DPIFaultManagerCommands

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultManager.h

Enum Documentation

enum eipScanner::vendor::ra::powerFlex525::**DPIFaultManagerCommands**

Fault Manager command codes

Values:

NO_OPERATION = 0

CLEAR_FAULT = 1

CLEAR_FAULT_QUEUE = 2

RESET_DEVICE = 3

Enum DPIFaultObjectAttributeIds

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultObject.cpp

Enum Documentation

enum eipScanner::vendor::ra::powerFlex525::**DPIFaultObjectAttributeIds**

Values:

FULL_INFORMATION = 0

FULL_INFORMATION = 0

Enum DPIFaultObjectAttributeIds

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.cpp

Enum Documentation

enum eipScanner::vendor::ra::powerFlex525::**DPIFaultObjectAttributeIds**

Values:

FULL_INFORMATION = 0

FULL_INFORMATION = 0

Enum FaultParams

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.cpp

Enum Documentation

enum eipScanner::vendor::ra::powerFlex525::**FaultParams**

Values:

FAULT_1_CODE = 7

FAULT_2_CODE = 8

FAULT_3_CODE = 9

FAULT_4_CODE = 604

FAULT_5_CODE = 605

FAULT_6_CODE = 606

FAULT_7_CODE = 607

FAULT_8_CODE = 608

FAULT_9_CODE = 609

FAULT_10_CODE = 610

FAULT_1_FREQ = 631

FAULT_2_FREQ = 632

FAULT_3_FREQ = 633

FAULT_4_FREQ = 634

FAULT_5_FREQ = 635

FAULT_6_FREQ = 636

FAULT_7_FREQ = 637

FAULT_8_FREQ = 638

FAULT_9_FREQ = 639

FAULT_10_FREQ = 640

FAULT_1_CURR = 641

FAULT_2_CURR = 642

FAULT_3_CURR = 643

FAULT_4_CURR = 644

FAULT_5_CURR = 645

FAULT_6_CURR = 646

FAULT_7_CURR = 647

FAULT_8_CURR = 648

FAULT_9_CURR = 649

FAULT_10_CURR = 650

FAULT_1_BUS_VOLTS = 651

FAULT_2_BUS_VOLTS = 652

FAULT_3_BUS_VOLTS = 653

FAULT_4_BUS_VOLTS = 654

FAULT_5_BUS_VOLTS = 655

FAULT_6_BUS_VOLTS = 656

FAULT_7_BUS_VOLTS = 657

FAULT_8_BUS_VOLTS = 658

FAULT_9_BUS_VOLTS = 659

FAULT_10_BUS_VOLTS = 660

Enum FaultTimeStampFlags

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultObject.cpp

Enum Documentation

enum eipScanner::vendor::ra::powerFlex525::**FaultTimeStampFlags**

Values:

VALID_DATA = 1

REAL_TIME = 1 << 1

VALID_DATA = 1

REAL_TIME = 1 << 1

Enum FaultTimeStampFlags

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.cpp

Enum Documentation

enum eipScanner::vendor::ra::powerFlex525::**FaultTimeStampFlags**

Values:

VALID_DATA = 1

REAL_TIME = 1 << 1

VALID_DATA = 1

REAL_TIME = 1 << 1

Functions

Function eipScanner::cip::logGeneralAndAdditionalStatus

- Defined in file_src_cip_MessageRouterResponse.cpp

Function Documentation

void eipScanner::cip::**logGeneralAndAdditionalStatus**(**const** *MessageRouterResponse* &*response*)

Function eipScanner::vendor::ra::powerFlex525::getFaultDetail

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.cpp

Function Documentation

static uint16_t eipScanner::vendor::ra::powerFlex525::**getFaultDetail**(**const** *SessionInfoIf*::*SPtr* &*si*, **const** *MessageRouter*::*SPtr* &*messageRouter*, int *parameterNumber*)

Function eipScanner::vendor::ra::powerFlex525::processCurrent

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.cpp

Function Documentation

static cip::*CipLreal* eipScanner::vendor::ra::powerFlex525::**processCurrent**(uint16_t *current*, int *currentParam*)

Function eipScanner::vendor::ra::powerFlex525::processFrequency

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.cpp

Function Documentation

static cip::*CipLreal* eipScanner::vendor::ra::powerFlex525::**processFrequency**(uint16_t *frequency*, int *frequencyParam*)

Function eipScanner::vendor::ra::powerFlex525::processVolts

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultParameter.cpp

Function Documentation

static cip::*CipLreal* eipScanner::vendor::ra::powerFlex525::**processVolts**(uint16_t *volts*, int *voltsParam*)

Variables

Variable eipScanner::fileObject::FILE_OBJECT_CLASS_ID

- Defined in file_src_fileObject_FileObjectState.h

Variable Documentation

const cip::*CipUsint* `eipScanner::fileObject::`**FILE_OBJECT_CLASS_ID** = 0x37

Variable eipScanner::fileObject::MAX_TRANSFER_SIZE

- Defined in file_src_fileObject_FileObjectState.h

Variable Documentation

const cip::*CipUsint* `eipScanner::fileObject::`**MAX_TRANSFER_SIZE** = 255

Variable eipScanner::vendor::ra::powerFlex525::MAX_FAULT_PARAMETER_NUMBER

- Defined in file_src_vendor_ra_powerFlex525_DPIFaultManager.cpp

Variable Documentation

const int `eipScanner::vendor::ra::powerFlex525::`**MAX_FAULT_PARAMETER_NUMBER** = 10

Defines

Define EIP_DEFAULT_EXPLICIT_PORT

- Defined in file_src_sockets_EndPoint.h

Define Documentation

EIP_DEFAULT_EXPLICIT_PORT

Define EIP_DEFAULT_IMPLICIT_PORT

- Defined in file_src_sockets_EndPoint.h

Define Documentation

EIP_DEFAULT_IMPLICIT_PORT

Define EIPSCANNER_SOCKET_ERROR

- Defined in file_src_sockets_Platform.h

Define Documentation

EIPSCANNER_SOCKET_ERROR(err)

Typedefs

Typedef eipScanner::cip::CipBool

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint8_t `eipScanner::cip::`**CipBool**
Boolean data type

Typedef eipScanner::cip::CipByte

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint8_t `eipScanner::cip::`**CipByte**
8-bit bit unsigned integer

Typedef eipScanner::cip::CipDint

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef int32_t `eipScanner::cip::`**CipDint**
32-bit signed integer

Typedef eipScanner::cip::CipDword

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint32_t `eipScanner::cip::`**CipDword**
32-bit bit unsigned integer

Typedef eipScanner::cip::CipInt

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef int16_t `eipScanner::cip::`**CipInt**
16-bit signed integer

Typedef eipScanner::cip::CipLint

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef int64_t `eipScanner::cip::`**CipLint**
64-bit bit signed integer

Typedef eipScanner::cip::CipLreal

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef double `eipScanner::cip::`**CipLreal**
64-bit IEEE 754 floating point

Typedef eipScanner::cip::CipLword

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint64_t `eipScanner::cip::`**CipLword**
64-bit bit unsigned integer

Typedef eipScanner::cip::CipOctet

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint8_t `eipScanner::cip::`**CipOctet**
8 bit value that indicates particular data type

Typedef eipScanner::cip::CipReal

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef float eipScanner::cip::**CipReal**
32-bit IEEE 754 floating point

Typedef eipScanner::cip::CipShortString

- Defined in file_src_cip_CipString.h

Typedef Documentation

using eipScanner::cip::**CipShortString** = CipBaseString<*CipUsint*>

Typedef eipScanner::cip::CipSint

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef int8_t eipScanner::cip::**CipSint**
8-bit signed integer

Typedef eipScanner::cip::CipString

- Defined in file_src_cip_CipString.h

Typedef Documentation

using eipScanner::cip::**CipString** = CipBaseString<*CipUint*>

Typedef eipScanner::cip::CipUdint

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint32_t eipScanner::cip::**CipUdint**
CipUdint 32-bit unsigned integer

Typedef eipScanner::cip::CipUint

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint16_t eipScanner::cip::**CipUint**
CipUint 16-bit unsigned integer

Typedef eipScanner::cip::CipUlint

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint64_t eipScanner::cip::**CipUlint**
64-bit bit unsigned integer

Typedef eipScanner::cip::CipUsint

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint8_t eipScanner::cip::**CipUsint**
8-bit unsigned integer

Typedef eipScanner::cip::CipWord

- Defined in file_src_cip_Types.h

Typedef Documentation

typedef uint16_t eipScanner::cip::**CipWord**
16-bit bit unsigned integer

Typedef eipScanner::fileObject::EndUploadHandler

- Defined in file_src_FileObject.h

Typedef Documentation

using eipScanner::fileObject::**EndUploadHandler** = std::function<void (cip::*GeneralStatusCodes* status, **const** std::vector<uint8_t> &fileContent)>

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

E