CONNECTING TO AN RFID READER

1. Introduction

RFID readers in vehicle telematics offer enhanced security, asset tracking, and operational efficiency by identifying operators and assets in real time. They support access control, ensuring only authorised personnel use specific machines and enter restricted zones. RFID also aids in tracking asset movements, optimising routes, and monitoring operator compliance, making it valuable in fleet management, logistics, and high-security areas.

This application note describes how to interface to an RFID reader to be able to identify drivers and enable machine operation. The reader chosen is the MW-R8B from Netronix. This reader was chosen because it is optimised for use when mounted on metal surfaces and is readily available. Other features include:

- Support of MIFARE Classic, Plus, Ultralight C, DESFire, ICODE SLI, HID iCLASS (CSN only) family cards and transponders, working on frequency of 13,56MHz.
- RS232(TTL), RS485, 1-Wire, WIEGAND, and CAN interfaces
- Built-in antenna.
- Buzzer, touch button and programmable LED RGB diode.

Figure 1- Netronix MW-R8G (grey) and MW-R8B (black) RFID Readers

For testing, we will use the ART10354 MIFARE Classic 13.56MHz key fob. It is highly available but is not recommended for high security applications.

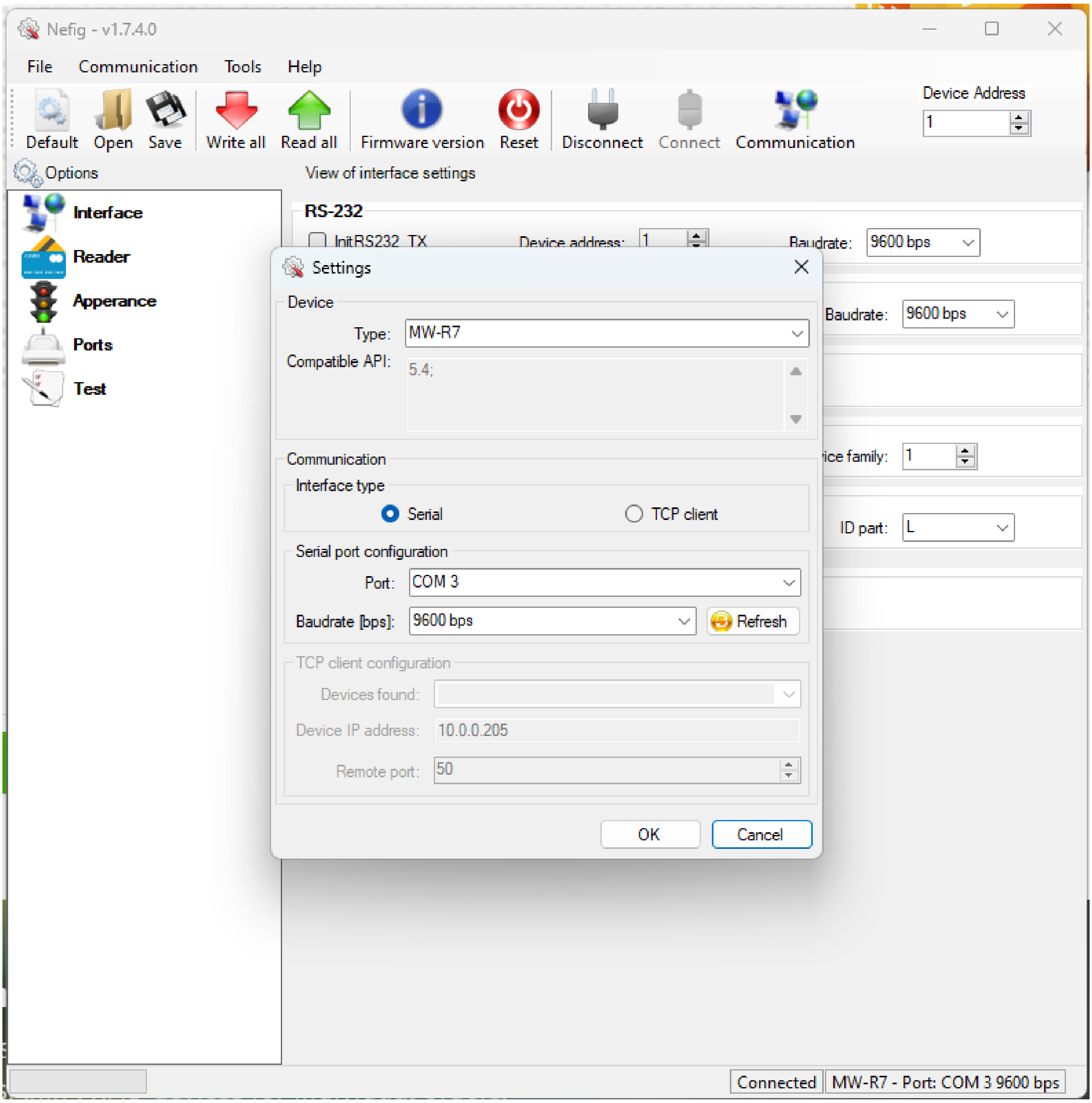
Figure 2 - ART10354 Key Fob

Extensive use of the Senquip scripting language will be used in this application note. Further details on the Senquip scripting language can be found in the Device Scripting Guide.

2. Getting to Know the Reader

Netronix provide configuration software for their readers. Although the reader that we purchased was an MW-R8B, and it is labelled as such, the software will only recognise it if we specify an MW-R7. We used a USB to RS485 converter to connect to the reader for initial configuration and testing.

We will use the MW-R7 manual from here on.



The following changes were made to the default settings on the reader by using the Netronix software:

- **Device Address:** The reader address was left as the default of 0x01.
- **Read Type**: Changed to Mifare only since we will only be using a Mifare card in this application.
- **Trigger Type**: The reader can be set to turn on based on serial data arriving. We turned it to permanently on to ease complexity. We ended up polling the device more often than the 2 second timeout and so changing the setting will have made no difference.
- **Inform Type Serial:** By sniffing the settings sent to the reader, we were able to ascertain that this setting sets whether the reader only sends the card number on first application or on every read. We set it to every read.
- **Inform Type Buzzer:** We set this to beep the buzzer on every read of a card.
- **Return on Interface:** The function of this setting is unclear but may direct the reader to return the card ID on a particular interface. We set this to RS485, the same interface over which the setup is being performed.

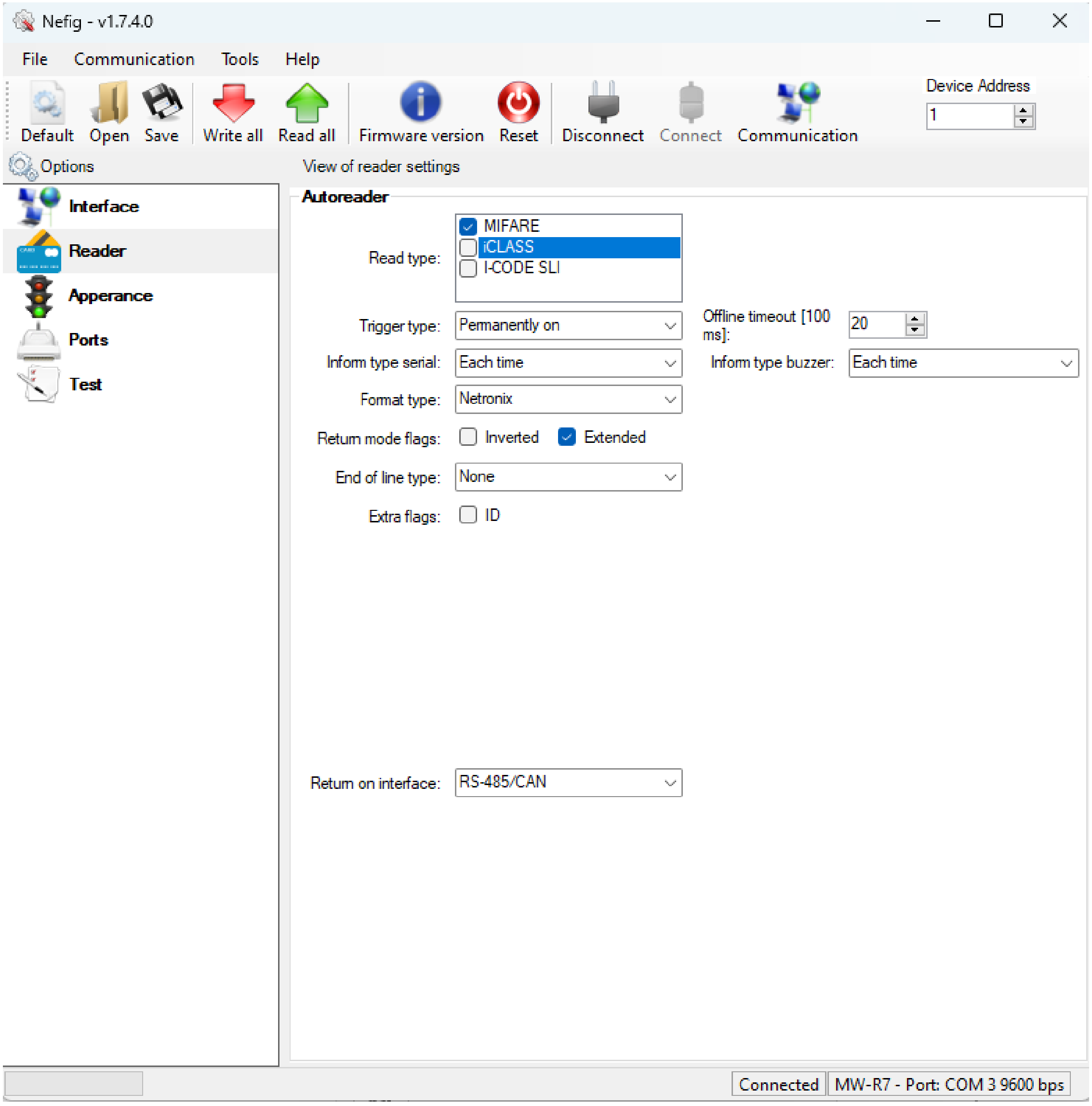


Figure 3 - Changing Reader Settings with Netronix Software

The MW-R7B supports the Netronix protocol and a Modbus protocol. The initial intention was to use the Modbus protocol however it appears as though the Modbus implementation is just another layer on top of the Netronix protocol and so we will proceed using the Netronix protocol. The Netronix Protocol is described in the device Technical Datasheet and in the Netronix Protocol Technical Datasheet.

The protocol defines the following format for a message sent to the reader:

Module Address	Frame Length	Command	Parameters	Checksum	
1 byte	1 byte	1 byte (always even)	N bytes	High byte	Low byte

Our module address, or the address of the card reader on the RS485 network is 0x01. There could be multiple card readers on the same network as long as each one has a different address.

The frame length is the length of the entire frame including the address, frame size, command, parameters and CRC16 checksum bytes.

The commands are described in the device Technical Datasheet. For instance, the command to read the firmware revision is 0xFE.

A response takes the following form:

Module Address	Frame Length	Response	Parameters	Operation Code	Checksum	
1 byte	1 byte	1 byte (command +1, always odd)	N bytes	1 byte	High byte	Low byte

The response byte is always 1 larger than the command byte, for instance the command for a firmware revision read is 0xFE and so the response is 0xFF.

The response code provides information on whether the command was correctly executed or not and is command dependent.

To better understand the Netronix protocol, we used the vendor software and an RS485 sniffer to see what commands were being sent when reading the firmware version, resetting the card reader, uploading and download settings, and to do a card read.

Read Firmware

From the Technical Datasheet:

7.7.2 READING-OUT SOFTWARE VERSION FROM READER

Command frame:

header	C_FirmwareVersion		CRC

Where:

Parameter name	Parameter description	Value range
C_FirmwareVersion	Reading-out reader software version	0xfe

Response frame:

header	C_FirmwareVersion+1	Data1.....n	OperationCode	CRC

Where:

Data1 ... n is a string of characters stored in the form of ASCII codes.

From the RS485 sniffer:

Sent to reader:

Module Address	Frame Length	Command	Parameters	Checksum	
01	05	FE	-	C6	14

Response from reader:

Module Address	Frame Length	Response	Parameters	Operation Code	Checksum	
01	10	FF	4D 57 2D 52 37 2D 56 38 2E 36	FF	1E	12

The response FF is as expected one greater than the command.

The Parameters, when converted to ASCII are “MW-R7-V8.6”. Again, frustrating as this is an MW-R8.

The operation code FF is for “Operation completed correctly”.

Reset

From the Technical Datasheet:

7.7.1 REMOTE READER RESET

Command frame:

header	C_Reset		CRC

Where:

Parameter name	Parameter description	Value range
C_Reset	Remote reader reset	0xd0

Response frame:

header	C_Reset +1		OperationCode	CRC

From the RS485 sniffer:

Sent to reader:

Module Address	Frame Length	Command	Parameters	Checksum	
01	05	D0	-	03	B8

Response from reader:

Module Address	Frame Length	Response	Parameters	Operation Code	Checksum	
01	06	D1	-	FF	FC	F2

The response D1 is as expected one greater than the command.

There are no parameters.

The operation code FF is for “Operation completed correctly”.

Reading a Card

When reading a card, we noticed that the reader would cycle through a command to turn on the antenna reader field and then a read-card-id command.

From the Technical Datasheet:

7.2.2.1 ENABLING AND DISABLING READER FIELD

Command frame:

C_TurnOnAntennaPower	State

Where:

Parameter name	Parameter description	Value range
C_TurnOnAntennaPower	Enabling and disabling reader field	0x10
State	State	0x00 – disabling field 0x01 – enabling field

Response frame:

C_TurnOnAntennaPower +1		OperationCode

7.2.2.2 SELECTION OF ONE TRANSPONDER FROM MANY

Command frame:

C_Select	

Where:

Parameter name	Parameter description	Value range
C_Select	Reading-out ID	0x12

Response frame:

C_Select +1	Coll, TType, ID1…….IDn	OperationCode

From the RS485 sniffer (turn on antenna power):

Sent to reader:

Module Address	Frame Length	Command	Parameters	Checksum	
01	06	10	01	D7	46

Response from reader:

Module Address	Frame Length	Response	Parameters	Operation Code	Checksum	
01	06	11	-	FF	EA	A6

From the RS485 sniffer (read card id):

Sent to reader:

Module Address	Frame Length	Command	Parameters	Checksum	
01	05	12	-	FA	B6

Response from reader (with no card present):

Module Address	Frame Length	Response	Parameters	Operation Code	Checksum	
01	08	13	00 00	1E	34	09

Here the ID is seen to be 00 00 and the operation code 1E which indicates “CRC error/transmission with card”. We also saw operation code 16 which indicates “Operation time exceeded.” This indicates that there was no card found in the field withing the allocated time.

Response from reader (with a card present):

Module Address	Frame Length	Response	Parameters	Operation Code	Checksum	
01	0C	13	00 50 F0 EF A0 D4	FF	4A	DD

According to the reference for the response, the Coil is 00 and the Ttype is 50. No information could be found on what this means. The card number is F0 EF A0 D4, which is correct as confirmed with the supplier software by doing a card test.

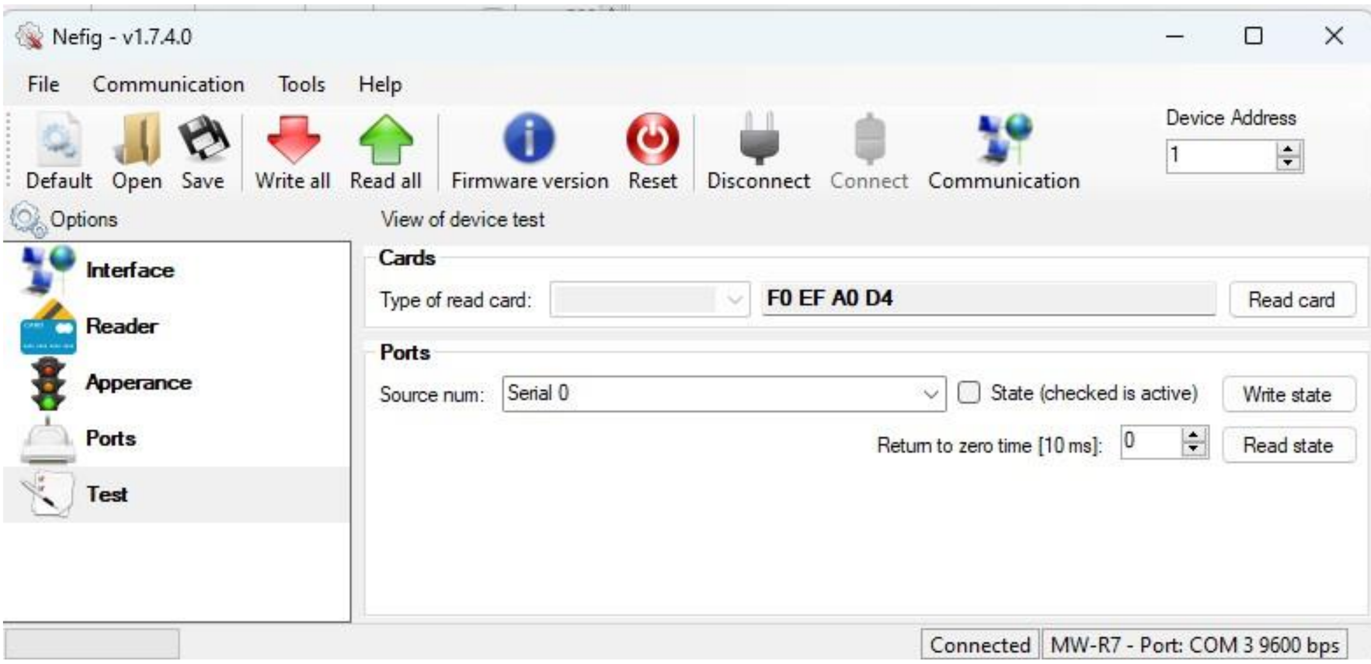
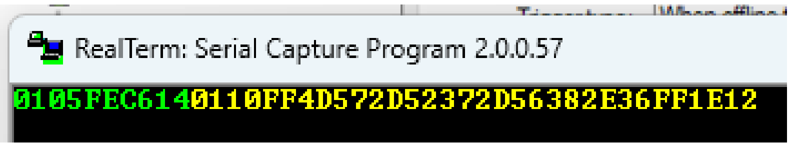


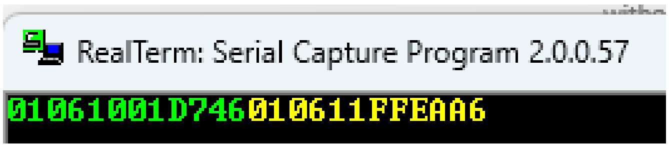
Figure 4 - Card Test using Netronix Software

Some of these commands were then sent (green) using Realterm and the retuned data (Yellow) was as expected:

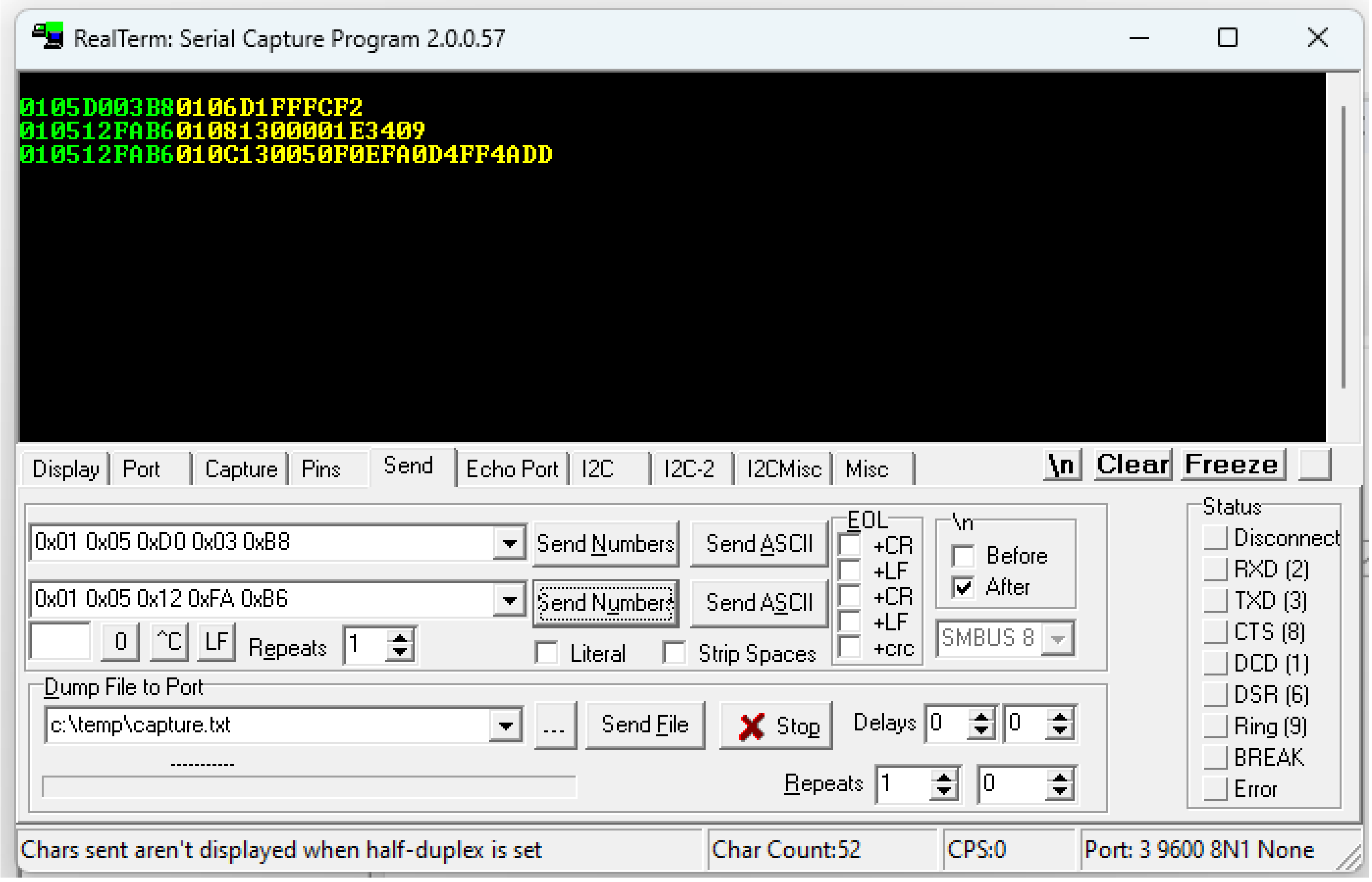
Read Firmware:



Enable Antenna Field:



And then a reset followed by a read with no card and a read with a card present:



After this testing, we were confident that we understood how to do a card read and could automate the process using a script on a Senquip device.

3. Connection to a Senquip Device

A Senquip QUAD-C2 was used in this application.

We used RS485 as the preferred connection method. The RS232 available is TTL level and is not suitable for use with the Senquip serial port. CAN could also have been used. RS485 is the only available interface for some of the lower cost Netronix RFID readers and is the default for the MW-R8B.

The supply voltage is 8V to 24V and the nominal supply current is 40mA although it is noted that the peak current can be 120mA. We will connect the card reader to the same 12V supply as the Senquip device.

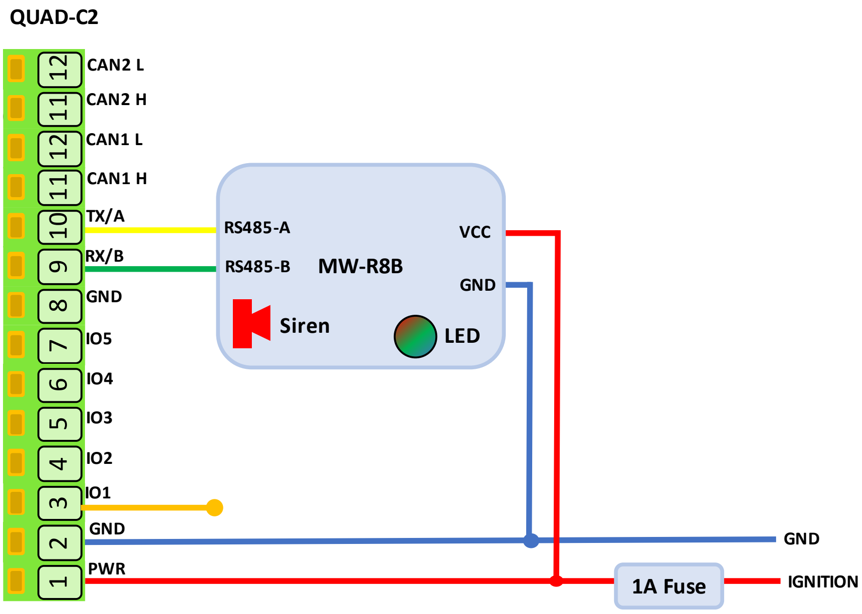


Figure 5 - Card Reader Connection to Senquip QUAD

The Senquip QUAD was configured with a *Base Interval* of 5 seconds and the serial port was set to RS485, with a baud rate of 9600, 8 bits, no parity, and 1 stop bit. The serial port *Mode* was set as scripted as the serial port will be completely controlled within a script.

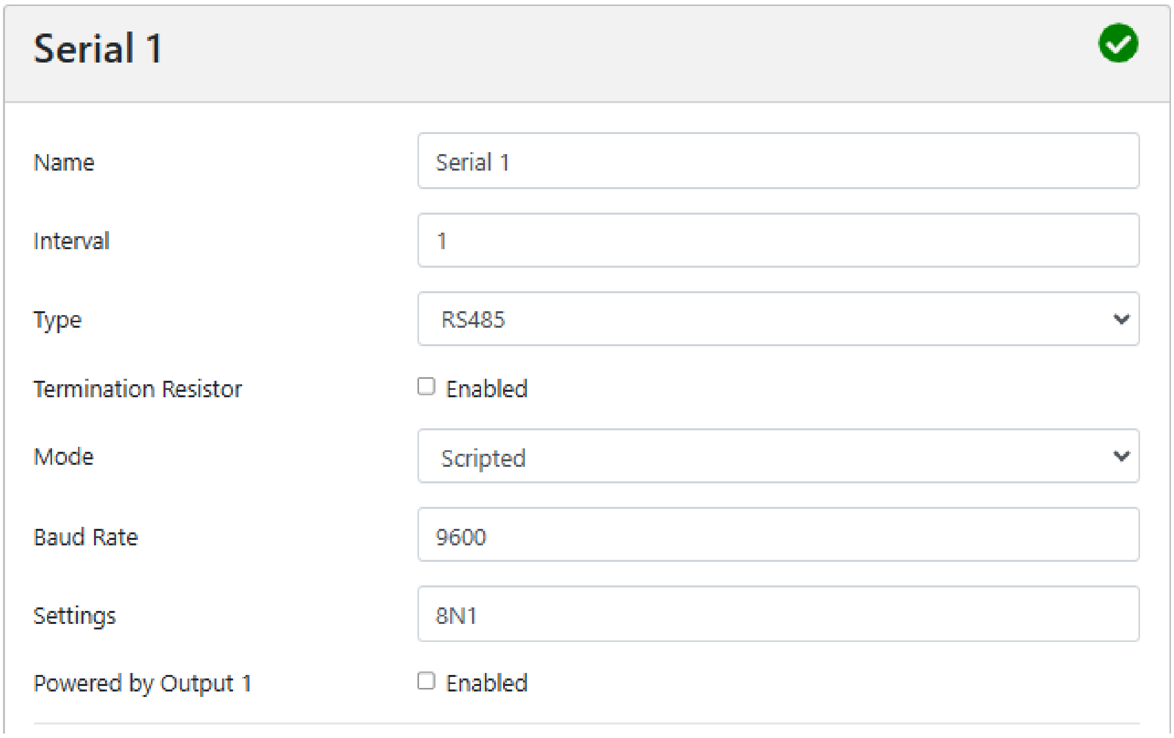


Figure 6 - Senquip ORB Serial Port Settings

4. The Scripted Application

The scripted application will check for cards, read the card number, display it on the Senquip Portal, and will check the card number against a list of cards with known users. If a card is matched, the output will be turned on for 30 seconds and then turned off.

To check the card status, the antenna field is first turned on, and then a card read is executed. A timer is configured to repeat this every second. In the timer function, a message to enable the reader field is sent. The card reader is expected to respond with a confirmation message.

```
load('senquip.js');
load('api_config.js');
load('api_serial.js');
load('api_timer.js');

let serial_data = "";
let card_no = -1; // rather than 0 because custom cumbers default to 0
let access = 0; // default access denied
let timer_id = 0;   // Zero is the only safe value for an invalid timer id

Timer.set(1000, Timer.REPEAT, function() { // This will repeat every second
  let s = "\x01\x06\x10\x01\xD7\x46";  // turn on field
  SERIAL.write(1, s, s.length);
}, null);
```

A serial handler is called each time data arrives on the serial port. The handler checks that the correct number of bytes as required for the message type has been received and if so, what type of message has been received. If a response to the turn on field command has been received, then a read command is sent to the reader. If a response to the card read has been received and there is a card, then the card number is checked for access.

```
SERIAL.set_handler(1, function(channel) {
  serial_data = serial_data + SERIAL.read(channel);
  if(serial_data.at(0) !== 1){ // address byte is in first position
    serial_data = "";
  }
  else if (serial_data.length >= serial_data.at(1)){
    if (serial_data.at(2) === 0x11){ // turn on field response
      let s = "\x01\x05\x12\xFA\xB6";  // read card command
      SERIAL.write(1, s, s.length,SERIAL.IMMEDIATE);
    }
    else if (serial_data.at(2) === 0x13){
      if (serial_data.at(1) > 8){
      card_no = (serial_data.at(5)<<24) + (serial_data.at(6)<<16) + (serial_data.at(7)<<8) +
(serial_data.at(8));
      check_access(card_no);
      }
    }
      serial_data = "";
  }
}, null);
```

Custom number settings are used to store usernames against card numbers. The 10 custom numbers are checked against the card number and if a match is received, output 1 is switched to Vin for 10 seconds.

```
function check_access(card){
  for (let i = 1; i < 10; i++){
    if (card === Cfg.get('script.num' + JSON.stringify(i))){
      access = 1;
      IO.write(1, IO.VIN); // card matches so turn the IO on
      if(timer_id === 0){ // if there is no timer running
        timer_id = Timer.set(10000, 0, function() {// After 10 seconds, turn the IO off
          timer_id = 0;       // IMPORTANT!: Clear the stored id. This is not done automatically.
          IO.write(1, IO.GND);
          access = 0;
          card_no = -1;
        }, null);
      }
      break;
    }
  }
}
```

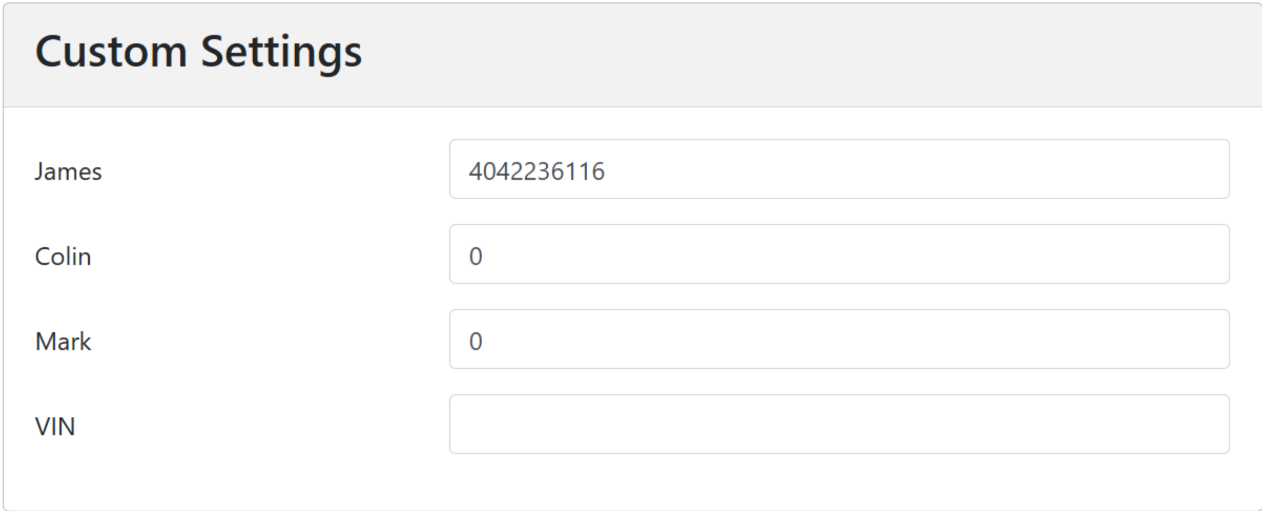


Figure 7 - Custom Number Settings Used to Store Users and Card Details

```
SERIAL.set_handler(1, function(channel) {
  serial_data = serial_data + SERIAL.read(channel);
  if(serial_data.at(0) !== 1){ // address byte is in first position
    serial_data = "";
  }
  else if (serial_data.length >= serial_data.at(1)){
    if (serial_data.at(2) === 0x11){ // turn on field response
      let s = "\x01\x05\x12\xFA\xB6";  // read card command
      SERIAL.write(1, s, s.length,SERIAL.IMMEDIATE);
    }
    else if (serial_data.at(2) === 0x13){
      if (serial_data.at(1) > 8){
      card_no = (serial_data.at(5)<<24) + (serial_data.at(6)<<16) + (serial_data.at(7)<<8) +
(serial_data.at(8));
      check_access(card_no);
      }
    }
      serial_data = "";
  }
}, null);
```

In the main data handler, the current card number and access status are dispatched to the Senquip Portal.

```
SQ.set_data_handler(function(data)
{
  SQ.dispatch(4,card_no);
  if (access === 1){SQ.dispatch(2,"Granted");} else {SQ.dispatch(2,"Denied");}
}, null);
```

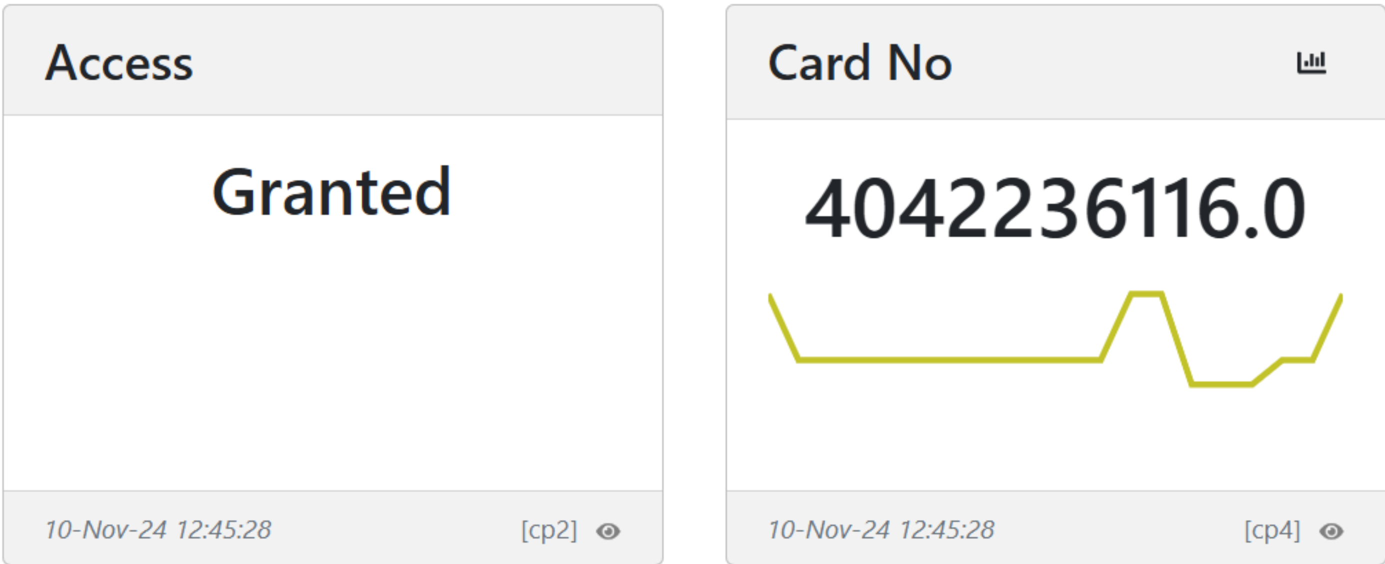


Figure 8 - Access Status and Card Number in the Senquip Portal

4.1. Further work

The lights and buzzer on the card reader can be controlled through the serial port. This would be useful as the buzzer could be sounded when the Senquip device gets a card read, rather than the card reader. The LED lights would be useful as general status indicators.

5. Conclusions

Although the documentation for the card reader was not very clear; by sniffing the serial lines, it has been fairly simple to work out how to read cards using the Netronix MW-R8B card reader by using a script to control the serial port.

The use of custom numbers allows a list of users to be loaded using the Senquip Portal. Outputs can be controlled based on the card status.

6. Appendix I – Example Script to Read Card Details

```
load('senquip.js');
load('api_config.js');
load('api_serial.js');
load('api_timer.js');

let serial_data = "";
let card_no = -1; // rather than 0 because custom cumbers default to 0
let access = 0; // default access denied
let timer_id = 0;   // Zero is the only safe value for an invalid timer id

Timer.set(1000, Timer.REPEAT, function() { // This will repeat every second
  let s = "\x01\x06\x10\x01\xD7\x46";  // turn on field
  SERIAL.write(1, s, s.length);
}, null);


function check_access(card){
  for (let i = 1; i < 10; i++){
    if (card === Cfg.get('script.num' + JSON.stringify(i))){
      access = 1;
      IO.write(1, IO.VIN); // card matches so turn the IO on
      if(timer_id === 0){ // if there is no timer running
        timer_id = Timer.set(10000, 0, function() {// After 10 seconds, turn the IO off
          timer_id = 0;       // IMPORTANT!: Clear the stored id. This is not done automatically.
          IO.write(1, IO.GND);
          access = 0;
          card_no = -1;
        }, null);
      }
      break;
    }
  }
}

SERIAL.set_handler(1, function(channel) {
  serial_data = serial_data + SERIAL.read(channel);
  if(serial_data.at(0) !== 1){ // address byte is in first position
    serial_data = "";
  }
  else if (serial_data.length >= serial_data.at(1)){
    if (serial_data.at(2) === 0x11){ // turn on field response
      let s = "\x01\x05\x12\xFA\xB6";  // read card command
      SERIAL.write(1, s, s.length,SERIAL.IMMEDIATE);
    }
    else if (serial_data.at(2) === 0x13){
      if (serial_data.at(1) > 8){
      card_no = (serial_data.at(5)<<24) + (serial_data.at(6)<<16) + (serial_data.at(7)<<8) +
(serial_data.at(8));
      check_access(card_no);
      }
    }
      serial_data = "";
  }
}, null);


SQ.set_data_handler(function(data)
{
  SQ.dispatch(4,card_no);
  if (access === 1){SQ.dispatch(2,"Granted");} else {SQ.dispatch(2,"Denied");}
}, null);
```

7. Appendix II – Introduction to RFID Standards

RFID devices come in various types, each tailored to different use cases and frequency ranges. High Frequency 13.56MHz systems are the most popular and will be the focus of the rest of this description.

Frequency Range	Typical Frequency	Read Range	Characteristics	Popular Standards	Typical Applications
Low Frequency (LF)	125 kHz & 134.2 kHz	Up to 10 cm (sometimes up to 1 m)	Performs well around metals and liquids; slower data speeds	ISO 11784/11785	Animal tracking, vehicle immobilizers, access control
High Frequency (HF)	13.56 MHz	Up to 1 m	Moderate data speed; low sensitivity to interference; globally adopted	ISO 14443 (NFC), ISO 15693	Contactless payment (NFC), library books, secure access control
Ultra High Frequency (UHF)	865-868 MHz (EU), 902-928 MHz (USA)	Up to 12 m or more	High data speeds; long range; sensitive to metal and liquid interference	ISO 18000-6C (EPC Gen2)	Supply chain, item tracking, warehouse management, vehicle ID
Microwave RFID	2.45 GHz & 5.8 GHz	Up to a few meters	Very high data rates; susceptible to interference; line-of-sight needed	ISO 18000-4	Electronic toll collection, active RFID for real-time location tracking

The following high frequency 13.56MHz protocols are the most popular:

Standard	Overview	Security	Use Cases	Characteristics
MIFARE Classic	Early, widely used RFID card technology by NXP	Proprietary Crypto-1 (compromised)	Access control, public transport, low-security applications	Limited security, often being phased out
MIFARE Plus	Upgraded version of MIFARE Classic with better security	AES encryption	Secure access control, public transport, cashless payment	Backward compatible with MIFARE Classic
MIFARE Ultralight C	Cost-effective, lightweight solution for single-use	3DES encryption	Disposable tickets, public transport, loyalty cards	Low memory, designed for limited-lifetime use
MIFARE DESFire	Advanced with high security and multi-application support	AES encryption, multi-application support	High-security access control, government ID, public transport	Supports multiple applications on one card
ICODE SLI	ISO 15693-compliant, optimized for item tagging	Basic, no encryption	Library books, retail inventory, item-level applications	High read range, supports mass reading of tags
HID iCLASS (CSN only)	Secure HF standard by HID for access control	Limited to card serial number only (CSN)	Access control requiring only unique ID (no data encryption)	Widely used in corporate/government access control

The tag used in this application note is a high frequency 13.56MHz MIFARE Classic, is highly available but is not recommended for high security applications.