# Interface Control Document

## CR8200 Engine-based products

**TABLE OF CONTENTS**

# 1 Scope

This interface control document (ICD) specifies the communication protocol between Code Reader™ 8200 hardware and application software that runs on the host computer, specific Reader commands, examples of a variety of ways to communicate and send data to the Reader (i.e. RS232) and command/communication types.

# 2 Communication Medium

The reader communicates with the Host via USB (keyboard/HID) or RS-232. The Host includes appropriate hooks and/or drivers to enable two-way communication with the reader.

# 3 Reader to Host Communication

The reader is configured for raw mode, where no packet framing or check characters are sent, and packet mode. In raw mode, the reader is configured to not expect response from the host and no automatic retry. In packet mode, the reader is configured to expect an acknowledgment from the host after each packet and automatic retry when no acknowledgment is received. If no acknowledgment (ACK) is received, five attempts to resend are made.

## 3.1 Packet Data

Data from the Reader to the Host consists of *packets* as specified below. The communication protocol for CR8200 consists of three layers. The protocol makes use of some features of the TCP/IP protocol.

- Packet Layer
  - Parsing for packets and raw data
  - Handle packet protocol.
- Protocol Layer
  - Handles the payload from a packet or out-of-band raw data
- Payload protocol layer
  - Abstraction layer defined by the protocol layer

### 3.1.1 Endianness

Unless specified otherwise, all fields are big-endian.

### 3.1.2 Packet Layer

The principle service provided by this layer is transferring data from protocol layer on the source to the protocol layer on the destination. The following illustrates the format of the packet layer.

Frame Format -

| Start of Frame (3-bytes) | Packet Version (1-byte) | Packet Length (2-bytes) | Destination Address (4-bytes) | Source Address (4-bytes) | Protocol Type (1-byte) | Payload (0-n bytes) | CRC16 (2-bytes) |
|---|---|---|---|---|---|---|---|

#### 3.1.2.1 Start of Frame

The sequence **"<SOH>CT"** indicates the start of the frame.

#### 3.1.2.2 Packet Version

The packet version field indicates the version of the received packet. Currently, this field can only have a value of '1'.

#### 3.1.2.3 Packet Length

This field indicates the length of the packet. The length of a packet is the number of bytes including destination address field through the CRC16 field.

#### 3.1.2.4 Source and Destination Address

The destination address field identifies the device or devices that will receive the packet. The source address identifies the device that originated the packet. The destination address can be either an "individual address" destined for a single device or a "broadcast address" destined for all connected devices. The source and destination address structure is shown below.

Address Structure:

| 31:30 | 29:28 | 27:0 |
|---|---|---|
| Device type:<br>b00: Reader<br>b01: Host<br>b10: Modem<br>b11: Reserved | Reserved for future use, set to b00 | Device address:<br>0x0: No known address<br>0x001 – 0x1FFF: Reserved<br>0xFFFFFFF: Broadcast address<br>All others: Preprogrammed or negotiated address |

- *Device Type* – Bits [31:30] in the destination address indicates the type of the device or devices that will receive the packet. Same bits in the source address identifies the device that originated the packet.
- *Reserved* – Bits [29:28] are reserved should be set to '0'.
- *Device Address* – Bits [27:0] identifies the actual device address. In case of a reader, the address is the chip ID. A value of 0 would indicate that the source or destination device does not have an address, e.g. CortexTools.

#### 3.1.2.5 Protocol Type

This field identifies the type of the protocol.

| Protocol Types | |
|---|---|
| Value | Description |
| 0x00 | Packet Protocol |

| | |
|---|---|
| 0x01 | Connection Protocol |

#### 3.1.2.6 Payload

This field contains the data transferred from source device to the destination device or devices. The maximum size of this field is 65536 bytes.

#### 3.1.2.7 CRC16

This field contains 2-bytes cyclic redundancy check (CRC) value used for error checking. When the source device assembles a packet, it performs a CRC calculation on all the bits in the packet from the destination address through the payload fields. The source device stores the value in this field and transmit it as a part of the packet. When the packet is received by the destination device, it performs an identical check. If the calculated value doesn't match the value in the field, the destination device assumes an error has occurred and discards the packet. See source files crc16.[hc] in Appendix: Example CRC16 C Code for details on the crc16 algorithm and polynomials to be used.

### 3.1.3 Protocol Layer

This layer is a connection oriented layer, it is made possible by exchanging acknowledgment for the packets received. This layer also provides reliable service by implementing an error control mechanism. If the sender doesn't receive the ACK for the sent packet before it times out, the sender retransmits the packet. Following protocol types have been defined at this layer.

#### 3.1.3.1 Packet Protocol

The packet layer has a specific protocol designed to notify the host/reader of the basic packet capabilities. Allows for negotiating certain parameters relevant to sending/receiving packets.

Packet Payload Format:


##### 3.1.3.1.1 Message Type

This field identifies the type of the message. The message could be either an error report or a query message.

- 0x00 – Indicates the current packet is a request. Message subtype field would describe the request in detail.
- 0x01 – Indicates the current packet is a reply. Message subtype field would describe the reply in detail.
- 0x02 through 0x7F – Reserved.
- 0x80 – Indicates an error message. Message subtype field describe the error in detail.

##### 3.1.3.1.2 Message Subtype

This field describes the payload type in more detail.

| Payload Type | Code | Description | Payload size (bytes) | Notes |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 0x00 – Request | 0 | MTU Size | 0 | |
| | 1 | Max Window Size | 0 | Reserved – not defined |
| | 2 | Max Buffer Size | 0 | |
| | 3 | Connected devices | 0 | |
| 0x01 – Reply | 0 | MTU Size | 4 | |
| | 1 | Max Window Size | 4 | Reserved – not defined |
| | 2 | Max Buffer Size | 4 | |
| | 3 | Connected devices | | Reserved – not defined |
| 0x02 through 0x7F | | Reserved | | |
| 0x80 – Error | 0 | MTU exceeded, payload contains MTU size. | 4 | |
| | 1 | Buffer size exceeded, payload contains Max buffer size | 4 | |
| | 2 | Unreachable, no payload | | Reserved – not defined |

#### 3.1.3.1.3 Payload

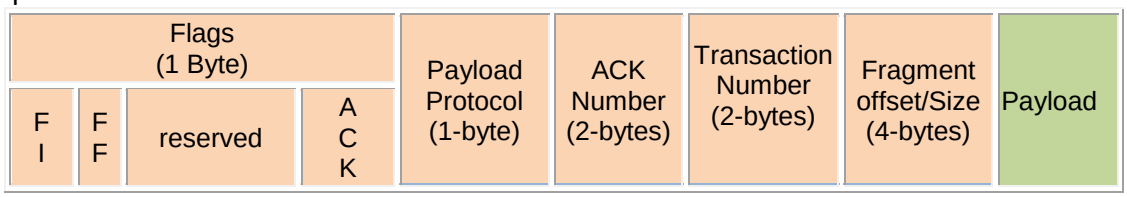
Data in this field depends upon the message type and message subtype field. Payload field is not present when message type is a request. When message type is a reply, it will contain the same code that was in the request and payload will contain a value. If message type is an error, payload may or may not be present depending upon message subtype.

#### 3.1.3.1.4 Terms

- **MTU Size** – This is the maximum packet size that the device can receive. Or, in other words, it is the largest number that can be placed in the "packet length" field of the packet layer that the device can accept.
- **Max Window Size** – This is reserved, and is not defined yet.
- **Max Buffer Size** – This is the largest command that the device can receive. Even though a command can be fragmented between multiple packets, the combined length of the command layer data contained within those fragments must not exceed this size.

### 3.1.3.2 Connection Protocol

The connection protocol establishes and maintains a connection. Performs the fragmentation and reassembly of the larger packets.




#### 3.1.3.2.1 Flags

- Bit[0] – Indicates ACK number in the "ACK Number" field is valid
- Bit [5:1] – Reserved and should be set to 4'b0.
  Bit [6] – First Fragment - Indicates this is the first fragment of the requested data. When this bit is set to 1, bit [7] must be set to 1.
- Bit [7] – Fragment Included - indicates that the current payload includes a fragment of the requested data.

#### 3.1.3.2.2 Payload Protocol

This field identifies the payload protocol (See 3.1.4 for protocol details)

- 0x00 – Connection Protocol ACK.
  If the Fragment Included flag is set to 0, this payload protocol does **NOT** contain the following fields:
  - Transaction Number
  - Fragment Offset/Size
  - Payload

  If the Fragment Included flag is set to 1, the Transaction Number and Fragment Offset/Size fields will be included, but the Payload field will not be included.
- 0x01 – Decode data protocol.
- 0x02 – Command protocol.
- 0x03 – Image transfer protocol.

#### 3.1.3.2.3 ACK Number

If ACK bit in the flags field is set, this field contains the transaction number of the last packet received.

#### 3.1.3.2.4 Transaction Number

This field indicates the transaction number of the current packet. The sender assigns each packet it sends a unique sequence number. Zero is not a valid transaction number. A transaction number of zero can only be used in conjunction with a Connection Protocol ACK packet (when the Fragment Included flag is 1). When sending fragmented packets, the transaction number is the same for all packets associated with that transaction.

#### 3.1.3.2.5 Fragment Offset/Size

This field is present only if fragments included bit in the flags field is set. When the First Fragment flag is set, this field contains the total data size of the requested data and offset of the payload is 0, otherwise this field indicates the relative position of the current fragment with respect to the whole data.

#### 3.1.3.2.6 Payload

This field contains data for the payload protocol.

#### 3.1.3.2.7 Fragmented Packet acknowledgment

When acknowledging fragmented packets, the acknowledgement packet must have a protocol type of “Connection Protocol ACK”. The Fragment Included flag will also be set to 1, and if this packet is acknowledging the first packet, the First Fragment bit will also be set to 1. The Fragment Offset/Size field will be the same as the received packet that is being acknowledged.

#### 3.1.3.2.8 Cancel Fragmented Transaction

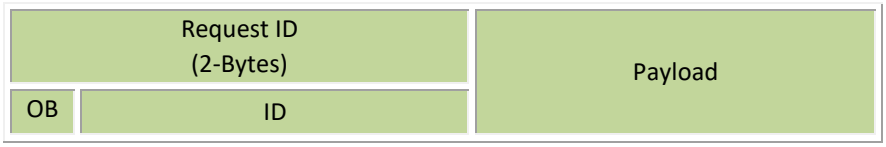
To cancel a fragmented transaction, set the First Fragment and Fragmented Included flag to 1 and set the Fragment Offset/Size field to 0. The Transaction Number field indicates the transaction being cancelled.

## 3.1.4 Payload Protocol Layer

This section describes the sub-protocols for the Connection Protocol. Each sub-protocol contains a unique id used for associating request/response packets.

### 3.1.4.1 Command/Data Protocol

Format:




#### 3.1.4.1.1 Request ID

This is the ID that will associate request/response packets. The sender assigns a unique ID to each request it sends.

#### 3.1.4.1.2 Originator Bit (OB)

When initiating a request this bit will be set to 1. Any response to a request with a matching request ID will set this bit to 0.

#### 3.1.4.1.3 Decode Data Protocol

The payload for this type of request contains decode data. This is typically originated from the Reader, however a host may want the reader to process data as though it were decoded.

#### 3.1.4.1.4 Command Protocol

The payload for this type of request contains commands. This data will typically contain configuration type commands to be processed by the Reader.

### 3.1.4.2 Image Transfer Protocol

Format:


#### 3.1.4.2.1 XML formatted Header

The XML formatted header describes the image. The header includes the image type and other information necessary to describe the image. At this time only “RAW” and “PGM” image types are supported.

Example Header:

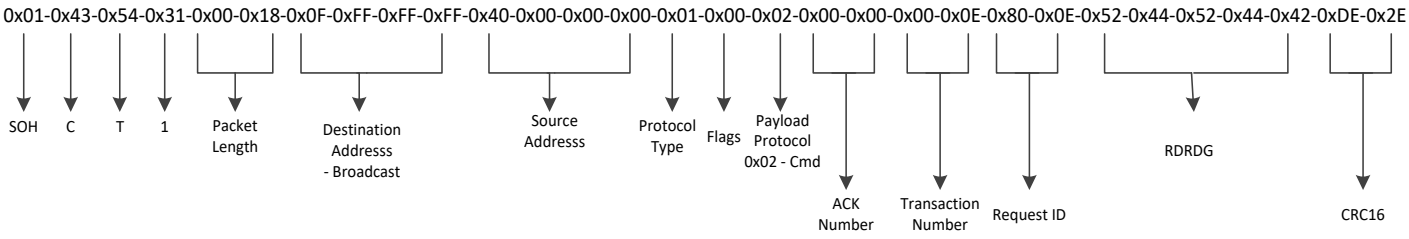
<IMG Type = “RAW”, Width= “1280”, Height = “960” />

<IMG Type = “PGM />

Payload:

Payload contains the image data.

A host can query the reader information by sending a packetized "RDRDG" command to the reader. Bytes in a packetized command to query the reader information are shown below – each byte is separated by a hyphen.




## 3.2 Raw Data

Reader to Host communication consists of decoded raw data having no framing or check characters. Responses to commands, or asynchronous non-decode data is wrapped in a "non-decode" identifier, shown below:

Header: <SOH>X<RS>
Response <command response>
Footer: <EOT>

When sending raw non-decode responses, if a command ID was received, the command ID will be prepended to the response, contained within [] characters, similar to how the command ID is specified in the raw host-to-reader commands.

# 4 Host to Reader Communication

Commands and data from the Host to the Reader are sent in the form of *commands* as specified in this section. Code Configuration Document (CCD) describes all the supported configuration commands. Two command formats are supported: **raw-command** and **packetized-command**.
After the Host sends a complete command, it should wait for a response packet from the reader. The reader will respond with an XML formatted response:

<Response Val = "0" Description = "none" />

A value of 0 indicates that the command was processed successfully. If there is an error in processing, the value will be less than 0.

## 4.1 Raw Command

Raw commands can be sent to the reader in RS-232 mode using any serial communication software (e.g. SecureCRT, TeraTerm). The format of the raw command is described below:

**[cmdID]<command><0x0D>**

| Element | Description |
|---|---|
| [cmdID] | Optional, but must be contained within square bracket [] characters. Contains a marker that will be returned with any responses to the command |

| | |
|---|---|
| <command> | A single array of characters (non-null terminated) that is the command.<br>(See the Code Configuration Control Document (CCD) for supported configuration commands.) |
| <0x0D> | Represents an actual carriage return that terminates the raw data |

**Example** – Command to enable Aztec (AZTC) symbology on the reader, with a command ID (terminated with a carriage return)

**[1234]SYAZTCSEN**

**Example** – Command to enable Aztec (AZTC) symbology on the reader, without a command ID (terminated with a carriage return)

**SYAZTCSEN**

## 4.2 Packetized Command

Packetized commands consist of packetized data sent from Host to Reader to configure and cause the Reader to perform certain functionalities. In addition, they include error detection, making them more robust than raw commands. The protocol to packetize commands is described in section 3.1.

## 4.3 Firmware download

To download the firmware on the Reader, the Host needs to send two commands.

1. The first is to start/initialize a firmware download (RDFSX).
2. The second is to send the firmware binary data to the reader (RDFD).

The commands to update the firmware on the reader are described below

| Function | Parameter | Comments | Command | Property | Sample Value |
|---|---|---|---|---|---|
| Firmware download Start | Format | Default is -1<br>(Invalid format) | RDFSX | FM | 0 |
| | Size | Size in bytes of data to write | RDFSX | SZ | 1000000 |
| | Base Address | Starting address of the data to write | RDFSX | BA | 0 |
| | CRC | CRC of the data<br>(-1 is default and means no CRC) | RDFSX | CR | -1 |
| | Reboot | Default is 0 | RDFSX | RB | 1 |
| Firmware download data | | | RDFD | Binary data | |

**Example:**

- To start the firmware download the Host sends:

  **RDFSXFM0,SZ800000,CR1234,RB1** (Start firmware, expecting 800000 bytes file, and reboot at the end)

When the reader receives the RDFSP command, it will assign any defined parameters, and then enter the “upgrade” scanner mode. It will also provide a large buffer (>= to 5 + the firmware size) to the communication protocol layer, so that the large RDFDP command (which will likely be fragmented) can be reassembled. Once the reader has processed the RDFSP command, it would respond with a “success” or “error” response.

- Once the reader has successfully processed firmware start command, the host will send

  **RDFDfjdksal;fjskd;ajfkd;sjafkjdsl;ajfkas;hgjds;ahfkjdsa;fjskdajfs** (800000 bytes of firmware)

  When the reader receives RDFD command, it will perform any validation necessary, then write the firmware to flash. After writing firmware, the reader will reboot if either of the following conditions are met:

  - The reboot option was specified as 1 in the RDFS command
  - The first 0x4000 bytes of the flash have changed

When this process is complete, the reader will exit the “upgrade” scanner mode, and restore the previous communication protocol buffer size.

# 5 Appendix: Example CRC16 C Code

The CRC16 required by Reader-to-Host and Host-to-Reader packets (see Section 3.1.2.7) can be calculated using the following sample C code. This CRC16 consists of two consecutive bytes, each in range [0,255] most significant byte first. A CRC16 is calculated on each packet from destination address through payload fields.

```
crc_t crc = 0;
int pktLen_crc = PacketLen - 2;
crc = crc(crc, DestAddrFirstByte, pktLen_crc);
<send SOH>
<send PacketVersion>
<send PacketLength>
<…>
<send crcHighByte>
<send crcLowByte>
```

```
/* crc16.h */

#ifndef     crc16_h
    #define crc16_h

    #include <stdint.h>
    #include <stddef.h>

    #ifdef __cplusplus
extern "C" {
    #endif

    typedef uint16_t crc_t;

    crc_t crc
( crc_tinitialCrc
, const unsigned char* bufPtr
, size_t       length
);

    #ifdef __cplusplus
} // extern "C"
    #endif

#endif
```

```
/* crc16.c */

#include <crc16.h>
```

```
crc_t crc
    ( crc_tinitialCrc
    , const unsigned char* p
    , size_t       n
    )
{
      enum
      {
  crcBits  = 16,
  charBits = 8,
  diffBits = crcBits - charBits
      };

      crc_t c = initialCrc;

      #include "crc16tab.h"

      while( n-- )
    c = (c << charBits) ^ crcTab[( c >> diffBits ) ^ *p++];

      return c;
}

/*eof*/
```

---

```
/* crc16tab.h
 * crc16 table of partial remainders generated by
 * mkcrctab.c with polynomial 1021.
 * included only from within crc() function in file crc16.c
 */

static const crc_t crcTab[] =
{
    0x0000,  0x1021,  0x2042,  0x3063,  0x4084,  0x50a5,  0x60c6,  0x70e7,
    0x8108,  0x9129,  0xa14a,  0xb16b,  0xc18c,  0xd1ad,  0xe1ce,  0xf1ef,
    0x1231,  0x0210,  0x3273,  0x2252,  0x52b5,  0x4294,  0x72f7,  0x62d6,
    0x9339,  0x8318,  0xb37b,  0xa35a,  0xd3bd,  0xc39c,  0xf3ff,  0xe3de,
    0x2462,  0x3443,  0x0420,  0x1401,  0x64e6,  0x74c7,  0x44a4,  0x5485,
    0xa56a,  0xb54b,  0x8528,  0x9509,  0xe5ee,  0xf5cf,  0xc5ac,  0xd58d,
    0x3653,  0x2672,  0x1611,  0x0630,  0x76d7,  0x66f6,  0x5695,  0x46b4,
    0xb75b,  0xa77a,  0x9719,  0x8738,  0xf7df,  0xe7fe,  0xd79d,  0xc7bc,
    0x48c4,  0x58e5,  0x6886,  0x78a7,  0x0840,  0x1861,  0x2802,  0x3823,
    0xc9cc,  0xd9ed,  0xe98e,  0xf9af,  0x8948,  0x9969,  0xa90a,  0xb92b,
    0x5af5,  0x4ad4,  0x7ab7,  0x6a96,  0x1a71,  0x0a50,  0x3a33,  0x2a12,
    0xdbfd,  0xcbdc,  0xfbbf,  0xeb9e,  0x9b79,  0x8b58,  0xbb3b,  0xab1a,
    0x6ca6,  0x7c87,  0x4ce4,  0x5cc5,  0x2c22,  0x3c03,  0x0c60,  0x1c41,
    0xedae,  0xfd8f,  0xcdec,  0xddcd,  0xad2a,  0xbd0b,  0x8d68,  0x9d49,
    0x7e97,  0x6eb6,  0x5ed5,  0x4ef4,  0x3e13,  0x2e32,  0x1e51,  0x0e70,
    0xff9f,  0xefbe,  0xdfdd,  0xcffc,  0xbf1b,  0xaf3a,  0x9f59,  0x8f78,
    0x9188,  0x81a9,  0xb1ca,  0xa1eb,  0xd10c,  0xc12d,  0xf14e,  0xe16f,
    0x1080,  0x00a1,  0x30c2,  0x20e3,  0x5004,  0x4025,  0x7046,  0x6067,
    0x83b9,  0x9398,  0xa3fb,  0xb3da,  0xc33d,  0xd31c,  0xe37f,  0xf35e,
    0x02b1,  0x1290,  0x22f3,  0x32d2,  0x4235,  0x5214,  0x6277,  0x7256,
    0xb5ea,  0xa5cb,  0x95a8,  0x8589,  0xf56e,  0xe54f,  0xd52c,  0xc50d,
    0x34e2,  0x24c3,  0x14a0,  0x0481,  0x7466,  0x6447,  0x5424,  0x4405,
```

```
    0xa7db,  0xb7fa,  0x8799,  0x97b8,  0xe75f,  0xf77e,  0xc71d,  0xd73c,
    0x26d3,  0x36f2,  0x0691,  0x16b0,  0x6657,  0x7676,  0x4615,  0x5634,
    0xd94c,  0xc96d,  0xf90e,  0xe92f,  0x99c8,  0x89e9,  0xb98a,  0xa9ab,
    0x5844,  0x4865,  0x7806,  0x6827,  0x18c0,  0x08e1,  0x3882,  0x28a3,
    0xcb7d,  0xdb5c,  0xeb3f,  0xfb1e,  0x8bf9,  0x9bd8,  0xabbb,  0xbb9a,
    0x4a75,  0x5a54,  0x6a37,  0x7a16,  0x0af1,  0x1ad0,  0x2ab3,  0x3a92,
    0xfd2e,  0xed0f,  0xdd6c,  0xcd4d,  0xbdaa,  0xad8b,  0x9de8,  0x8dc9,
    0x7c26,  0x6c07,  0x5c64,  0x4c45,  0x3ca2,  0x2c83,  0x1ce0,  0x0cc1,
    0xef1f,  0xff3e,  0xcf5d,  0xdf7c,  0xaf9b,  0xbfba,  0x8fd9,  0x9ff8,
    0x6e17,  0x7e36,  0x4e55,  0x5e74,  0x2e93,  0x3eb2,  0x0ed1,  0x1ef0,
};

/*eof*/
```