

Entegris IntelliGen & InVue Communications Protocol



Table of Contents

TABLE OF CONTENTS.....	2
1. GENERAL.....	3
SCOPE.....	3
SERIAL PARAMETERS.....	3
PACKET SYNCHRONIZATION.....	4
COMMAND/RESPONSE PACKET FORMAT:.....	5
2. COMMAND CODES.....	6
3. ENTEGRIS NETWORK PROTOCOL.....	7
OVERVIEW.....	7
<i>Command Header</i>	7
<i>Response header</i>	9
INTERFACE LOCKING THEORY OF OPERATION.....	10
4. CRC CALCULATION.....	11

1. General

The IntelliGen Mini or later Entegris photochemical dispense systems and second generation InVue products use this serial interface to communicate with front end software for configuration and monitoring using a simple binary block protocol with a master command/slave response sequence. The PC initiates a packet with an address. The system with the indicated address receives the packet, operates on it according to the command and returns with a reply packet. Every command and response-packet are protected with a 16 bit CRC. The algorithm for the CRC calculation is discussed in Appendix D.

Scope

IntelliGen Mini, IntelliGen AFS, IntelliGen ULV, IntelliGen LV, IntelliGen MV, IntelliGen HMV, IntelliGen HV, IntelliGen HV-20, IntelliGen HV-50, InVue GV148

Serial Parameters

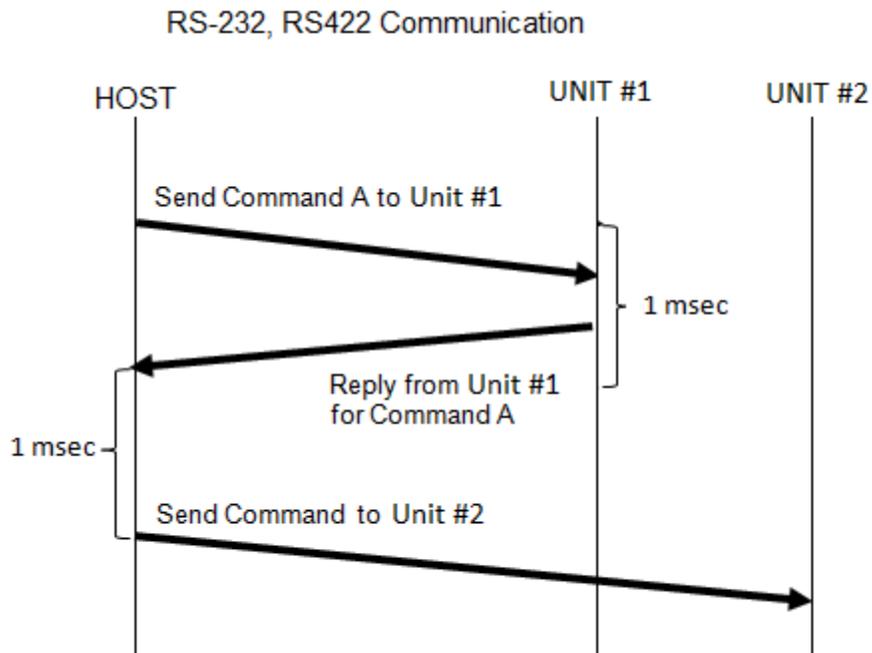
The serial protocol uses normal PC serial ports to communicate. The technical parameters are:

Bit:	RS-232 or RS-422 depending on the interface module	
Rate:	IntelliGen Mini	19,200 default (57600 for certain OEM versions)
	IntelliGen AFS	38,400 or 19,200
	IntelliGen ULV	57,600 default
	IntelliGen LV	57,600 default
	IntelliGen MV	57,600 default (38,400 for certain OEM versions)
	IntelliGen HMV	57,600 default (19,200 for certain interface module types)
	IntelliGen HV	57,600 default (19,200 for certain interface module types)
	IntelliGen HV-20	57,600 default (19,200 for certain interface module types)
	IntelliGen HV-50	57,600 default (19,200 for certain interface module types)
	InVue GV148	57,600 default
Data:	8 bits, No Parity	

Packet Synchronization

To synchronize the unit receiver and slave transmitter control the master host must allow 1mS between receiving the last byte of a reply from a device and sending the next command. The slave device shall have a minimum of 1 mS before sending a reply. There is no maximum specification but design guidelines strive to reduce the absolute maximum processing time to less than 500 mS (typically EEPROM writes can take 100-200 mS).

- Minimum time from end of Master Send and start of Slave Send: 1mS
- Minimum time from end of Master Receive and Master Send: 1 mS



Command/Response Packet Format:

Byte Position	Byte	Description
0	Address	Unique address ID from 1 to 63 of the dispense system the packet is intended for. For reply packets the Address byte shall return the address of the unit sending the packet.
1	Command or Response	Command or response code for this packet.
2	Size LSB	The size is the total number of bytes of the whole packet including the header, data (if any) and CRC bytes. The minimum number for a packet with no data is 6.
3	Size MSB	
4...Size-2	<Data>	The data payload, if any, is here. There must always be an even number of bytes of data.
Size-2	CRC LSB	The CRC field is the 1's complement of the running CRC total of the preceding data and header bytes.
Size-1	CRC MSB	

This packet structure is used to send commands to the unit and to receive data and responses. For every valid packet sent the dispense system responds with a packet. The dispense system will never transmit without first receiving a command.

2. Command Codes

Overview

Command codes are specific to each device. See individual product Entegris Commands Documents for specific command code details.

3. Entegris Network Protocol

Overview

The Entegris Network Protocol allows system commands to be sent through a wired (Ethernet) or wireless LAN to Entegris systems. It is a simple client / server model and includes additional features for querying the server's interfaces as well as a simple cooperative interface locking mechanism. The protocol uses a TCP/IP port to send and receive data. The data can be a single UDP datagram or a TCP connection. In the latter case the connection is opened, the command sent and the connection is closed after the server returns the response. Every command has a single response.

The Entegris Network Platform includes a server for the Network Protocol.

Command Header

FIELD	SIZE
Command Code	UINT16
Data size	UINT16
Timeout	UINT16
Control bits	UINT16
Logical Serial	UINT16
Lock Code	ULONG
Reserved	ULONG

The header has a total of 18 bytes, followed by the data (if any).

Command Codes

Command name	Value	Description
SEND_PACKET	0	Send serial data, return the response data.
GET_INTERFACES	1	Returns how many logical serial interfaces available on this server
GET_NAME	2	Returns the null padded name for the server; uses either the name specified in the configuration file (if any), or the value returned from <code>gethostname(2)</code>
GET_VERSION	3	Returns a null padded string with the model and version number of this server
QUERY_INTERFACE	4	Returns 5 pieces of information about a specific interface: enabled (LONG), baudrate (LONG), requests served (ULONG), device name (variable), device name string (variable)
LOCK_INTERFACE	5	Locks the specified serial interface to a particular client; this prevents other users from using the interface at the same time
UNLOCK_INTERFACE	6	Unlocks an interface previously locked by the LOCK_INTERFACE command

Size

This is a UINT16 that describes the size of the entire packet, including the header and any data. In the case of a UDP datagram the size of all data must be less than the MTU which is typically 1024 bytes. TCP connections do not have a size restriction.

Timeout

This is a UINT16 that specifies a timeout for the serial command in milliseconds. If the server does not get a response from the unit in this amount of time it will respond with ERR_SERIAL_READ_TIMEOUT (1015). If a timeout of 0 is specified, the server will wait the maximum amount of time possible, which is 64 seconds.

Control bits

This is a UINT16 that controls various behaviors of the client/server communications. The meaning of the bitfields is as in the following table.

Bit Number	Name	Description
0	CTRL_CHK_CRC_PACKET	If set, the server will check the CRC on the packet from the client to the unit before sending it to the unit. On CRC failure, it will respond to the client immediately without ever making contact with the unit.
1	CTRL_CHK_CRC_DEVICE	If set, the server will check the CRC on the response packet from the unit to the client. If CTRL_AUTO_RETRY is not set, then it will return a CRC_FAILED error to the client. If CTRL_AUTO_RETRY is set, it will retry a configurable number of times before returning an error to the client.
2	CTRL_AUTO_RETRY	If set, the server will automatically retry failed CRC's from the unit, based on the logic explained in CTRL_CHK_CRC_DEVICE above.
15	CTRL_LOCK_OVERRIDE	If set, the server will allow commands to succeed on otherwise locked interfaces. This is to ensure that a misbehaving or malicious client cannot lock out valid users.

Logical Serial

This is a UINT16 that specifies which logical serial device to access with this command. Note that this value is ignored on GET_INTERFACES, GET_NAME, and GET_VERSION commands.

Lock Code

This is a ULONG that specifies the lock code to be used when executing “locked” commands. If the interface is not locked, this MUST be set to zero.

Reserved

ULONG which MUST be set to zero.

Response header

The response header is similar to the command header but only defines the first two fields:

FIELD	SIZE
Return Code	UINT16
Data size	UINT16
Reserved	ULONG
Reserved	ULONG

Return Codes

Error Code	Value	Description
ERR_NO_ERROR	0	The command succeeded; any response data is available in the rest of the packet
ERR_INTERNAL	1001	An internal server error occurred
ERR_UNKNOWN_COMMAND	1002	The client sent a command that this server doesn't understand
ERR_MALFORMED_PACKET	1003	The command packet had invalid data in it
ERR_UNKNOWN_SERIAL_PORT	1004	A command that requires a logical serial port (SEND_PACKET or QUERY_INTERFACE) was sent with an invalid logical serial number
ERR_DISABLED_SERIAL_PORT	1005	A command was sent to a logical serial port that is currently disabled
ERR_INPUT_INVALID_CRC	1006	A SEND_PACKET command was sent, but the packet failed the CRC check
ERR_EXTRA_DATA	1007	A command was sent with more data than the server expected
ERR_INVALID_SERIAL_DATA	1008	A SEND_PACKET command was sent, but the data portion of the packet was too short to be a valid serial command
ERR_LOCKED_INTERFACE	1010	A command was sent to a locked interface, but the lock code did not match the code that locked the interface.
ERR_NOT_LOCKED_INTERFACE	1011	An UNLOCK_INTERFACE command was sent to

		an interface that is not currently locked
ERR_INVALID_LOCK_CODE	1012	An UNLOCK_INTERFACE command was sent, but the lock code did not match what was stored in the server; this is deprecated and not used
ERR_SERIAL_COMMAND_FAILED	1013	A SEND_PACKET command was sent, but sending the command to the unit failed
ERR_SERIAL_SHORT_READ	1014	A SEND_PACKET command was sent, but the response from the unit was shorter than expected
ERR_SERIAL_READ_TIMEOUT	1015	A SEND_PACKET command was sent, but the server timed out waiting for response data from the unit
ERR_SERIAL_OUTPUT_INVALID_CRC	1016	A SEND_PACKET command was sent, but the CRC on the unit response was invalid

Interface Locking Theory of Operation

The LOCK_INTERFACE and UNLOCK_INTERFACE commands provide a way to ensure exclusive access to one of the interfaces (COM ports) on the server (Network Platform). To lock an interface, a client sends a LOCK_INTERFACE command with specific lock code, known only to the client. On subsequent SEND_PACKET, LOCK_INTERFACE, and UNLOCK_INTERFACE commands the client will send the same lock code in the header of the packet. The server will check that this lock code matches the lock code that was used to lock the interface, and only allow access if the codes match. When the client is done with exclusive access, it should use the UNLOCK_INTERFACE command with the same lock code to unlock the interface.

If the client is NOT using locked commands, the lock code in the header must explicitly be zero.

To provide a measure of safety against misbehaving or malicious clients, the SEND_PACKET, LOCK_INTERFACE, and UNLOCK_INTERFACE commands provide a CTRL_LOCK_OVERRIDE bit in the control bits. Generally the CTRL_LOCK_OVERRIDE bit allows access despite the interface being locked by another client.

4. CRC Calculation

The dispense system uses a 16 bit Cyclic Redundancy Check (CRC) calculation on blocks of data to validate the integrity of the block of data. There are three places where CRCs are used: all communication packets both to and from the dispense system, for data stored in internal EEPROM non volatile storage and for the flash firmware image.

The polynomial calculation is theoretically straightforward but it can be difficult to implement the mathematically equivalent algorithm. Since any host PC wishing to communicate with the dispense system must calculate and use the CRC values the C source code used in the dispense system is shown below. This source code was designed to run on a little Endean processor with 16 bit ints and is included here for informational purposes only.

```
CRC16_Init();
unsigned short CRC16_Byte(unsigned short byte, unsigned short crc);
unsigned short CRC16_Str(unsigned short *str, int length, unsigned short crc);
void CRC16_Add(unsigned short* str, int length);
int CRC16_Valid(unsigned short *str, int length);
/* private definitions */

#define CRC16TBL_SIZE 256
static unsigned short crc16tbl[CRC16TBL_SIZE];
static int crc16rdy = 0;

//
// CRC16_Init
//
// Description:
//   Compute entries for CRC-16 lookup table. Must be called one time only
//   before other CRC functions can be called.
//
CRC16_Init(void)
{
    int i,j,k;

    if (crc16rdy)
        return;

    for(i = 0; i < CRC16TBL_SIZE; i++) {
        j = 0;
        for(k = i; k != 0; k /= 2) {
            j = (j + k) % 2;
        }
        crc16tbl[i] = (i << 7) ^ (i << 6) ^ (j * 0xC001);
    }
    crc16rdy = 1;
}

//
// CRC16_Word
//
```

```
// Description:
//     compute CRC-16 one word at a time
//
// Return Value:
//     accumulated new crc value
//
unsigned short CRC16_Word(unsigned short word, unsigned short crc)
{
    return (crc16tbl[0xff & (word ^ crc)] ^ (crc >> 8));
}

//
// CRC16_Str
//
// Description:
//     compute CRC-16 for a string of words
//
// Return Value:
//     CRC value for the string
//

unsigned short CRC16_Str(unsigned short *str, int length, unsigned short crc)
{
    int i;

    for(i = 0; i < length; i++) {
        crc = CRC16_Word(str[i]&0xff, crc);
        crc = CRC16_Word(str[i]>>8,  crc);
    }
    return crc;
}

//
// CRC16_Add
//
// Description:
//     compute complemented CRC-16 for word string; append to string
//
void CRC16_Add(unsigned short* str, int length)
{
    str[length] = ~CRC16_Str(str, length,0);
}

//
// CRC16_Valid
//
// Description:
//     test string for correct (complemented) CRC-16
//
// Return Value:
//     true if the CRC is valid for the block
//

int CRC16_Valid(unsigned short *str, int length)
{
    return (CRC16_Str(str, length,0) == 0xB001);
}
```