Gilson Communications GECP Sample Program Documentation

Revision History		
Rev.	Summary of Change	Date
01	Original	01/26/18

Table of Contents

Table of Figures

Actions Performed by Sample Program

The sample program demonstrates use of the Gilson Communications assembly. The sample program includes the following operations:

- Get a list of available serial ports.
- Connects to the instrument using the Gilson Communications assembly.
- Communicates with the instrument via commands specific to instrument.
- Receives responses from the instrument.

Components

GECP Sample Application.sln

- Microsoft Visual C# 2015
- .Net Framework 4.5.2

GECP Sample Application.cs

- Contains the WinForm component of the solution.
- Contains well documented sample code to communicate with an instrument over GECP.

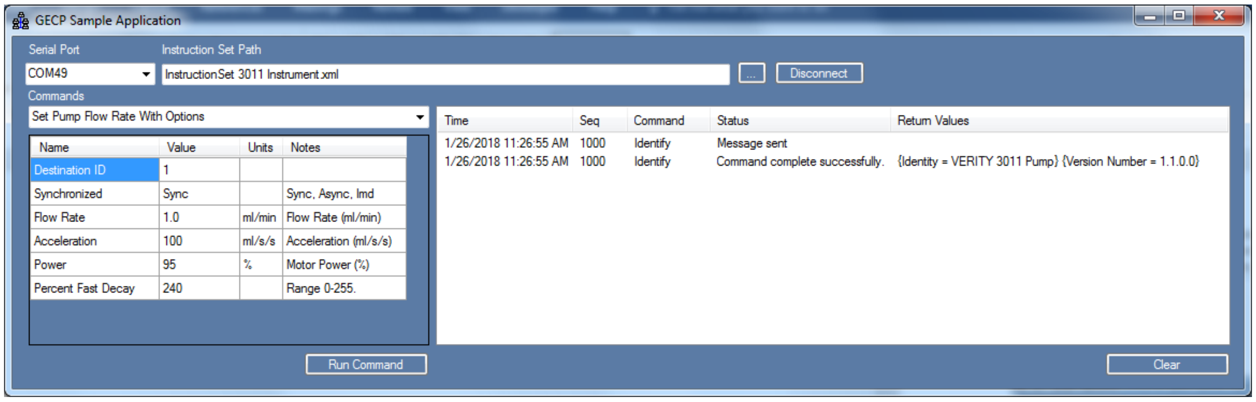
Gilson Communications.dll

- Gilson communication backbone for instrument control over Ethernet, GECP, GSIOC and serial.
- All communication through the assembly is accomplished using Gilson Ethernet protocols and procedures.
- Documented in Gilson Communications.chm.
- Version 3.2.0.0

ISReport.xsl

1. When copied into the folder containing the instruction set, this will format the instruction set xml for ease of viewing when the instruction set is opened in a web browser.

Instrument Connection



Name	Value	Units	Notes
Destination ID	1		
Synchronized	Sync		Sync, Async, Imd
Flow Rate	1.0	ml/min	Flow Rate (ml/min)
Acceleration	100	ml/s/s	Acceleration (ml/s/s)
Power	95	%	Motor Power (%)
Percent Fast Decay	240		Range 0-255.

Time	Seq	Command	Status	Return Values
1/26/2018 11:26:55 AM	1000	Identify	Message sent	
1/26/2018 11:26:55 AM	1000	Identify	Command complete successfully.	{Identity = VERITY 3011 Pump} {Version Number = 1.1.0.0}

Figure 1 - Screen shot of Verity 3011 Pump connected via GECP

Connecting to a GECP instrument requires a comm port and an instruction set.

1. Select a Serial Port.

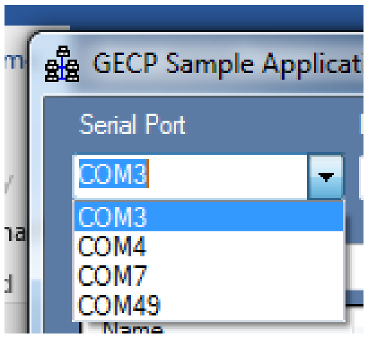


Figure 2 - Serial Port List

   1.1.
   1.2. The serial port list is populated when the application starts from the list of all known serial ports as provided by Windows.

2. Select an Instruction Set Path. The Browse button can be used to show an Open file dialog to assist in selecting an instruction set. Instruction sets must have an xml extension.

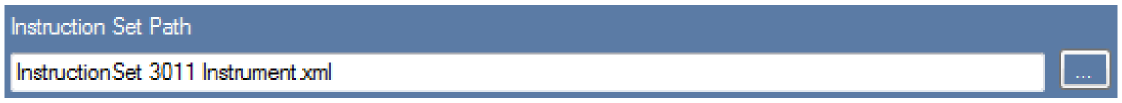


Figure 3 - Instruction Set Path

3. Connect by clicking the Connect button.

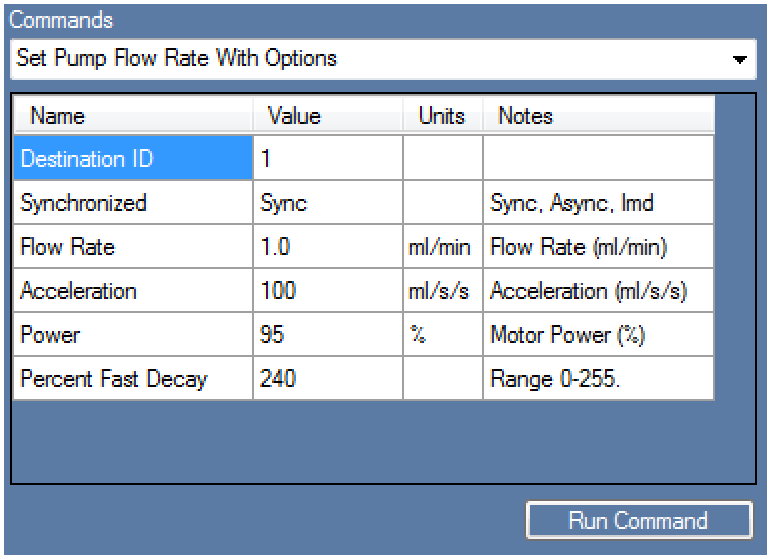
Figure 4 - Connect Button

3.1. This will create a new GilsonSerial object that contains an open comm port (see `private void Connect(string port, string instructionSetPath)`).

4. Once the instrument is connected, the Commands drop down should be populated.

Instrument Control

Controlling the instrument requires creating a SerialCommand object and sending it to the instrument.



Name	Value	Units	Notes
Destination ID	1		
Synchronized	Sync		Sync, Async, Imd
Flow Rate	1.0	ml/min	Flow Rate (ml/min)
Acceleration	100	ml/s/s	Acceleration (ml/s/s)
Power	95	%	Motor Power (%)
Percent Fast Decay	240		Range 0-255.

Figure 5 - Command Selection and Configuration

1. Select a command from the Commands list.
   1.1. The command parameters table will be updated to show the available parameters for the selected command.
      1.1.1. Name -> the name of the parameter from the instruction set.
      1.1.2. Value -> the value for the parameter – the value is initially populated based on the default value in the instruction set.
      1.1.3. Units -> the units for the value, if applicable.
      1.1.4. Notes -> this can include ranges, possible options, or more details on the command parameter.
2. Enter the desired values for each command parameter.
3. Execute the command by clicking the Run Command button.
   3.1. If the command cannot be executed due to invalid parameter values, invalid instrument state or other reasons, a response will be added to the response list.
   3.2. A SerialCommand is created and send to the GilsonSerial object created by the Connect action. For this sample, a new SerialCommand is created based on the Commands and parameters table. Other alternatives are available.
      3.2.1. Alternative -> create the command using strings

3.2.1.1. This option allows for creating commands by setting the values in an empty SerialCommand object. The CommandName, Parameters and SequenceNumber must be set before sending the command.

3.2.2. Alternative -> create the command using the instruction set command object

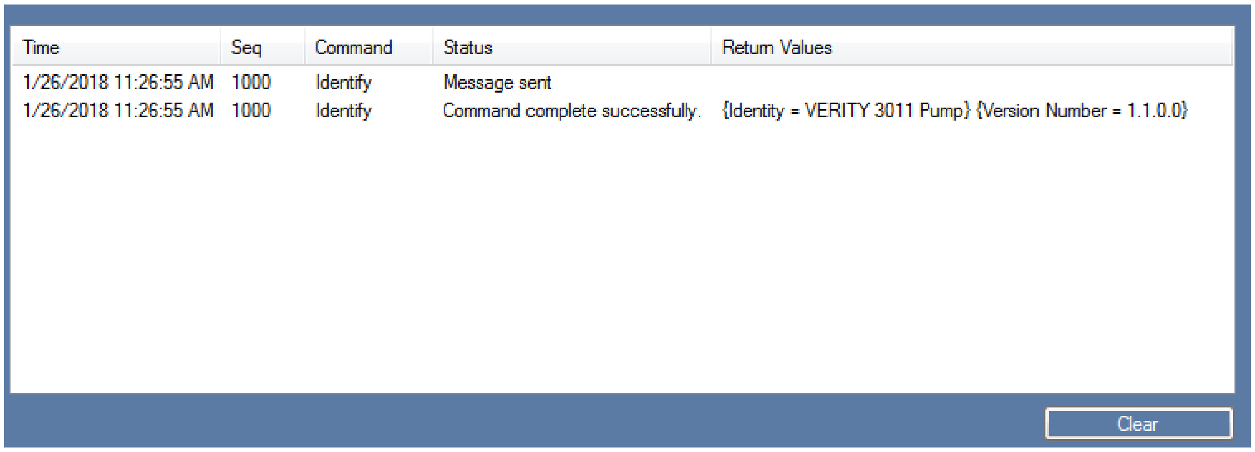
3.2.2.1. This option uses the instruction set to create the SerialCommand object. This is a safe and easy way to create a SerialCommand object that has the correct setting already configured. All that is required is to update any parameter values that need changed from the default. Parameters are indexed by number or by name. The sequence number should also be updated.

3.3. Commands are executed by calling GilsonSerial.Send(…). An alternative is to use GilsonSerial.SendAndWait(…).

3.3.1. GilsonSerial.Send(…) -> This will send the command to the instrument using the GECP protocol. The function will return as soon as the command has been queued to send to the instrument. All responses to the command will be sent on the EthernetDataDelegate event (see `private void ethernetResponseData(EthernetReturnBase data)`).

3.3.2. GilsonSerial.SendAndWait(…) -> This will send the command to the instrument using the GECP protocol. The function will block until a response is received or until the timeout has elapsed. The function returns the SerialResponseData. This method allows for simple queries of the instrument to retrieve data. All responses to the command will still be sent on the EthernetDataDelegate event (see `private void ethernetResponseData(EthernetReturnBase data)`). `This function does not block events and other commands (on other threads) from executing.`

Instrument Response



Time	Seq	Command	Status	Return Values
1/26/2018 11:26:55 AM	1000	Identify	Message sent	
1/26/2018 11:26:55 AM	1000	Identify	Command complete successfully.	{Identity = VERITY 3011 Pump} {Version Number = 1.1.0.0}

Figure 6 - Response List

When commands are sent, the instrument sends a response. All responses (and unsolicited events) are captures by an event handler, parsed and logged to a list.

1. Response columns are defined as follows

1.1. Time -> the time the message was sent or the response or event was received – this is local PC time
1.2. Seq -> the sequence number of the data
1.3. Command -> the command name of the data
1.4. Status -> information related to the data being logged
    1.4.1. Commands -> the status will say Message sent
    1.4.2. Responses -> the status will show the response code sent from the instrument

ResponseCode	Message
GECPReserved	GECP Reserved
GECPSuccess	Success
GECPCommandAccepted	ACK
GECPCommandFinished	Command complete successfully
GECPDeviceBusy	Error: Device is busy due to other command execution
GECPPeriodicMessage	Intermediate or periodic data message
GECPSequenceIDError	Error: Error on request/sequence ID
GECPInvalidDestinationID	Error: Invalid destination/device ID
GECPInvalidCommandName	Error: Invalid Command Name
GECPCommandNotAllowed	Error: Command not allowed in this state
GECPReceiveTimeoutError	Error: Receive Timeout error
GECPInvalidCommandParameter	Error: Invalid command parameter
GECPInvalidMessageStartEndTag	Error: Invalid/Missing message start/end tags
GECPCommandNotExecuted	Error: Command unexecuted due to error
GECPInvalidCommandStartEndTag	Error: Invalid/Missing command start/end tags
GECPCommandWarning	Error: Command finished with a warning
GECPInvalidMessageParameter	Error: Invalid/Missing message parameters
GECPCommandAborted	Error: Command aborted and flushed from the command queue

Figure 7 - Response Codes

2. The response list can be cleared by clicking the Clear button.