NORGREN

Kloehn Versa Pump 6
55 Series
Syringe Dispense Module

HARDWARE USER MANUAL

Norgren Kloehn, Inc., 10000 Banburry Cross Dr., Las Vegas, Nevada 89144 Phone: (702) 243-7727

NOTICE

Norgren-Kloehn Inc. maintains a continuous process of product evaluation and improvement. This is done through in-house engineering evaluations, market research, and customer feedback. As a result of this process, Norgren-Kloehn offers the most capable and sophisticated syringe pumps and valve drives in the world today, at competitive prices. To sustain this effort, Norgren-Kloehn Inc. reserves the right to evolve and upgrade its standard products at Norgren-Kloehn's discretion.

Norgren-Kloehn Inc. has a policy of making every effort to ensure all upgrades are fully backwards compatible with earlier versions. We have been successful in nearly all respects, producing better and more capable versions of our standard products that can be seamlessly integrated into existing system designs without increasing prices. However, differences may exist in internal parts and materials, and rarely, in firmware commands.

If such differences may impact your product certifications, or if a unique configuration must be maintained over long production runs, contact Norgren-Kloehn Customer Service to inquire about assigning a unique part identification number for your application. Alternatively, you may request to add your name to the list of customers to be notified of changes prior to release of standard product upgrades.

Norgren Kloehn Customer Service: (702) 243-7727 or 1-800-358-4342.

Contents

List of Figures

List of Tables

1 Safety Messages

The following hazard statements pertain to the VersaPump 6. Read and understand these messages before continuing. Misuse or use in a manner not consistent with the User's Manual is not authorized and may result in a loss of warranty coverage.

Table 1 1 General Safety Messages

CAUTION: Pump motors can exceed 70°C (158°F) during extended operation. It is recommended to shield these parts from inadvertent operator contact.

CAUTION: Care should be taken in instrument design and usage to prevent the exposure of flammable fluids to the motor or the valve during operation or service due to the elevated temperatures at which these devices may operate. Fluid lines should be routed to minimize any such risk.

CAUTION: When dispensing fluids other than water, cleaning procedures are recommended. Solutions that produce or contain particulates, such as saline solutions or the products of reactions between different solutions should not be allowed to stand for extended periods of time. As sharp-edged particulates accumulate on the inner surfaces of the valve and syringe parts, damage to diaphragms or syringe piston seal materials may occur, greatly shortening their operating life. Saline solutions should never be allowed to stand long enough for salt crystals to come out of solution.

To preclude a problem with particulate accumulation, flush the pump with a buffer solution or DI (de-ionized) water after each use. This is also advised between different solutions if cross-contamination is a potential issue. Sufficient buffer or DI water should be used to adequately dilute the possible residues from previous solutions.

Do not attempt to disassemble the pump or its valve and syringe parts for cleaning. This is a factory-level operation.

Table 1 2 Section 3 Safety Messages

CAUTION: Make sure that the card edge connector is oriented with the power pins at the bottom of the connector as shown in Figure 2–7 and Figure 3–7.

CAUTION: Do NOT have more than two RS-485 terminators switched on regardless of the number of devices on the bus. Use ONLY one network on at each end of the overall bus wiring.

2 Getting Started

This section guides a first-time user through the basic setup required to control a single VersaPump 6 from a PC.

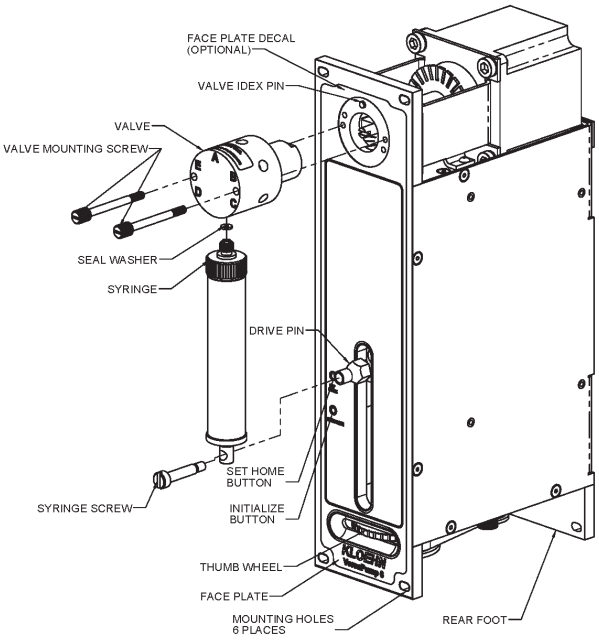
1. Take Inventory
2. Install the Valve
3. Install the Syringe
4. Set the Switches
5. Connect the Power and Communication
6. Set Up Communication
7. Configure the Pump

2.1. Factory Default Settings

The VersaPump 6 ships with the following default settings:

- Valve type 1 (3-way non-distribution)
- Address: 1
- DIP switches (4) closed
- Speed: 5,000
- Baud rate: 3 (9600 bits per second)
- Protocol: DT

Figure 2–1 Exploded View of Pump and Pump Parts



2.2. Take Inventory

A complete pump system will require the components listed in Table 2–1.

Table 2–1 Required Items for a Basic Installation and Operation

Quantity	Item
1	VersaPump 6 Drive Module
1	*Valve
1	*Syringe
2(mm)	• Teflon washers (one per port used plus a syringe)
1	*Power supply (24 to 30 VDC, 40 Watts, recommended P/N 17732)
1	*Communications cable, PC-to-pump, P/N 17734
1	*Card Edge Adapter board kit, P/N 23428 or Card Edge Connector, P/N 23277 or equivalent

* Contact Norgren Kloehn Customer Service for pricing and availability of recommended components.

2.3. Install the Valve

1. Turn on the power to the pump and press the **Initialize** button on the front panel. This is the lower of the front two panel buttons. Verify the slot in the valve drive shaft is horizontal.
2. When initialization completes, insert the valve into the faceplate so that the valve index pin engages a corresponding hole in the valve and the valve drive motor shaft slot. Make sure the valve seats flush onto the pump faceplate.
3. Install the two valve screws through the valve and into the faceplate. Tighten the screws firmly, but only finger tight. Over-tightening can damage the valve.

2.4. Install the Syringe

1. Make sure the syringe is in the full dispense position. The syringe drive pin will be in the full dispense position (top-of-stroke) after initialization during the valve installation processes.
2. Carefully place a seal washer on the end of the syringe and screw the syringe into the valve syringe port. Be careful to avoid misalignment and damage to the threads and seal washer. If resistance is encountered during installation, remove the syringe and check the seal washer for damage. If the seal washer is not damaged, repeat the attempt. Otherwise, replace the seal washer. When properly installed there will be no visible threads below the valve body.
3. Rotate the plunger button so that the hole is aligned with the drive pin on the pump. Rotating the plunger button will not cause damage to the syringe assembly.
4. Install the syringe screw through the plunger button and screw into the drive pin. The syringe screw must be securely tightened to ensure accurate pump operation.

NOTE: Before using the pump, be sure to calibrate the syringe as described in section 2.7.2.

The table below shows the compatibility between syringes and valves for the V6 series pumps. The table lists the ports that will be partially or fully blocked when used with the corresponding syringe. All syringes 1.25mL and smaller will work with all valves and will not block any of the valve ports.

"X" – Denotes the port(s) that is (are) blocked. There is no room to install a fitting into that port. The port is open to the atmosphere.

"*X" – Denotes the port(s) that is (are) partially blocked but may accept a custom, smaller fitting. Kloehn does not have a suitable recommendation for this fitting.

√ – Indicates there is no conflict.

ND – Non-Distribution valve type.

dist – Distribution valve type.

	50 mL	25 mL	10 mL	5 mL	2.5 mL	≤1.25 mL
3-way ND	√	√	√	√	√	√
3-way dist	√	√	√	√	√	√
4-way ND	√	√	√	√	√	√
4-way dist	C, D	√	√	√	√	√
5-way dist	√	√	√	√	√	√
6-way ND	√	√	√	√	√	√
6-way dist	D	D	D	D	*D	√
8-way dist	*D, E, *F	E	E	E	*E	√
12-way dist	√	√	√	√	√	√

2.5. Set the Switches

2.5.1. Com Setup Switch

Make sure that the four toggles located on the Com setup switch shown in Figure 2–2 and Figure 2–3 are set to the **On** position.

Figure 2–2 Com Setup Switch

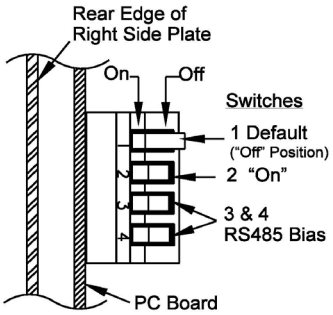


Figure 2–3 Pump Connections with Card Edge Connector

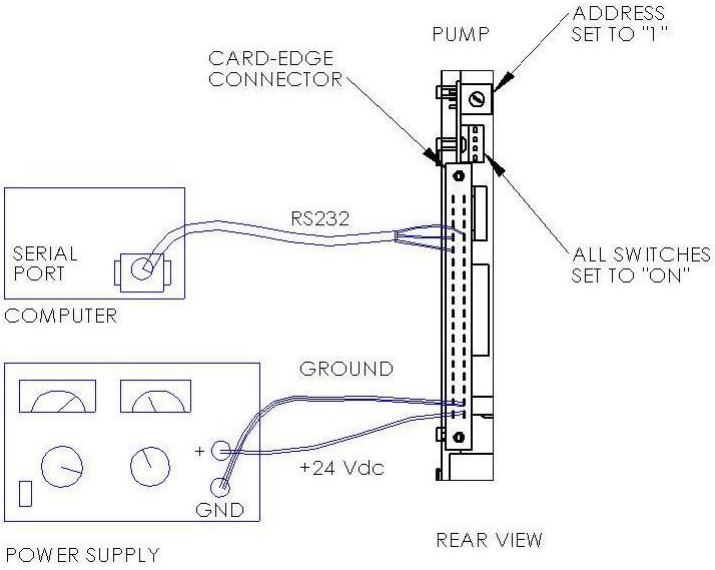
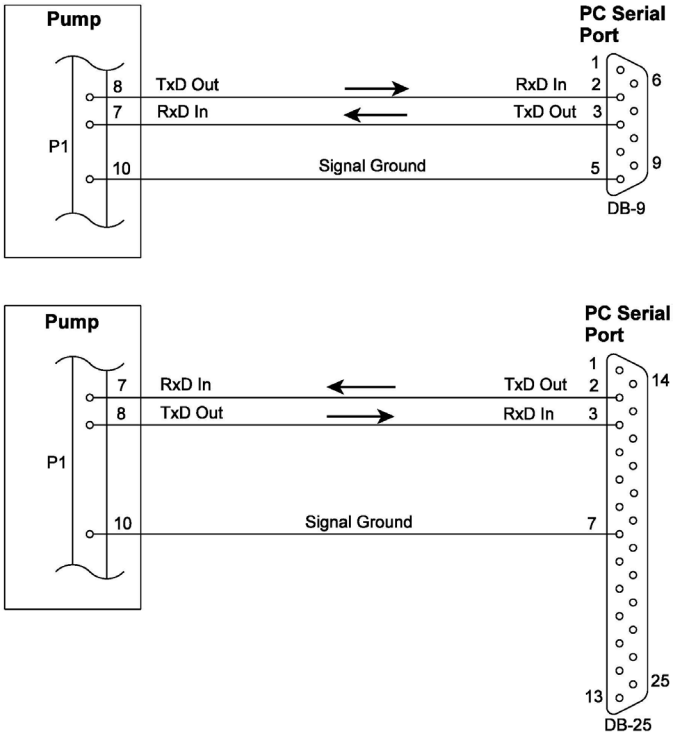


Figure 2–4 RS-232 Communications Cable Wiring



NOTE: Pins 2 and 3 are reversed between the DB-9 and the DB-25.

2.5.2. Set the Address Switches

The address switches shown in Figure 2–3 are used to set the device address number. Use a small screwdriver to set the switch to **1**.

All communications with the pump begin by sending the address number. The pump address can be set with the address switch or by wiring on the address pins of the card edge connector. The card edge wiring method permits a wire harness to set an address when multiple pumps are used in an instrument.

2.5.3. Connect the Power and Communication

This section describes how to connect the pump to power and a PC using the Kloehn Starter Kit. If card edge connector with user-supplied wiring is used, see *Connecting the Power* without the Starter Kit in section 3.6.4.

The Norgren Kloehn Starter Kit (P/N 23427) contains the following items:

- 24VDC power supply (P/N 17732) with power cables (Figure 2–7)
- RS-232 communications cable (P/N 17734) (Figure 2–6)
- Card Edge Adapter Board (P/N 23428) (Figure 2–5)
- Disk with software and manual (P/N 23422)
- Installation instruction sheet

2.5.4. Connect the Card Edge Adapter Board to the Pump

The Card Edge Adapter Board (P/N 23428) attaches to the pump to make it compatible with Kloehn pump accessories, including the power supply cable and the communications cable.

Insert the card edge connector on the board onto the card edge of the pump (Figure 2–5). Locate the 1-inch connectors near the center of the rear of the pump.

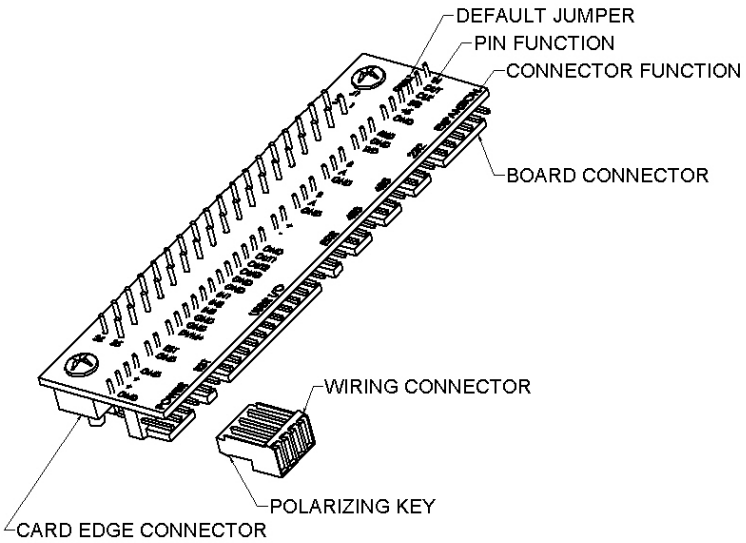
When inserting a Wiring Connector into an Card Edge Adapter Board Connector, make sure the polarizing key is oriented toward the locking tab, as shown in Figure 2–5, and the pins are properly aligned with the connector.

2.5.5. Connect the Communications Cable

The communications cable (Com cable) has a DB-9 connector on one end and a 3-pin, 1-inch connector on the other end as shown in Figure 2–6.

1. Plug the 3-pin connector into the adapter board connector labeled **232**. Note the polarization of the locking tab on the 3-pin connector as shown in Figure 2–6.
2. Plug the DB-9 connector into the serial port on the PC.

Figure 2–5 Card Edge Adapter Board (P/N 23428)



2.5.6. Connect the Power Cables

The 24VDC power supply (P/N 17732) is provided with two cables as shown in Figure 2–7. The 24VDC cable is integral to the supply and has a 4-pin connector attached. This cable connects to the pump via the card edge adapter.

1. Plug the 4-pin connector into the adapter board connector labeled "POWER".
2. Observe the locking tab polarization.
   A separate cable is provided to connect the power supply to an AC power source. This cable has a power switch, a 3-prong wall plug, and a 3-pin power connector.
3. Plug the 3-pin power connector into the mating receptacle in the power supply, as in Figure 2–7.
4. Plug the wall plug into wall power socket.
5. Note the green indicator light on the power supply:

   When the indicator light is off, change the position of the switch on the AC power cable. There will be a slight delay before the indicator lights.

   When the indicator light is on, the power supply is delivering 24VDC power to the 4-pin connector.

NOTE: On the 4-pin DC power connector, the two inner pins are identical positive (+) Power Input, and the two outer pins are identical Ground pins. The power supply has sufficient capacity to power one VersaPump 6.

Figure 2–6 PC Serial Communications Cable P/N 17734

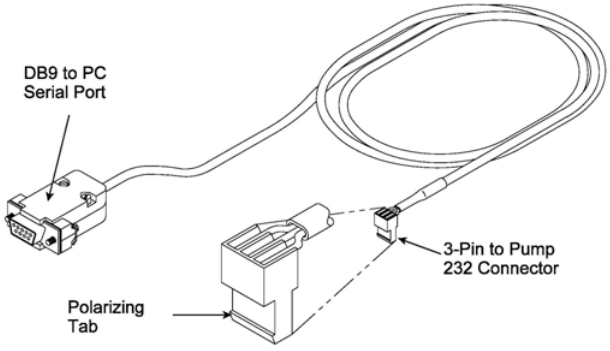
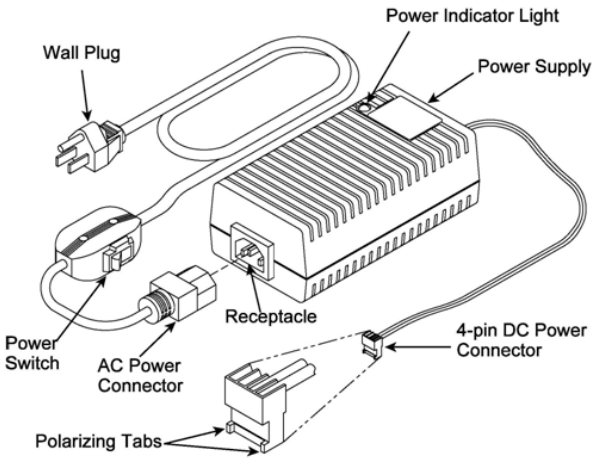


Figure 2–7 24VDC Power Supply P/N 17732



2.6. Set Up Communication

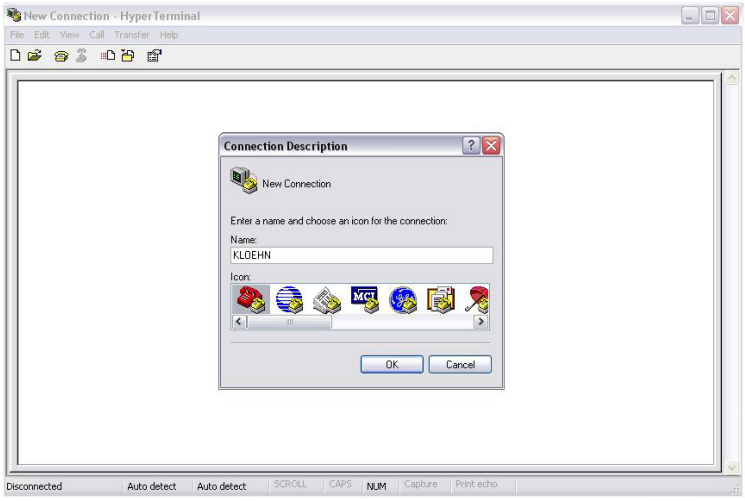
If a communications program has been supplied by Kloehn Co., follow the setup directions supplied with the software. Otherwise use the HyperTerminal program supplied with Windows to verify communications with the pump.

2.6.1. Set Up HyperTerminal

HyperTerminal allows monitoring of all pump responses to commands, without the filtering done by programs. The following setup assumes the default communications parameters of the DT protocol and 9600 baud.

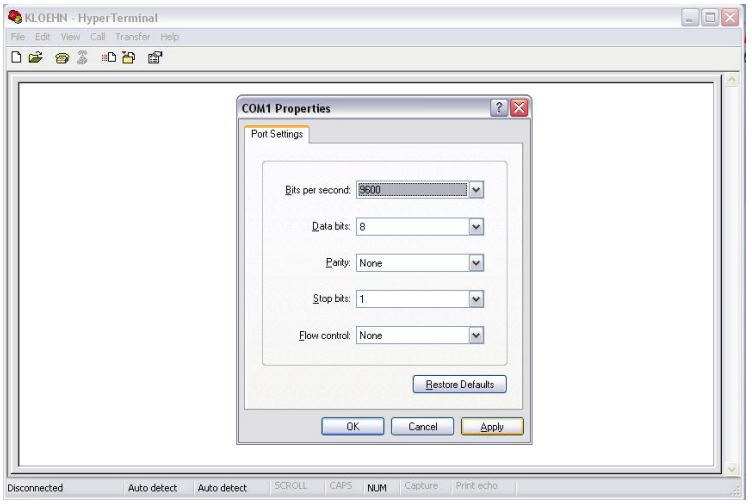
1. Open HyperTerminal from the **Start menu: Start > Program > Accessories > Communication > HyperTerminal**.
2. At the name prompt, type **Kloehn** (suggested) and click **OK**.

Figure 2-8 HyperTerminal Start Window



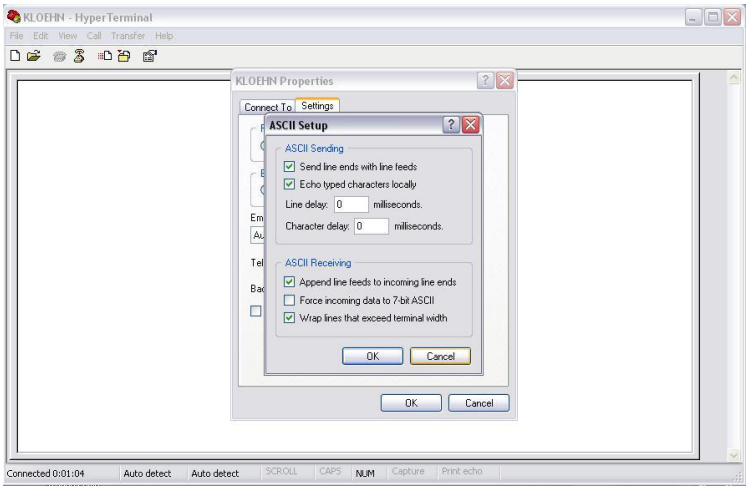
3. Click **Connect Using**, select **"Com 1"**, and click **OK**. This will select the COM 1 serial port. If using another serial port, then select the port as appropriate.
4. In the window that appears, make these entries:
   a. Bits Per Second: **9600**
   b. Data Bits: **8**
   c. Parity: **None**
   d. Stop Bits: **1**
   e. Flow Control: **None**

Figure 2-9 HyperTerminal COM Properties Window



5. Click **OK**.
6. Go to the top line terminal menu and select **File > Properties**.
7. In the Properties window, click the **Settings** tab.
8. Click the **ASCII Setup** button and click to check the following boxes:
   → Send line ends with line feeds
   → Echo typed characters locally
   → Append line feeds to incoming line ends
   → Wrap lines that exceed terminal width

Figure 2-10 HyperTerminal COM Properties Window



9. Click **OK** twice.
10. Go to the top line terminal menu and select **File > Save As**. Save the HyperTerminal setup as "KLOEHN" or as preferred.
11. Click **OK**.
    An icon named Kloehn (or the preferred name) is created in the HyperTerminal Window. Drag and drop this icon onto the desktop and use it for direct access to a pre-set version of HyperTerminal. Each time the HyperTerminal program is required in the future, click this new icon to start a pre-configured HyperTerminal.

2.6.2. Check the Connection

1. Turn on power to the pump. A red LED should be visible, indicating the card edge adapter and power supply are correctly installed on the pump.
2. With a syringe and valve mounted to the pump, press the **Initialize** button. This is the lower of the two front panel buttons. The syringe then moves to a position a small distance below the top of the stroke. This position is internally fixed and is considered the "soft limit". When the initialize move completes, the syringe motor power will be off. (Normally, when a move ends, the motor is left at half-power.)
3. After the pump has initialized, send the command "**/1?R**" and press **Enter**.
4. The pump should respond with "**Ω**" if the pump motor is busy, or "**`**" if the move has finished. If this response is returned by the pump then proceed to the *Configure the Pump* section below. If another response appears, or no response appears, repeat all prior steps to ensure the process was done correctly.

2.7. Configure the Pump

The configuration is set by parameters stored in non-volatile memory (NVM). The parameters determine such things as the type of valve, the communications baud rate, and other operating characteristics.

The parameters are set by the configuration commands, which follow the format of tilde (~), letter, and number. For example, the configuration command "**~V8**" sets the valve type to 6-way distribution:

→ The "**~**" denotes a configuration command.
→ The "**V**" denotes the valve parameter.
→ The "**8**" sets the valve to a six-way distribution.

NOTE: The valve parameter is the only parameter that MUST be set before the pump can be used. See Set the Valve Parameter below for more information.

2.7.1. Set the Valve Parameter

1. Look up the valve parameter that corresponds to the valve type mounted to the pump. The parameters are listed in *Valve Position Commands*, section 5.2.2.
2. Send the command "**/1~Vn**" and press **Enter**. Use the value from the previous step for **n**. For example, the "**\1~V4**" command sets the valve type to be a 4 way distribution valve.

2.7.2. Calibrate the Syringe

Calibrate the syringe zero position prior to first use whenever a new syringe, valve, or syringe washer is installed:

1. With the syringe and valve already mounted to the pump, press the **Initialize** button. This is the lower of the two front panel buttons. The syringe then moves to a position a small distance below the top of the stroke. This position is internally fixed and is considered the "soft limit". When the initialize move completes, the syringe motor power will be off. (Normally, when a move ends, the motor is left at half-power.)
2. Located below the syringe, rotate the thumbwheel to the left to move the syringe piston upward until it barely contacts the top of the syringe. This is the zero position. The position may be slightly below the top-of-stroke position if a small air gap is desired.

NOTE: The zero position MUST be set above the initialize position by at least a small distance or an error will result.

3. Press the **Set Home** button. This is the upper of the front two panel buttons. The syringe moves downward to the Initialize position and then returns to the zero position. The location of the zero position is stored in NVM. This value remains even after the pump's power is turned off and will be remembered by the pump when the pump is powered back on.

NOTE: Do NOT perform the calibration procedure each time the pump is powered on. Only perform the calibration when changing the syringe, valve, or syringe washer.

All the basic setup procedures are now complete.

2.7.3. Example Command Sequence: Positioning the Syringe

This section introduces some basic commands used to position the syringe. The "**R**" at the end of each command means "Run the command now".

1. Enter the command "**/1W4R**" and press **Enter**. This initializes the syringe just as the **Initialize** button did on the front panel.
2. Enter the command "**/1A24000R**" and press **Enter**. This causes the syringe to move to the position 24000 steps below the zero position. "**A**" means "go to the absolute position". The numeric value represents the number of desired steps to move; "**24000**" is half-way down for a 48000-step resolution model or a full aspiration move for a 24000-step resolution model.
3. Enter the command "**/1o3R**" and press **Enter**. The valve will move clockwise (viewed from the front) to port "C". The "**o**" denotes a valve move and the **3** corresponds to port "c" (1= port A, 2= port B, etc.) ("**o**" is lower-case "Oh").
4. Enter the command "**/D16000R**" and press **Enter**. The "**D**" command commands the pump to dispense. The numeric value represents the number of desired steps to move; the syringe will move 16000 steps towards the zero position.

   A syringe that has been aspirated 24000 steps and dispenses 16000 steps is 8000 steps from the zero position. The "8000" is considered the relative syringe position. See Commands, section 5, for more information on constructing and issuing commands.

2.8. Options and Customization

The VersaPump6 allows for different operating interfaces and customization possibilities detailed in the following sections.

→ Options — See section 3.3
→ Supporting Software — See section 3.4
→ Mounting — See section 3.5
→ Power supplies — See section 3.6
→ Custom functionality through programming — See section 3.8

3 Pump Overview

The Norgren Kloehn VersaPump 6 (V6) syringe pump is a programmable, liquid-metering instrument with user-programmable memory and command Input/Output (I/O) capabilities. It is available in 12000, 24000, or 48000 step resolutions for the 6 cm stroke. Two to twelve port valves can be mounted and syringes from 50uL to 50mL can be used, depending on the valve. The unit can accept individual commands or command-string programs via its serial communications interfaces.

Table 3-1 Resolution and Travel Specifications:

Resolution (steps)	Travel (cm/step)	Travel (in/step)
12000	0.0005	0.0002
24000	0.00025	0.0001
48000	0.000125	0.00005

Two-way, serial communications between the V6 and a controlling host is done via an RS-485 or RS-232 interface. Up to 15 addressable pumps or other devices can share a single bidirectional, half-duplex RS-485 communication bus, controlled from a single PC serial port at baud rates from 1,200 to 38,400 baud. Two protocols, DT and OEM, are supported, both of which are fully compatible with the Cavro command set. The unit can be queried for status or operating parameter values at any time by sending individual or groups of commands for immediate or later execution by the pump.

The RS-232 adapter converts the signal levels used by the PC host the RS-485 voltage levels. The adapter also sets an RS-485 bus timer, enabling transmit mode while data is received from the RS-232 port, and then switching to receive mode on the RS-485 bus when a command is completed. This allows any response data from the devices to be sent back to the PC host.

NOTE: Do NOT send data to the RS-232 interface while data is being received from the RS-485 network. This will reset the RS-485 bus timer and cause response data to be lost.

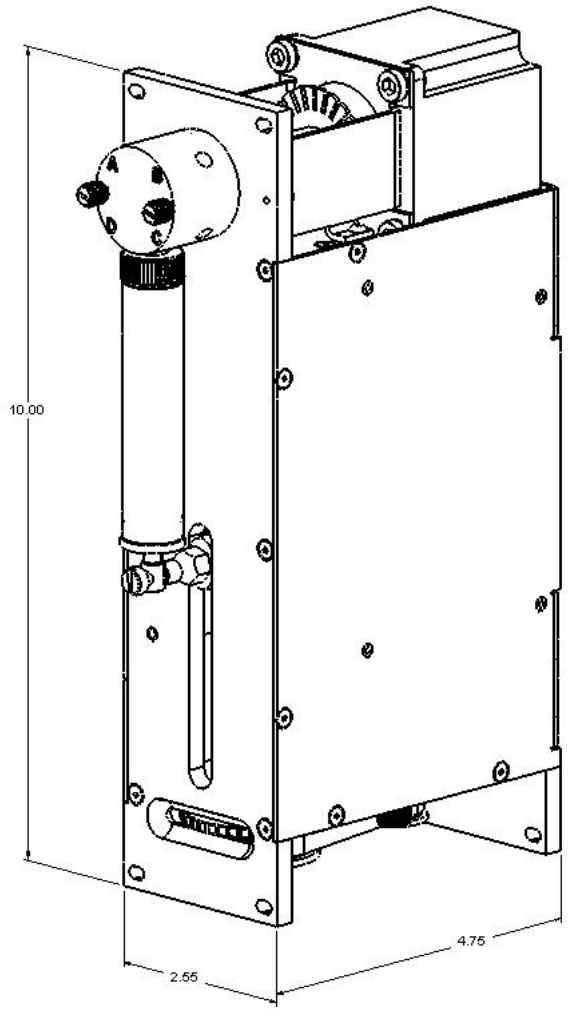
Command strings or programs can be executed from RAM or may be stored into and executed from the non-volatile memory (NVM). Ten programs can be stored in the standard NVM. An expanded NVM increases this to 99 programs. A program in the NVM can be set to self-start when power is first applied to the pump and immediately after a Reset input. Program retention in the NVM is typically greater than 15 years without batteries. Program looping and if-then program flow control is supported.

Three external logic inputs and three outputs permit interfacing to a variety of other devices. One input can be used to halt a dispense in progress. A built-in digital voltmeter (DVM) is included. For applications that require additional input/output capabilities, an I/O expander card is available to provide 8 more inputs and 8 more outputs.

Program test-and-jump instructions allow the pump to respond to external conditions as well as internal program conditions. This allows the pump to respond, react to an external signal condition. External inputs can be used to set programmed operating parameters. The I/O can synchronize two pumps for continuous flow applications. Real-time, remotely-controlled and monitored I/O is supported simultaneously with other pump operations.

The V6 interface is located on a single card edge connector at the rear of the pump. This permits simple connections and the use of modular, plug-in mounting.

Figure 3-1 VersaPump 6 Shown with Syringe and Valve Installed



3.1. Specifications

3.1.1. Environmental

Table 3-2 Temperature and Humidity Specifications

Temperature	Humidity
Operating: -25 to 55°C (-13 to 131°F) Storage: -25 to 85°C (-13 to 185°F)	5 to 95% RH, non-condensing • Altitude Operating: 10,000 feet pressure altitude, maximum • Non-operating: 40,000 feet pressure altitude, maximum

3.1.2. Physical

Table 3-3 Pump Height and Weight Specifications

Height	Width	Depth	Weight
10.00 inches (254 mm)	2.55 inches (64.77 mm)	4.75 inches (181 mm)	5.07 pounds (2.30 Kg)

3.1.3. Power

Table 3-4 Current and Power Specifications

Voltage	20 to 30 VDC, 24VDC nominal
Current (at 24VDC)	**Idle, syringe on:** 0.45 to 0.55 Adc **Idle, syringe off:** 0.08 Adc **Valve move:** 1.5 typ, 1.8 max Adc **Initial surge:** 3.6 to 4 A peak, 6.5 msec **Syringe move:** 1.0 typ, 1.3 max Adc **Initial surge:** 3.3 A peak, 4msec **Turn-on surge:** 2.5 A peak, 10 msec
Power consumption	**Idle, syringe off:** 2 Watts, max. **Idle, syringe on:** 13 Watts, max. **Syringe or valve in motion:** 44 Watts, max.

3.1.4. Communications

The VersaPump 6 syringe drive has two serial communications protocols: OEM and DT.

Table 3-5 Communications Parameters

Baud Rate	1200, 2400, 4800, 9600, 19200, 38400 baud
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None
Physical Protocols	RS-485, RS-232
Logical Protocols	DT, OEM

3.1.5. I/O Interface

The following subsections describe the User Inputs/Outputs (User I/O) available on the V6 Syringe Drive Unit.

- 3 parallel digital outputs
- 3 parallel digital inputs
- 1 digital voltmeter input
- 1 serial I/O expansion port for adding additional I/O externally

Table 3-6 Digital Output Specifications

DC or Peak Load Current	170 mA maximum per output (4 watt load at 24VDC)
Output Resistance	5 ohm typical
Output Voltage	45VDC maximum (external load supply voltage)
Output Leakage (off)	Io < 100 uA

Table 3-7 Digital Input Specifications

Logic Compatibility	TTL, 5V CMOS
Input Voltage, Maximum	100 V peak for 8.3 msec maximum +30 VDC to =25 VDC, continuous, maximum
Logic "True" Level	<1.0 V
Logic "False" Level	>3.5 V or open

Table 3-8 Voltmeter Input Specifications

Input impedance	1 Megohm DC, 20 Kohm into 0.1 :F AC
Resolution	8 bits (0 - 255), 20 mV/LSB
Accuracy	+/- 20 mV (1 LSB)
Conversion Time	18 :sec
Range	5.10 V full scale
Input Filter	80 Hz lowpass, -6 dB/octave

Table 3-9 Expansion Port Specifications

Inputs	Logic "0"	+3.5 V to +5.0 V
	Logic "1"	-0.3 V to +1 V
	Input Current	<10 µA
Outputs	Logic "0"	<0.4 V as 1.6 mA sink
	Logic "1"	>4.2 V at 0.8 mA source
	+5V Output	100mA dc to load, maximum
Clock	Quiescent Level	Logic "0"
	Active Edges	Positive-going
	Frequency	115.2 KHz
Data Strobe	Quiescent Level	Logic "0"
	Data Hold Time	0 seconds
Data Transfer Cycle Time (strobe pulse width)	1-byte Mode	93± 4 µsec
	2-byte Mode	187 to 262 µsec

3.1.6. User Program Memory

Table 3–10 Program Memory Specifications

Communication Buffer/RAM	390 bytes
Non-volatile Program Memory	400+ bytes standard, 8000+ expanded
Non-volatile Program Retention	15 years, minimum
User Program Capacity	10 programs in standard NVM 99 programs with expanded NVM

3.2. Drivers

3.2.1. Syringe Driver

The syringe driver is designed to drive a syringe having a full-stroke length of 6 cm at thrust forces up to 150 pounds force (68 Kgf), depending on speed (steps/second). The driver is available in three resolutions: 12000 steps, 24000 steps, and 48000 steps for the same 6 cm stroke length.

Accuracy

Accuracy is described by two parameters:

- Accuracy measures how closely a dispensed amount of fluid corresponds to the ideal programmed value.
- Precision describes the ability of the drive to deliver the same quantity of fluid for the same size programmed dispenses.

For both, the value given is expressed as a percentage of the full stroke of the syringe.

Table 3 11 Accuracy and Precision Specifications

Accuracy	Precision
0.20% CV (typical, full-stroke), 0.4% max	0.06% CV (typical, full-stroke), 0.12% max

Additional factors contributing to system accuracy are the total syringe size, any air bubbles or gaps, and any elasticity in the fluid path.

The syringe tolerance is a maximum ±1% of total volume. This error contribution is proportional to the amount dispensed as a fraction of syringe volume. Air bubbles, gaps, and tubing elasticity can contribute errors due to compressibility or expansion of their volumes. Such errors are proportional to the positive or negative fluid pressures in the fluid path.

For small dispensed volumes, the accuracy of the volume can be sensitive to the means by which the volume is removed from the probe or tubing tip. Any meniscus can contribute several microliters of dispense error. To minimize these errors, submerge the tip into the destination fluid or "touch off" the tip against the container.

Speed

Syringe speeds are measured in steps per second. The definition of a step is one increment of motion in either the aspirate or dispense direction.

Table 3–12 Syringe Speed Specifications

Normal Range*	Extended Range**
60 to 10000 steps per second	Less than 1 step per minute

*Norgren Kloehn minimum recommended lowest speed setting, 60 steps per second

**To achieve extended range performance, pump must use step-and-hold command sequence.

Syringe Thrust and Pressure

Syringe thrust is related to syringe fluid pressure by this relation:

psi = 38.7 x (Thrust - Friction) / Volume

Where:

- Volume is the total rated syringe volume in milliliters (cc).
- Thrust is the drive force in pounds.
- Friction is the syringe piston friction force in pounds. Syringe friction is larger for the larger syringes. The largest is the 50 mL, with 8-15 pounds friction, depending on syringe tip material. The smallest syringes have 3-7 pounds friction, depending on tip material.

The 48000-step model has a typical maximum thrust of about 50 pounds or more at any speed. For the 24000-step model, as the syringe speed is increased, the available syringe thrust can decrease substantially, as shown:

Table 3–13 Thrust Specifications for a 24K Model

Speed (sps)	Thrust (pounds)
<1500	>100
<7000	>50
8000	44
9000	22
10000	15

The pump will stall if the necessary syringe force to drive fluid exceeds the pump capabilities.

3.2.2. Valve Driver

The valve driver controls a user-selected valve mounted to the faceplate of the pump. The valve driver is configurable for different types of valves. This allows the VersaPump 6 to accommodate any of the available valve types without modification. Distribution and non-distribution types are available with from two to twelve ports.

3.3. Options

The VersaPump 6 series is available in 48000, 24000, or 12000 step-resolutions per 6 cm stroke, with or without electronic controllers and motor drives. Versions without controllers and motor drives include an interface board with the electromechanical interfaces, signal conditioning, and a convenient card edge connector.

A starter kit is also available from Norgren Kloehn Inc. The kit includes a power supply, cables, manuals, and software.

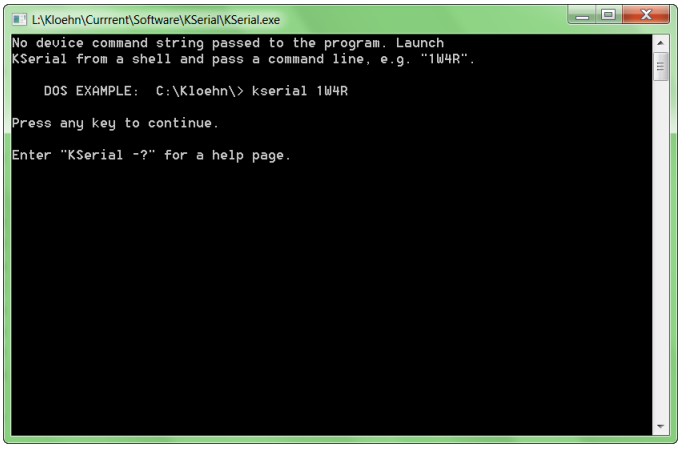
3.4. Supporting Software

All Norgren Kloehn supporting software is available by request at no charge on a CD. The CD also contains manuals and application notes for Norgren Kloehn pumps and valve drives. In the following subsections, the terms "Kloehn products" and "device" refer to all Norgren Kloehn pumps and valve drives that have integrated microprocessor control.

3.4.1. KSerial

KSerial is a command-driven program that operates through the MS DOS command line prompt and is used to communicate between Norgren Kloehn products and a PC. KSerial handles the communications overhead for both the DT and OEM protocols while providing error checking and response interpretations. The program allows the user to customize protocol, format, and baud rate. KSerial runs from a MS DOS or command prompt window, Windows® 98, ME, 2000, XP, and Windows 7, and uses the same command structure as HyperTerminal. Additional information about using KSerial is available in the included KSERIALManual.txt file.

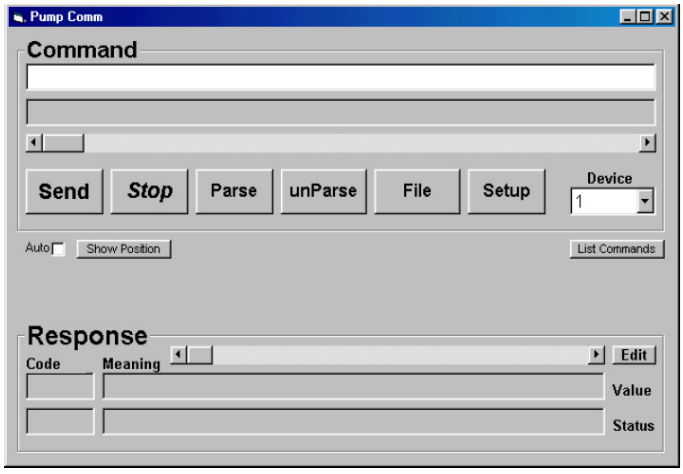
Figure 3-2 KSerial Screen



3.4.2. Kcom

Kcom functions as an intelligent terminal program customized for control and programming of Norgren Kloehn products. It runs in a Windows environment under Windows 95, 98, ME, 2000, XP, and Windows 7. A command input box with full editing capability is used to create command strings to send to a device. A response window displays responses from the device with interpretations. The user can type commands directly into the command window or select them from the drop-down list with prompts for command parameters. The user can also store and recall program strings from the PC memory for downloading to a device. Additional information about using Kcom is available in the included manual and readme files.

Figure 3-3 KComm Screen



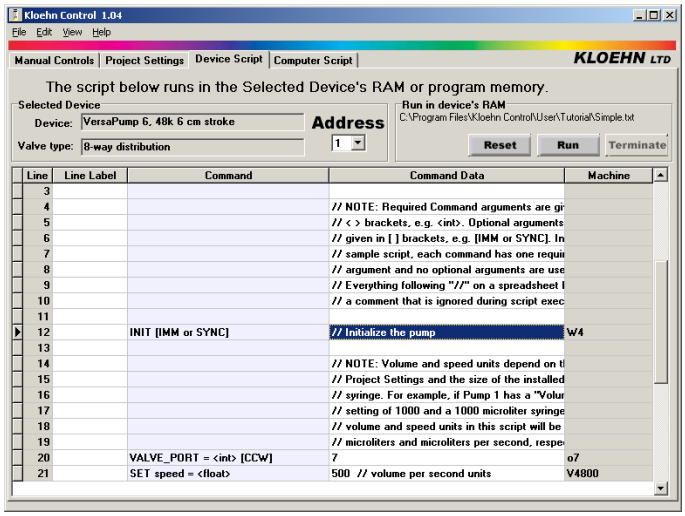
3.4.3. Kloehn Control

Kloehn Control is a program used to automate the control of microprocessor-controlled pumps and valves. It provides graphical manual controls and tools for writing, debugging, and running pump and valve control scripts. Scripts are control-programming strings that can run in a device or on a computer that controls a pump. Kloehn Control scripting is especially useful when writing programs that are lengthy, parameterized, or frequently modified such as an assay method. They are also useful for computer automations of pumps and valves without investing the time to learn programming or write software. Additional information about using Kloehn Control is available online, accessible from the Help menu bar.

Kloehn Control scripts use simple English commands such as ASPIRATE and VALVE_ PORT that can run on different devices. Using one set of script commands simplifies automation by bridging the differences of firmware, resolution, microprocessor, and computer. Scripting is menu-driven so that it knows the command set before writing the script. Error messages point out errors in syntax, configuration, and parameter values. The user can test scripts in simulation mode without hardware to catch many run-time errors quickly.

Kloehn Control also provides standard Integrated Development Environment (IDE) tools: file handling, editing, compiling, simulation, and watch windows for variables and device parameters.

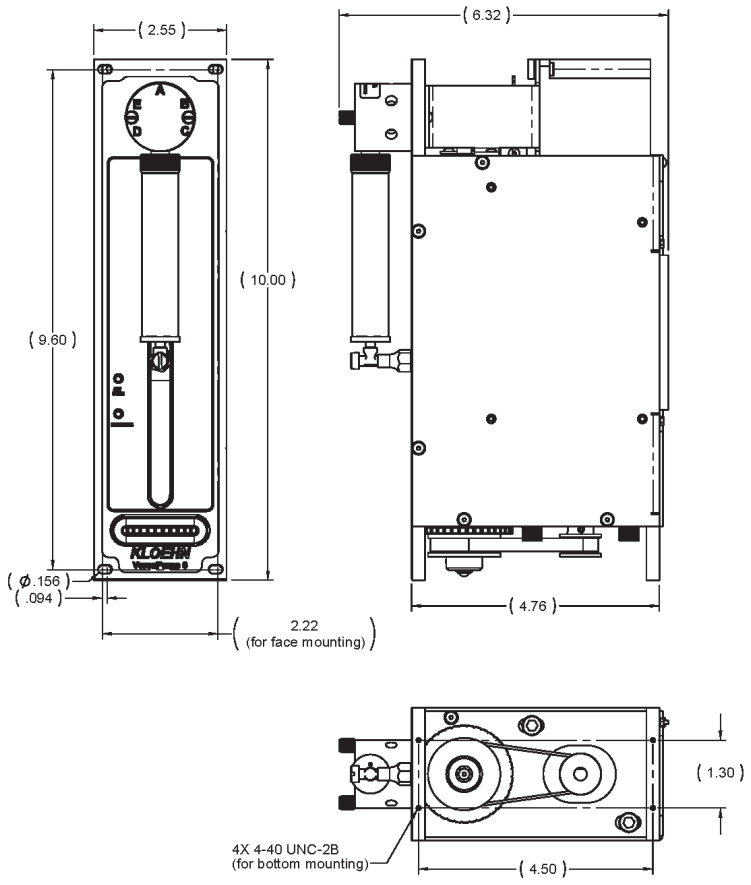
Figure 3-4 Kloehn Control Screen



3.5. Mounting

The mounting dimensions of the VersaPump 6 are shown in Figure 3–5. The drive is usually base-mounted using the holes in the bottom of the front and rear mounting feet. Alternately, the pump can be face-mounted.

Figure 3-5 Mounting Dimensions for the VersaPump 6



3.5.1. Mounting Surface Requirements

It is recommended that the pump be mounted to a solid base. If the pump is mounted to an instrument panel, reinforce the panel to create a very stiff, rigid surface.

NOTE: If these precautions are not observed, vibrations from operating the drive may result in the instrument face acting as a loud speaker, amplifying all pump acoustics.

Where possible, use vibration isolation material between the pump and the mounting surface to improve acoustic isolation from the mounting structure. A material such as Sorbothane (see http://www.sorbothane.com) is recommended. A gasket cut from a mouse pad can serve in a breadboard as a convenient isolator. When selecting a dampening material remember that different materials are acoustically transparent at some frequencies and acoustically opaque at other frequencies.

3.5.2. Faceplate Mounting

Faceplate mounting uses four faceplate mounting holes, one in each corner of the faceplate. Each hole is elongated and can accommodate a screw with a diameter as large as 0.156 in (0.396 cm) with a horizontal travel of up to 0.094 in (0.238 cm) from the center of the hole.

3.5.3. Base Mounting

Base mounting uses the holes in the bottom edges of the faceplate and support plate. These holes are four 4 40 UNC 2B tapped holes shown in the bottom view of the pump in Figure 3 5. The pump is mounted with four 4 40 screws passing up through the mounting surface and into the corresponding mounting holes, front and rear. This method is useful for light enclosures that are easily turned over for access to the screws.

3.5.4. Non-Standard Mounting Positions

- **Inverted mounting** — Some situations can benefit from mounting the pump inverted, or upside-down, with the valve at the bottom.
- **Sideways mounting** — The pump can also be mounted on its side in a horizontal position. The syringe can be positioned up, down, or facing to one side.

3.5.5. Instrument Enclosures

An instrument enclosure to which the drive is mounted should have good electrical conductivity to the system chassis ground. This reduces radiated emissions from the equipment. A wire from the pump chassis to the equipment chassis does not provide a satisfactory system ground because it does not provide the high-frequency transient conductivity required. If possible, a metallic enclosure or a plastic enclosure with RF shielding is preferred.

3.5.6. Air Flow and Ventilation Considerations

This pump is designed with a large operating ambient temperature margin. This margin depends on a good natural convection air flow across the driver side plate. The power devices on the driver board, on the right side as viewed from the rear, use the side plate as a heat radiator for cooling.

NOTE: If air flow is inhibited, the driver board may overheat and fail prematurely.

Adequate air venting for an enclosure is required for system reliability. In most applications, a large cooling air inlet at the bottom of an enclosure and an adequate hot air vent near the top can provide adequate ventilation.

If a good natural convection air flow cannot be assured, it may be necessary to place a small fan at the lower rear part of the pump. In general, a 40 mm x 40 mm, 24 V, brushless DC motor fan works well. Unloaded flow ratings of 8 cubic meters per hour or higher are sufficient. Place a fan so that it faces the motor at the bottom of the pump. This causes good air flow between the driver board and the side plate. Cool air flow will enter from the bottom and heated air will exit at the top of the pump.

When measuring the internal temperatures, a temperature probe placed into the center of the pump cavity is generally not adequate. The probe should measure the temperature on the inside of the left side plate near the front of the pump.

NOTE: Do not exceed 55°C (131°F).

3.6. Power Supply

All electrical interfaces are located on the puns of the rear card-edge, P1 as show in Figure 3–6. The mating connector has two rows with 18 pins-per-row, and a 36-pin card edge connector. A 300 Series connector can be obtained from EDAC, Inc. (http://www.edac.net).

This section describes the selection of power supplies, best wiring practices, and low-voltage condition detection.

3.6.1. Connection

Power for the pump is input on pins 33 though 36 of the card edge connector. The positive power lead is duplicated on pins 35 and 36. The power ground is duplicated on pins 33 and 34. This permits two solid, straight wires to be passed though the edge connector in-line for a multi-pump system with plug-in pump modules.

Low Voltage Warning

When the pump supply voltage drops below an internal reference minimum (20 V), a "Low Voltage" error message is generated. This message is generated only once for each drop in voltage. Continued low voltage error messages are generated only if the voltage rises above the reference minimum and then drops below it again. If continuous low voltage error messages occur, the most likely cause is either low-frequency noise or ripples in the power supply.

If the voltage remains below minimum, valve and syringe movements are inhibited. However, unless the supply voltage drops below 8 VDC, the internal control electronics and memory are not affected and all other instructions, such as I/O operations and queries, will still operate normally after the low voltage error message has been reported or cleared.

Some power supplies will turn on gradually. If the rise time of the supply is slow, the internal computer may report a "Low Voltage" error when the pump is initially queried. This does not cause any operation problems after the message is reported.

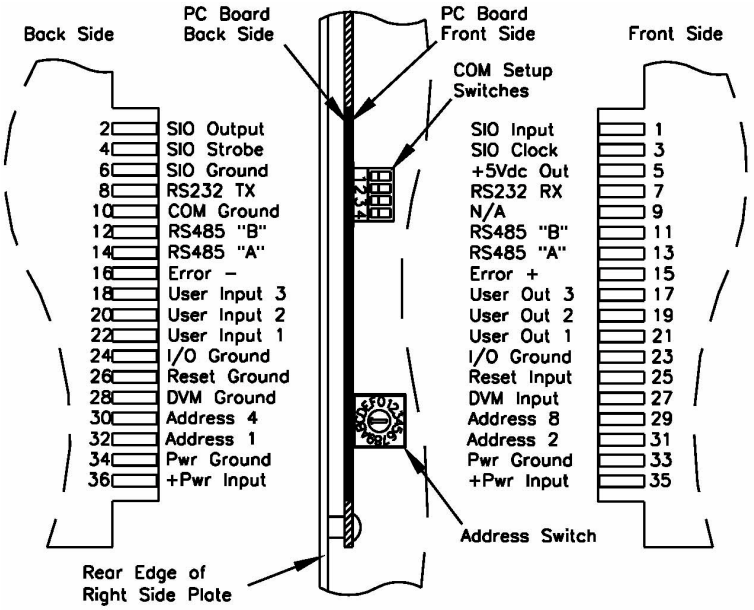
3.6.2. Capacity Selection

Make sure that the power supply capacity is consistent with the specifications in Power, section 3.1.3. A single P/N 17732 power supply has sufficient capacity to power one VersaPump 6 device.

Powering a Single Pump

An output capacity of 45 watts at 24VDC is considered a practical value for a one-pump power supply for normal pump operation. The normal idle power consumption is about 13 watts. The 45 watt rating allows for the maximum power required during syringe and valve moves, with a small reserve. The AC transient currents during syringe and valve moves are negligible.

Figure 3–6 Drive Interface



Powering Multiple Pumps on a Single Supply

For multiple pumps on one supply, the overall system operation should be considered. If there are N pumps, of which only M units will be making a syringe or valve move at the same time, then the average power capacity of the supply should be at least P = 45M + 13(N-M) watts. The pump automatically turns on the valve motor for moves and then turns it off when not moving.

See the *Pump Programming Tips*, section 4.3, for additional multiple-pump system wiring considerations.

At power-up, the valve motor in every pump connected to the power supply will make simultaneous valve calibration moves, demanding about 1.8A each. To allow for the valve calibration moves, the power supply should have a maximum average power capacity of P = 45N watts. If this results in an excessively large power rating for the normal system operation then the power-up initialize inhibit parameter can be set to prevent the power up moves. Make sure that there is always sufficient capacity to handle the worst-case simultaneous moves.

The in-rush current at initial power-up is the idle current plus the current required to charge the nominal 470 uF capacitance at each pump power input. For almost any power supply, this current will be well within its capacity.

Power Conservation for Battery Applications

The V6 pump draws a current that depends upon the power supply voltage, the idle logic current, and the motor currents. The current consumption can be minimized in some applications through the pump programming.

The supply voltage is inversely proportional to the current consumption. This is because the pump is a constant power device. As the voltage increases, the current decreases so the product of the two will remain approximately constant. The capacity of a battery for portable operation is best estimated using a watt-hour rating (watt-hours = ampere-hours x volts).

The motor current is the sum of the valve motor and syringe motor demands. The valve motor draws a current during a valve move, and automatically turns off when a valve move ends. The valve motor automatically turns on at the start of a valve move.

The syringe motor normally idles at half-power. When a syringe move begins, the syringe switches to full-power operation for the duration of the move. When the move ends, the power automatically switches back to half-power. If the syringe is not required to hold a significant back-pressure, the syringe motor can be turned off at the completion of each move, thereby reducing idle power to the logic idle power alone. The logic idle power alone is typically about 2 Watts.

3.6.3. Power Supply Types

There are four general types of power supply:

- A switching power supply is the preferred choice. It offers higher efficiency, lower heat generation, and a well-filtered output. Some switching power supplies have a minimum load current requirement. Since the pump can idle as low as 70 milliamps, the supply should be rated for a minimum load current equal to the minimum total system idle current. Add a ballast resistor across the supply output to guarantee the minimum load requirement of the supply.
- The unregulated supply is the cheapest and simplest. Its output voltage varies about 5% to 20% from no-load to rated load. In addition, the output varies in proportion to the input line voltage. Since the driver is specified for 24VDC ± 10%, it is not recommended for use with the V6 pumps.
- The linear regulator supply usually has a current limiting feature which must be set high enough to handle any current transients generated by the syringe drive.

NOTE: If the current limit is too low, erratic pump operation will result with no obvious cause.

This condition can be detected by monitoring the power with a storage oscilloscope. Another possible consequence of low current limit is a too-slow power rise at turn-on. Linear power supplies are inefficient, requiring larger power input, more space, and more weight than the switching power supplies.

- Battery operation from a 24 V battery system is feasible due to the pump's wide operating range. A standby battery voltage of 28 VDC is usually seen in automotive and aircraft systems. This is acceptable for normal operation. Mobile systems should provide over-voltage clamping for transient exceeding 34 VDC.

3.6.4. Connecting the Power without the Starter Kit

If the Card Edge Adapter Board (adapter P/N 23428) is not used, the Card Edge Connector (P/N 23277) or its equivalent is required. The connector is a 36-pin card edge connector that has 0.156-inch pin centers.

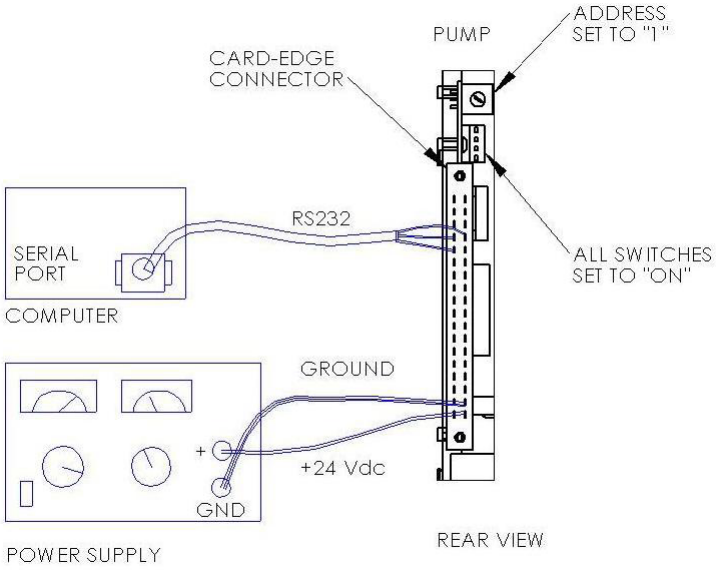
Insert the card connector onto the card edge at the rear of the pump. Note the power pins are at the bottom edge of the connector, as indicated in Figure 3–7.

CAUTION: Make sure that the card edge connector is oriented with the power pins at the bottom of the connector as shown in Figure 2–7 and Figure 3–7.

Figure 3–7 shows the connections for DC power and communications with a PC. The PC serial port connects to the RS-232 pins.

The RS-232 cable wiring is illustrated in Figure 2–4 for both DB-25 and DB-9 connectors. For connecting more than one pump, see *Connecting Multiple Devices*, section 3.9.5, for communications wiring and *System Wiring Practices*, section 3.7.1, for power distribution wiring.

Figure 3–7 Pump Connections with Card Edge Connector



3.7. Wiring and Connectivity

3.7.1. Wiring and System Reliability

In a system with two or more syringe pumps, the power distribution wiring can affect the system reliability. The best system wiring practice is to connect each drive module with an individual pair of power leads from the power supply to that individual module, as shown in Figure 3–14 in section 3.9.5. The pair of leads for each module should be twisted together along their length to reduce radiated fields. Route the twisted pairs of power leads close to the chassis if a metal chassis is used for mounting. This aids in reducing unwanted stray electromagnetic fields in the overall equipment design. For applications that use a backplane to wire multiple pumps, an alternative power distribution wiring configuration is shown in Figure 3–8 in section 3.7.1. Wire gauges should be selected for the total DC current.

The J1 interface connector has two power input pins and two ground pins. Good wiring practice uses both opposing pins for the positive lead and both opposing pins for the ground lead. Redundant pins ensure reliable power connections.

Communication Issues during Electrostatic Discharge (ESD) Testing

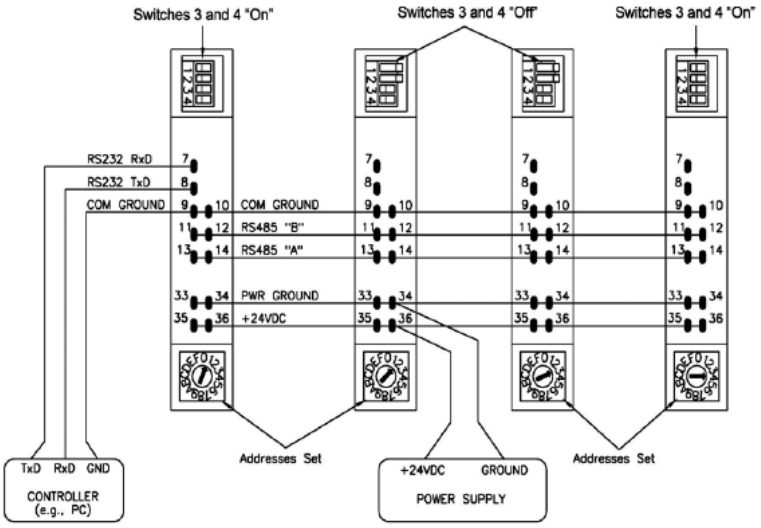
If communication upsets occur during ESD testing when high-voltage arcs are injected into either the controller or the syringe drive, insertion of a ferrite common-mode choke with good high-frequency impedance can eliminate the problem. Loop the power and communications lines two or three times through a ferrite toroid that has an outside diameter of one inch.

Improved EMC Performance

Improved EMC performance can often be realized by grounding the pump faceplate. Remove the black anodized surface coating from around a lower mounting hole with a wire brush and use a 20ga. or larger wire to connect the pump to a nearby ground plane.

Syringe drives should not share the same power leads as large, noisy electrical loads such as motors and large solenoids. While the syringe design contains filtering to reduce susceptibility to electrical noise conducted via the power supply wires, large current or voltage transients could be harmful to the pump.

Figure 3–8 Alternate Backplane Power Distribution Wiring



Address Inputs

The device address on the communication bus can be hard-wired into the connector so that a device can be inserted into an instrument without any need to set the address switch to a particular location. To use this feature, the address switch must be in the "**F**" position. If the address switch is in any other position, a conflict will result between a hard-wired address and the address indicated on the switch.

The address inputs have built-in pull-up resistors and use positive logic. The default address input level is logic "1". Logic "0" is made by grounding an address pin. The address is set as a 4-bit binary number by shorting those pins to ground which should have a zero value. The address weighting on the pins is as follows:

Address 8 = 8 Address 4 = 4 Address 2 = 2 Address 1 = 1

The address value is the sum of the pin weights that are not connected to ground. See Table 3–14 for the pin connections corresponding to the equivalent address switch settings.

Wiring an Address onto the Card Edge Connector

To wire an address onto the card edge connector, convert the address "**1**" through "**F**" into a binary representation of the hexadecimal number and ground the pins which should have a zero value. Do not use the "**0**" address as it is reserved for the PC host controller.

The pin connections are shown in the following table. A "Ground" indicates the pin should be connected to a ground pin. The notation "n/c" signifies there should be no connection to the pin.

NOTE: To use the hard-wired address, the address switch MUST be set to the "F" position. If it is not an address conflict will exist between the wiring and the switch. The address can be set with either the address switch or with the card edged connector wiring. Only one of the two methods may be used.

Table 3-14 Hard-Wired Addresses

Address	Address 8	Address 4	Address 2	Address 1
1	Ground	Ground	Ground	
2	Ground	Ground	n/c	Ground
3	Ground	Ground	n/c	n/c
4	Ground	n/c	Ground	Ground
5	Ground	n/c	Ground	n/c
6	Ground	n/c	n/c	Ground
7	Ground	n/c	n/c	n/c
8	n/c	Ground	Ground	Ground
9	n/c	Ground	Ground	n/c
A	n/c	Ground	n/c	Ground
B	n/c	Ground	n/c	n/c
C	n/c	n/c	Ground	Ground
D	n/c	n/c	Ground	n/c
E	n/c	n/c	n/c	Ground
F	n/c	n/c	n/c	n/c

3.7.2. Inputs

Digital Voltmeter Input

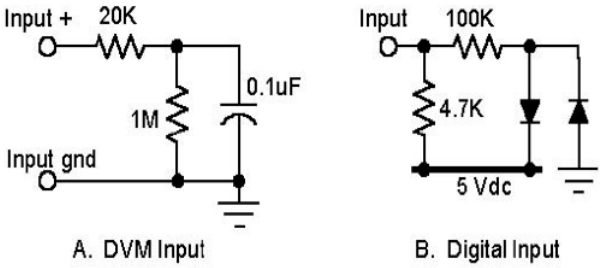
An 8-bit voltmeter (DVM) is built into the pump I/O. It is accessible on pins 27 and 28. Connect the analog signal to be read to pin 28.

The DVM Input allows 8-bit measurements of external analog voltages. Analog values may be reported to the host controller or used within a user program to compare a measured analog value to a user-preset value for a conditional program jump. Also, the DVM Input can be used to set program instruction parameter values, as described in *Indirect Variables*, section 5.5.3.

The input impedance is 1 Mohm for dc inputs, and is 20 Kohm in series with 0.1 uF capacitance to ground for high-frequency signals. This input impedance is due to the anti-aliasing filter. The anti-aliasing filter at the input, shown in Figure 3–9A. DVM Input half of the following figure has a – 3 dB cutoff frequency of 80 Hz. Although the conversion time for an input sample is approximately 18 microseconds, at least 11 milliseconds are required for the new value to settle to within the resolution (1/256 of full scale) of the DVM. If an abrupt step change occurs when the conversion begins, the input filter time constant ensures that the correct valve at the start of the conversion will be read to within the DVM accuracy.

The input voltage range is 0 to 5VDC, corresponding to an internal conversion integer value from 0 to 255, respectively. Restrict the input voltage range to 5V or less. Each increment of internal value (an LSB) corresponds to an analog input increment of 20 mV.

Figure 3-9 Equivalent Circuits of the User Inputs



Use this formula to calculate the internal numerical value for a DVM input:

N = Vin / 0.02

Where:

N = internal integer value

Vin = analog input voltage

To avoid noise and errors due to ground loops, twist the ground wire of the voltage source to be measured with the input wire (a "twisted pair") and connect it directly to the analog ground pin. When using shielded, twisted pair wire ground the shield at one end only.

Reset Input

A Reset Input is located on pin 25. When this is brought to a low level below 0.8 VDC, the processor is reset. The reset condition remains active while the input is low. When the input is returned to a higher level, the processor begins a pump initialization cycle after a 0.25 second delay. The reset input is referenced to ground pins 23 to 26. Reset is also automatically generated internally when power is first applied.

A reset causes the following actions:

- The syringe position value is set to zero. The position is no longer valid.
- The position snapshot values are reset to -1.
- Error messages are erased and the Error Output is turned **Off**.
- The valve moves to the home, or port A position, if enabled.
- Temporary memory (RAM) is cleared.
- The communications buffer is cleared.
- The pump address is read and saved.
- The Com Setup Switch status is read and saved.
- Syringe speeds are set to the values saved in the non-volatile memory.
- The User Outputs are reset to **Off**.

User Digital Inputs

Three protected digital User Inputs are provided on pins 18, 20, and 22. Each of these User Inputs can be queried at any time, including during pump operation and while an internal program is executing. These User Inputs are commonly used to control the pump operation.
As shown in the equivalent input circuit in the "B" half of Figure 3–9 in section 3.7.2, each input has a 4.7 Kohm pull-up resistor and is protected for input voltages up to 30 VDC. Inputs are compatible with CMOS and TTL logic operating from 5V supplies, with other pump's digital outputs, and with external switches. An **On** input is less than 1 V. An **Off** input is more that 3.5 V, or an open circuit. The internal resistance provides the bias required for external switches. External switches should make a connection to ground when in the **On** condition.

NOTE: Do NOT apply voltages greater than 30 Volts to a User Input. Doing so can damage the circuit board.

3.7.3. Outputs

Digital Outputs

Three digital outputs are provided on pins 17, 19, and 21. These outputs appear directly opposite the digital inputs described in *Setting the Switches*, section 2.5. Each output consists of an open collector N-channel MOSFET output rated to sink 170mA at 40VDC. The User Outputs can be controlled under the internal user program control or set them up with external commands from a controller. They are suitable for driving relays, solenoids, indicator lights, opto-isolators, or the logic inputs. In addition, these outputs can be "wire-OR'd" with an external pull-up resistance.

The digital outputs are active low:

- An **On** condition is a 5-ohm resistance to ground.
- An **Off** condition is an open-circuit. This is compatible with the digital inputs.

Error Outputs

Pin 16 provides an Error Out suitable for driving logic or an LED indicator. The output is activated whenever an error condition is detected within the pump. When an error condition occurs, the Error Out on pin 16 is set to an **On** condition. In the absence of an error or after an error has been reported to a controller via the communications I/O, the Error Out is an open circuit.

The Error Bias output on pin 15 consists of a 330 ohm resistor connected internally to the +5 VDC power. The output provides a current-limited output suitable for direct drive of an LED indicator anode. To drive an external LED error indicator, connect pin 15 to the anode and pin 16 to the cathode.

If an LED is not used, the Error Out can be used to drive some other error indicating device. The maximum output voltage in the off condition is 40 volts. If an external supply is used, a common ground connection should be taken from the I/O ground to the external supply ground. Since no internal protection is provided for inductive loads, if relays or solenoids are driven, a clamp diode across the load is required. For sending an error indication to remote electronic equipment, the Error Out pin can be used to drive the input to an opto-isolator.

The error outputs of several devices can be tied together to make a single wired or system error signal. Use this signal to drive an LED or an input to a controller.

5VDC Power Output

To power external I/O circuits, the card edge interface includes a +5 VDC power output on pin 5. This output is rated for loads up to 100 mA. Use any of the ground pins on the card edge in conjunction with this pin.

3.7.4. I/O Expansion (IOX)

The I/O Expansion Port (IOX) provides a means to expand the number of external inputs and outputs for the pump. An I/O Expander board adds 8 additional inputs and 8 additional outputs. Each of the expander outputs can sink up to 250 mA at voltages to 40 VDC.

The IOX has one 6-pin SPI (Serial Peripheral Interface) synchronous serial interface which simultaneously shifts output data out to the expansion port (MOSI pin) while shifting input data in from the expansion port (MISO pin), using the CLK pin as the shift clock. The strobe output (NSS) is active low during the data I/O operation. The connector has +5V and ground for powering external I/O circuits.

The IOX expansion port is independent of the serial communications port and is located on pins 1 through 6 of the edge connector. The OIX uses four signals:

- IOX Input (MISO) — Receives serial data as 8-bit bytes from an external circuit.
- IOX Output (MOSI) — Sends data as 8-bit bytes from the pump.
- IOX Clock (CLK) — Used by the IOX port to synchronize the serial data transfer bit-by-bit.
- IOX Strobe (NSS) — Acts as a synchronous, active-low enable signal for external devices.

Depending on the value of the IOX mode parameter "**~S**", IOX operation will perform one-byte or two-byte data transfer.

Figure 3–10 Serial Expansion I/O Timing, 2-byte Mode

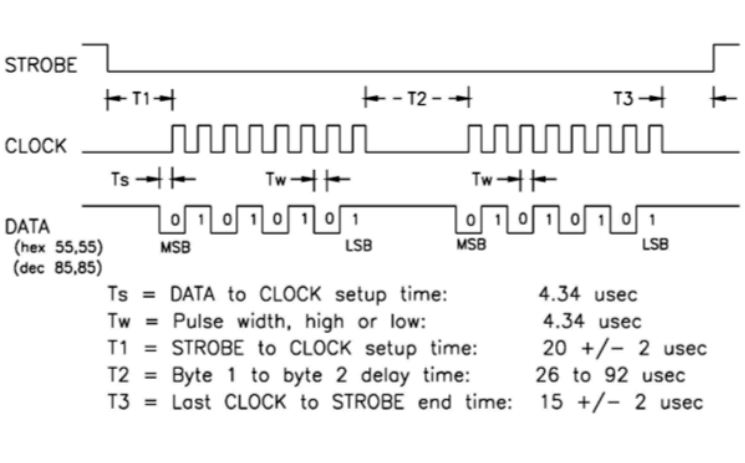
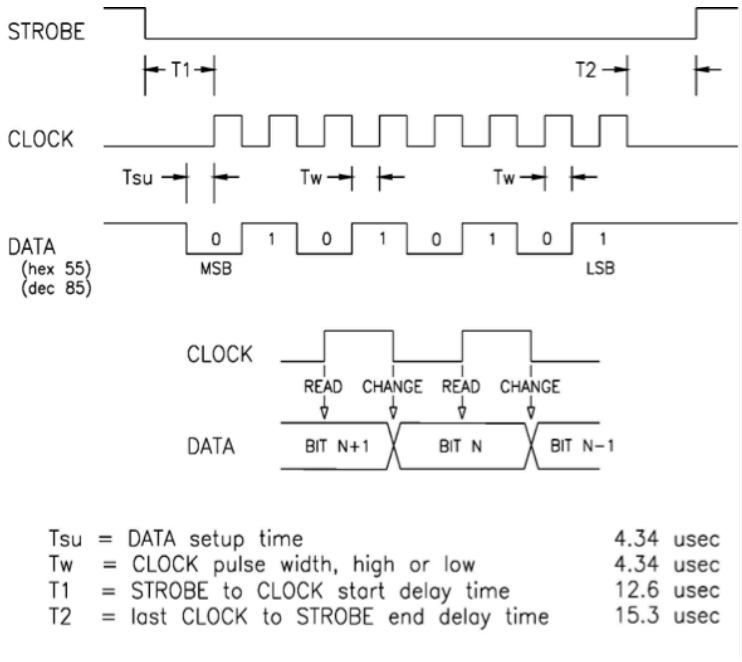


Figure 3–11 Serial Expansion I/O Timing, Single-byte Mode



Tsu	=	DATA setup time	4.34 usec
Tw	=	CLOCK pulse width, high or low	4.34 usec
T1	=	STROBE to CLOCK start delay time	12.6 usec
T2	=	last CLOCK to STROBE end delay time	15.3 usec

3.7.5. Communications I/O

The pump provides serial communications compatible with PC serial ports and RS-485 I/O cards. All communications use ASCII characters for both commands and responses. Numbers are expressed as integer decimal numbers in ASCII characters. The two communications protocols are DT and OEM. These protocols are explained in *Communication Protocols*, section 3.9.3, and *Communication Settings*, section 3.9.4, for the communications physical protocol specifications.

RS-485 Communications I/O

The RS-485 I/O is available on pins 11 through 14. There are two signals: RS-485 A and RS-485 B. The A line is the positive line and the B line is the negative line under idle bias conditions. To prevent common-mode voltage errors, the communications should also use the Com ground on pin 10 for an RS-485 communication ground in addition to the A and B lines.

CAUTION: Common mode voltage can be potentially hazardous when wires are run between buildings or floors within a building where the AC power distribution phase legs are not wired correctly. Before connecting ground or signal wires verify the voltage potential between the two points is below 7VDC by checking with a voltmeter.

The A signal is duplicated on pins 13 and 14, while the B signal is duplicated on pins 11 and 12. This duplication permits a straight wire to pass through each pair of the A and B lines to interconnect a series of devices.

The RS-485 bus requires a proper bias and termination network for reliable operation. The necessary network is included in the pump and is applied via the RS-485 bias toggles 3 and 4 on the Com Setup Switch shown in Figure 3 12. The first and last devices on an RS-485 bus should have the network switched to **On**. All other devices between the first and last devices should have the network switched to **Off**. A toggle is **On** when the button is positioned to the center of the switch housing; a toggle is **Off** when the button is positioned nearest the edge of the switch housing.

CAUTION: Do NOT have more than two RS-485 terminations switched on regardless of the number of devices on the bus. Use ONLY one termination on at each end of the overall bus wiring.

The com bus wiring should be a twisted pair for A and B signals. The rate of twists should be approximately one to three turns per inch. The ground wire may be a shield or a third wire twisted with A and B wires.

NOTE: An RS-485 bus with multiple devices must be wired directly from device to device using A, B, and Com ground pins.

RS-232 Communication I/O

An RS-232 communications option is available on the card edge connector. This provides a Communications I/O compatible with the serial ports found on PCs and controllers. RS-232 pins operate within a voltage range of +12VDC to -12VDC and are not compatible with RS-485 or TTL signals. RS-232 input signals should never be connected to ground (the Com line).

There are two lines:

- RS-232 RX on pin 7 — Receives data from a controller to the pump
- RS-232 TX on pin 8 — Sends data from the pump to a controller

Use the Com ground connection on pin 10 to connect a communications ground line to the controller.

The RS-232 protocol uses no flow control. Therefore, no signals other than RS-232 RXD, TXD, and Com ground are needed for serial communications.

3.7.6. Rear Panel Switched

Com Setup Switch

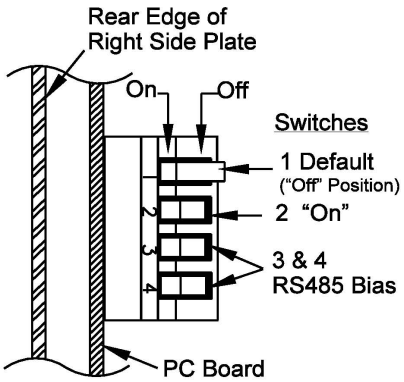
The Com Setup Switch is shown in Figure 3–12. This switch has four toggles (buttons) numbered 1 through 4. These toggles control whether the RS-485 bias network is attached to the RS-485 bus and whether the communications default for baud rate protocol are set to the factory default values or to the values set in the user configuration variables.

NOTE: The default toggle (number 1) must be set to Off in order to enable programs to auto start.

NOTE: Either turn On or Off both toggles 3 and 4 in order to connect the RS-485 termination network. Do not set RS-485 bias toggles to On.

NOTE: The default toggle (number 1) must be set to Off in order to permit the baud rate and protocol to change from the factory default settings.

Figure 3–12 Com Setup Switch

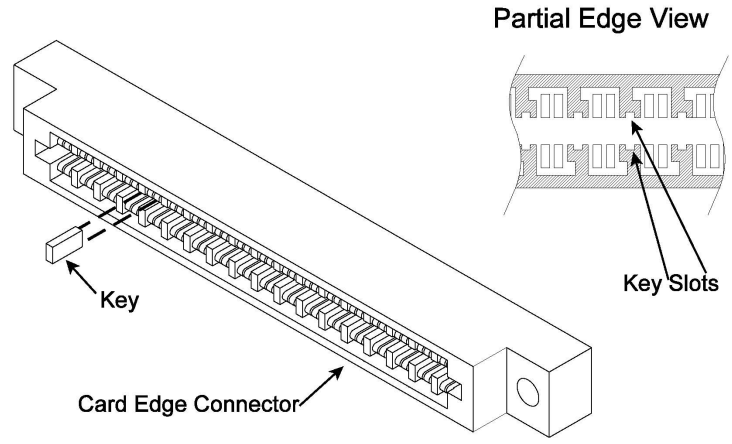


Address Switch

The address switch, shown in Figure 3–6, sets the communications address of the pump on the RS-485 bus. There are 15 legal pump addresses on switches "**1**" through "**F**". Address "**0**" is reserved for a controller address and cannot be used for a pump address.

If the address switch is set to "**F**", the pump address can be set by wiring the address pins on the card edge connector, as explained in *Outputs*, section 3.7.3. If the switch is in any other position, the wired address will conflict with the switch address.

Figure 3 13 Connector Key Slot



3.7.7. Connector Key

The card edge connector is supplied with a key. This key is a plastic insert in the connector which corresponds to the slot in the card edge as depicted in Figure 3–13 above.

Ensure the key is in the proper position in the connector to match the slot in the card edge. Move the key by grasping it with a pair of needle-nose pliers and pulling it out of the connector. Insert the key into the connector by pressing the key into the detents which are located between each opposing pair of connector contacts.

3.8. Use of Commands

The following section contains a general description of the VersaPump's use of commands to enable functionality. For more detailed information on using commands, see sections 4, 5, 6, and 7.

3.8.1. General Command Structure

A command is an instruction to the pump to perform a single action such as move the syringe or turn the valve. Multiple commands can be combined to form command strings. Command strings, also called programs, can perform complex tasks consisting of many operations, including decision making.

A command consists of ASCII characters and contains two parts:

- **The command** — A case-sensitive letter which represents a specific type of action to perform.
- **Its argument** — Follows the command letter and determines how the command will execute. For example, the command "**D1200**" tells the pump to dispense (D) 1200 steps.

Commands that make decisions have two arguments. The first is a number which works the same way as for other commands. The second is a letter that determines what the outcome of the decision will be depending on the circumstances. For example, for command "**i2F**", the "**2**" is for a low level. If the level is low, the program goes to the label "**F**" (a label is a place marker in a program).

3.8.2. Command Addressing

All commands and command strings must begin with a device address. The device address determines which devices will respond to a particular command string. In this way, many devices can be connected together on a single communications line without interfering with each other. The character which signifies an address is the forward slash (/). When the forward slash is seen by a pump, the pump reads the character which follows as an address to determine if that pump should accept the command string. The individual device address is set via the address switch or by the address pin wiring on the card edge connector.

For example, if the address switch is set to "**3**", a command string which begins with **/3** will be accepted by the pump. If the string were to begin with "**/2**", the pump would ignore the string.

Pumps may be addressed individually or in groups. Groups can be in pairs, groups of four, or all pumps on a single communications line. The details of pump addressing are given in *Individual Device Addressing*, section 3.9.1, and *Multiple Device Addressing*, section 3.9.2.

3.8.3. Pump Replies

When the pump receives a command string, it checks the string for correctness and sends a reply. The reply always begins with "**/0**" which is the address of the PC or controlling device. At least one character follows immediately after the "**/0**". This character is the status byte. The status byte informs the controller of the current status of the communication and the pump.

Status Byte Types

The two status types are "OK" and "Error". There is a unique letter assigned to each type of error the pump can recognize. For every error, the status letter can be upper or lower case. If the status byte is capitalized, the pump is busy doing something. If the status byte is lower case, the pump is not busy, and is ready for another command.

The OK status has two special characters to indicate "busy "or "ready". The accent grave mark (`) indicates a ready status and the ampersand (@) indicates a busy status. A typical response is "**/0`**" or "**/0@**". These responses indicate the pump and the command string are OK.

Queries

Most command strings cannot be accepted until the previous command string is completed. Queries are the exception to this rule. A query asks the pump to report information, not perform an action. Enter a query any time and it will be answered when it is received, even if the pump is busy.

3.9. Communications

The RS-485 bus supports up to 15 devices, plus a PC host controller. Each device can be addressed individually, in pairs, in groups of four, or all at once. A response from a device only occurs for individual addressing. In the multiple device addressing modes, no device provides a status response. Status messages are saved until an individual device is addressed.

Commands always originate from the PC host controller at address "**0**". Commands are never sent to the host.

3.9.1. Individual Device Addressing

In Table 3–15 , the Switch Setting column is the number to which the Address Switch is set on the pump. The ASCII Char column refers to the ASCII character (also the keyboard character) that corresponds to the address switch setting. Devices addressed in this mode respond with a status byte and an answer to a query.

Table 3–15 Individual Device Addressing

Setting Switch	Hex Address	ASCII Char	Switch Setting	Hex Address	ASCII Char
0	Reserved for controller	Reserved for controller	8	38	8
1	31	1	9	39	9
2	32	2	A	3A	:
3	33	3	B	3B	;
4	34	4	C	3C	<
5	35	5	D	3D	=
6	36	6	E	3E	>
7	37	7	F	3F	?

3.9.2. Multiple Device Addressing

Multiple pump addressing sends a command string to more than one pump on the communications bus at the same time to prevent bus conflicts. Pumps do not provide a response to a command in one of these multiple-pump addressing modes.

Dual Device Mode

In the dual device addressing mode, a group of two pumps is addressed. In this mode, individual devices do not provide status responses to commands.

Pump Group	1, 2	3, 4	5, 6	7, 8	9 , A	B , C	D, E
Hex Address	41	43	45	47	49	4B	4D
ASCII Character	A	C	E	G	I	K	M

Quad Device Mode

In the quad device addressing mode, a group of four pumps is addressed. In this mode individual pumps do not provide status responses to commands.

Pump Group	1,2,3,4	5,6,7,8	9, A, B, C	D, E, F
Hex Address	51	55	59	5D
ASCII Character	Q	U	Y	]

Global Mode

In the global device addressing mode, all devices on the bus are simultaneously addressed. In this mode individual devices do not provide status responses to commands. The address character is the underscore, "_" or hexadecimal 5F.

3.9.3. Communication Protocols

The communications software protocols are the command and response format used to send commands and receive responses from pumps. There are two protocols:

- **DT (Data Terminal)** — The DT protocol is a simple data terminal protocol that is compatible with nearly all terminal emulation programs and basic communications drivers. This is the preferred protocol in most situations
- **OEM (Original Equipment Manufacturer)** — The OEM protocol provides explicit error checking and a repeated-command sequencing algorithm. These features are not implemented in any standard terminal programs. Kloehn offers software which can communicate using this protocol. The KSerial driver can be called from within a user program and handles the communications overhead.

DT Command Protocol

This section describes the command package of the DT protocol. A command packet is a sequence of bytes sent by a host computer from the host to a device.

Table 3-16 Packet Description for DT Command Protocol

Byte #	Description	ASCII	Hex
1	Start Character	/	2F
2	Address Character	See *Individual* and *Multiple Device Addressing*, sections 3.9.1 and 3.9.2.	
3 to N	Command Characters	See *Commands*, *section 5*.	
N+1	End (Carriage Return)	<CR>	0D

Explanation of bytes:

Byte 1: The starting character signals the beginning of the new packet. It is the front slash character (/) on the computer keyboard, 2F hex.

Byte 2: The device address is an address number for a device for a group of devices. It can address a total of 15 devices in the network mode.

Byte 3: The command or a sequence of commands starts with byte 3. A command or a command sequence with length n bytes uses byte 3 to byte 3+n-1.

Byte 3+N: The ending character indicates the end of a packet. It is 0D hex, the carriage return on the keyboard.

DT Response Protocol

The section describes the device response packet format of the DT protocol. The device response packet is a sequence of bytes sent by a device from that device to a host computer after receiving a command package.

Table 3-17 Packet Description for DT Response Protocol

Byte #	Description	ASCII	Hex
1	Start Character	/	2F
2	Controller Address	0	30
3	Status Byte	See *Status and Error Messages* in section 6.	
4 to N	Response (if required)	See the *Commands* chapter in section 5.	
N+1	End of Text	<ETX>	03
N+2	Carriage Return	<CR>	0D
N+3	Line Feed	<LF>	0A
N+4	End (Blank)	<Blank>	FF

Explanation of bytes:

Byte 1: The starting character. 2F hex signals the beginning of a new packet and is the front slash character (/) on a computer keyboard.

Byte 2: The host address, 30 hex (ASCII "0"), is the address number for the host computer.

Byte 3: The status and error byte describes the device status and errors.

Byte 4: There may or may not be response byte(s) for a command. In general, all query commands, read an input value commands, and configuration query commands (~A, ~B, ~P, ~V, etc.) cause response bytes. Other commands do not cause a response.

Byte 4+n: The end-of-response mark is 03 hex.

Byte 4+n+1: The carriage return is 0D hex.

Byte 4+n+2: The end of packet character is the line feed character, 0A hex.

Byte 4+n+3: The extra ending character, FF hex, is an extra character to ensure the packet is properly sent. This character might not be displayed by the host terminal.

OEM Command Protocol

This section describes the command packet format of the OEM protocol. The OEM command format is identical to the Cavro OEM protocol. The command packet is used to send commands from the controlling host device to the syringe drive. Explicit synchronization and error checking are key aspects of this protocol.

Table 3–18 Packet Description for OEM Command Protocol

Byte #	Description	ASCII	Hex
1	Line synchronization character	<blank>	FF
2	Start Transmit Character	<STX>	02
3	Device Address	See *Individual and Multiple Device Addressing* starting in section 3.9.1.	
4	Sequence Number	See *Explanation of bytes*, below	
5	Command(s) (n bytes)	See the Commands chapter in section 5.	
5+n	End of Command(s)	<ETX>	03
5+n+1	Check Sum	See *Explanation of bytes*, below	

Explanation of bytes:

Byte 1: The line synchronization character, FF hex, indicates a command packet is coming.

Byte 2: The start transmit character, 02 hex, signals the beginning of a new packet.

Byte 3: The device address is an address number for a device or a group of devices. Up to 15 devices can be addressed.

Byte 4: The sequence number if an error occurs during the communication. If this happens, the host sends the last packet again to the device with a new sequence number. The sequence number starts with 31 hex (ASSCII). When repeating a command, the host sets bit 3 of the sequence number byte to 1 and increases the sequence number by 1. The valid sequence numbers are hexadecimal 31 for the first packet, hexadecimal 3A for the second packet (the first repeated packet), 3B for the third packet, and so forth. The maximum number of repeats is 7 with a sequence number of 3F.

Byte 5=n: The end-of-command(s) character, 03 hex, indicates the end of a command or command sequence.

Byte 5+n+1: The check sum is calculated by an exculpated-by-exclusive or operation on all bytes except line synchronization byte and check sum byte.

OEM Response Protocol

This section describes the response packet format in the OEM protocol. The OEM response format is identical to the Cavro OEM response format. The responses from the syringe drive to the controlling host device.

Table 3–19 Packet Description for OEM Response Protocol

Byte #	Description	ASCII	Hex
1	Line Synchronization Character	<blank>	FF
2	Starting Character	<STX>	02
3	Host Address	0	30
4	Status and Error Byte	See *Status and Error Messages*, section 6.	
5	Response, if any (n bytes)	See *Commands*, section 5.	
5+n	End of Response Mark	<ETX>	03
5+n+1	Check Sum	See *Explanation of bytes*, below	
5+n+2	Extra Ending Character	<blank>	FF

Explanation of bytes:

Byte 1: The line synchronization, FF hex.

Byte 2: The starting character, 02 hex, signals the beginning of a new packet.

Byte 3: The host address, 30 hex, is the address number for the host computer.

Byte 4: The status and error byte describes the device status.

Byte 5: There may or may not be response byte(s) for a command. In general all query commands, read input commands, and configuration commands (~A,~B,~P,~V, etc) produce response bytes. Other commands do not produce a response.

Byte 5+n: The end-of-response mark, 03 hex, indicates the end of the response byte(s).

Byte 5+n+1: The check sum is calculated by an exclusive or operation on all bytes except the line synchronization byte and the check sum byte.

Byte 5+n+2: The extra ending character, FF hex, is an extra character to ensure the packet is properly sent. This character might not be displayed in the host terminal.

3.9.4. Communication Settings

There are four communications settings: address, bus termination, baud rate, and end protocol. They are set by the configuration commands "**~B*n***" and "**~P*n***" as explained in Configuration Commands, section 5.6. The address is set via the address switch or the wiring on the card edge connector, as explained in *Address Switch*, section 2.5.2. The bus termination setting is explained in *Com Setup Switch*, section 2.5.1.

3.9.5. Connecting Multiple Devices

Up to 15 devices can connected to the same RS-485 communications bus. The bus consists of three wires: A, B, and a signal ground to the interface that is normally done via pins on P1. A proper bus structure consists of the twisted-pair bus wiring and termination resistors at each end of the bus.

Bus Wiring

The bus wiring should connect all RS-485 A pins to one wire, all B pins to another wire, and all Com ground pins to a third wire. The connections begin on the device and proceed from that device to the next device. Another three wires that connect one device to the next should be twisted together with a twist rate from one to three twists per inch. A wiring diagram is shown in Figure 3 9. The wiring to connect a PC to the first drive unit is shown in Figure 2–4.

NOTE: Pumps on the RS-485 bus MUST be wired from pump to pump in series.

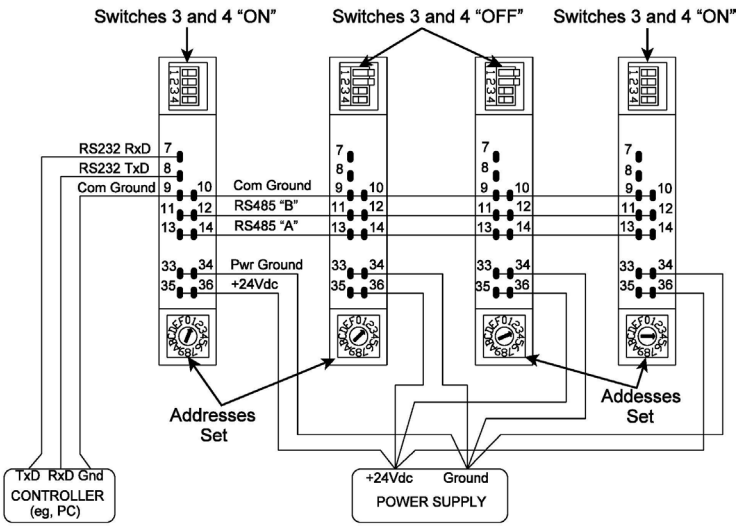
Bus Terminations

Each end of the bus must be terminated in a network which both terminates the bus and provides the proper impedance. Terminations are made only on the first and last devices along the bus as shown in Figure 3–14. Terminating networks are provided on each drive for these purposes.

To terminate the bus, set toggle switches 3 and 4 to the On position for the first and last pumps on the RS-485 bus. All pumps between the first and last must have these toggles set to Off.

NOTE: Only the pump at each end of the RS-485 bus can have the RS-485 bias switched to On. All pumps between the two end pumps MUST have their RS-485 bias switched to Off.

Figure 3–14 Multi-Device Wiring Diagram



3.9.6. Communications Checks

This section presents some procedures to determine if a device is communicating with a host controller. The checks are predicated upon the use of some form of terminal emulator program running on a PC. This is a type of program which sends ASCII characters typed on the keyboard to a serial port and displays the ASCII responses received.

Before using such a program, check the address switch to make sure the pump is set to the desired position on the RS-485 bus. The address switch setting determines the value which must be substituted for the notation <addr>. Each new command string must begin with a forward slash (/) followed by the value of <addr>. Each command string must end with a carriage return (the Return or Enter key). See Commands, section 5, for the command syntax. Note that each key is sent when the key is pressed, so typed characters cannot be edited. Editing keystrokes are sent to the syringe drive and result in syntax errors.

After the communications wiring is connected and the PC serial port cable has been connected to the first device on the bus, turn on the pumps. When the power up valve move is complete, send a query for the module status as shown here:

For example, enter "**/1**" to query the status of pump # 1. The response is "**/0`**" which indicates "not busy, no errors".

If a response occurs, communications are operating properly. A valid response from a communicating module always begins with **/0**. This is the default address of the host controller. If there are no errors to report, the next character after the "**0**" is either an accent grave ("**`**") or"**@**". The accent grave signifies that the module is not busy and is ready to accept any commands. The "**@**" signifies that the module is busy and therefore only queries and the terminate ("**T**") commands are acceptable.

If the query is sent while the power-up initialization sequence is in progress, no response is seen. The pump does not accept commands until the power-up sequence is completed.

If either an accent grave ("`") or "@" is not returned, and a letter is returned in its place, then the module is reporting an internal error condition.

If there is no response, then the pump is not powered up, the communications hardware connection is not properly made, or the communications program is either not properly configured or not operating correctly. Check the following items:

- Make sure that the communications connector is inserted properly into the RS-232 connector and NOT into the RS-485 connector.
- Try another Com port selection.

- Using a voltmeter with the communications cable connected to the RS-232 pins on P1, measure voltages from the GND pin to the RXD IN pin and the TXD OUT pin. Each should measure -6 VDC to -15VDC. If they do not, the following errors may exist.
  - If RXD IN fails the check, the host PC port is not functioning, the communications cable is defective, or it is plugged in backwards.
  - If TXD OUT fails the check, the RS-232 converter board or the communications cable is defective, or the module is not powered.

3.9.7. Communication Drivers

A communications driver is a module, procedure, or function which can be called by a program to send and receive ASCII strings to and from a pump. In general, drivers should be designed to trap (receive and recognize) the status codes returned by a syringe drive in response to a command.

Drivers should limit query rates to not more than 10 queries per second (not less than 90 milliseconds apart in time). Faster query rates, in combination with the 19,200 and 38,400 baud rates, can result in missed queries. In this situation, some queries might not be seen by the pump, resulting in missing responses. Although the probability of a missing query is very low, write driver software so that it allows for an occasional single query which does not cause a response.

NOTE: Consecutive queries should be separated in time by not less than 90 milliseconds or sent at a rate not more than 10 per second.

Norgren Kloehn Inc. provides a driver called KSerial that operates from the command line and can be called from within a user's program. KSerial handles all the communications overhead with Kloehn pumps in both DT and OEM protocols.

4 Programming Techniques

This section presents some application techniques for good system design. Some sections discuss using some of the pump's special features, while others deal with the best programming practices for host controller software development.

4.1. Program Memory

The V6 has the ability to store and execute command strings. A command string is a group of commands that run together, without spaces, to form a single line of legal ASCII characters. Such a string is also called a program.

For example, the following commands:

I	Move valve to input position
A0	Move syringe to fully-closed position
A3000	Fill syringe
M500	Delay 500 milliseconds
O	Move valve to output position
D1500	Dispense half of syringe

can be placed into a single command string (program) as follows:

IA0A3000M500OD1500

In a command string, each new command executes immediately after the preceding command has completed. The command sequence runs in the minimum possible time without the need to query the drive to determine whether it is busy or ready for the next command, which eliminates much communications overhead. This type of program executesimmediately or sometime after the program is sent to the drive. It can be executed from temporary memory (RAM) or from non-volatile memory (NVM).

4.1.1. Temporary Memory

When a command string is sent to the drive, the string is entered into temporary memory (RAM). This memory retains its contents while power is applied to the drive unit. When power is removed, the contents of RAM are lost. After a command string is executed, it can be repeated by sending the "**X**" command.

To execute a program at the time it is sent, append an "**R**" (run) command to the string. If no "**R**" command is appended, the program executes when a subsequent "**R**" command is sent. If another command is sent after the program string and before the "**R**", or if the "**R**" is appended to another command, the original program string is overwritten by the last command string.

For example, the command "**D1000R**" executes as soon as it is received by the drive. The command "**D1000**" is stored into RAM, but does not execute until a separate "**R**" is sent.

4.1.2. Non-Volatile Memory

Non-volatile memory (NVM) retains its contents for at least 15 years without power. Thus, the NVM acts as a "solid-state disk drive". Up to ten programs can be stored in the standard NVM; an expansion option provides additional storage for another 89 programs. There is no practical limit to the number of times a stored program can be read or executed.

NOTE: The maximum number of times that a program string can be saved into the NVM is 10,000. After this, the integrity of a stored program is not guaranteed.

Saving and Erasing a Program

A program string is saved into the NVM by sending it to the RAM without an "**R**" command appended, and then sending the "**E*n***" command as a separate command. When the "**E*n***" command is received, the string in RAM is transferred into the NVM and stored as program number ***n***, where ***n*** is a number between 1 and 99. To erase a command string in NVM, either overwrite it with another command string or send the "**e*n***" (erase) command.

Because of the limitation on writes, NVM should not be written every time an application is run. Use it to store a command sequence or program if that program will be long-lived in the application. Short-term programs such as programs which can change often should be executed from RAM. Some programs which vary in the numbers but not in the structure can use general variables.

Listing a Program

A program saved into NVM can be queried with the "**q*n***" (program query) command. When the "**q*n***" command is received, the drive responds by sending the complete command string, if any, found in the NVM. The command string is terminated with a period. If no command string is found, only the period is returned after the status byte.

Auto-Starting an NVM Program

The V6 has the ability to automatically begin executing a program stored in NVM when power is applied. This feature is known as Auto-Start. This feature is useful for those applications which may require rapid and automatic pump initialization or in cases where a program sequence is controlled with User Inputs or expansion I/O. The Auto-Start feature is enabled by setting the "**~A*n***" parameter to "**1**" with the "**~A1**" command. The Auto-Start can be disabled by setting the "**~A*n***" parameter to "**0**" with the "**~A0**" command. These commands do not require the "**R**" command.

NOTE: The Com setup switch Default toggle switch must be set to Off for the Auto-Start feature to function. The toggle inhibits Auto-Start.

When writing an Auto-Start program, remember that the syringe cannot be commanded to move until it has first been told to initialize to the soft limit with a "**W4**", "**Y4**", or "**Z4**" command. However, all commands other than a syringe move command can be executed before (or without) executing a "**W4**", "**Y4**", or "**Z4**". It is recommended that one of these commands, followed by an "absolute position" move to the zero position ("**A0**"), be included at the beginning of an Auto-Start program. Auto-Start is limited to program numbers 1 through 10.

4.2. Controller Interface Software

This section discusses best practices when writing software to control the pump. These techniques represent years of practical experience in developing applications.

4.2.1. System Initialization

When a pump is installed into a system, the pump may not have the configuration parameters set to the needed values. Such parameters include valve type, I/O operating modes, and initialization parameters. The overall system reliability in the field is greatly enhanced if the controller software can perform the setting of these parameters as needed when a pump is installed.

The parameters are set in non-volatile memory (NVM) by the configuration commands. The number of updates to NVM is limited. For this reason, the parameters should only be set when they are incorrect, and not each time the system is powered on.

NOTE: Do NOT set configuration parameters each time the system is powered on.

A good system programming technique is to read the parameters at system power-on, compare the values to the desired values, and set only those parameters which are not correct. The general form of such code is:

Query <parameter>

if <parameter> is not the <desired value> then Set <parameter>.

For example, if the valve type is 6, the code in the PC host controller might look like this (BASIC language):

Using this technique, factory installations and field replacements are always correctly configured. Use this technique to ensure any required user programs are stored into the NVM. If an external text-based configuration file is used, servicing the system software becomes easy.

4.2.2. Sending Single Instructions

A common error in application programming is to assume a given instruction or instruction sequence will take a fixed length of time to execute. The next instruction is then sent after this fixed delay. This programming technique can lead to system failures.

NOTE: A command might NOT take the same time to execute if an error condition occurs. Do NOT use timing loops as a means of establishing when to send.

The only valid way to determine if the pump is ready to accept another command is to query its status. The simplest way to do this is to send just the address and a carriage return character (hex 0D or decimal 13). A typical query might be:

/**1**<Carriage Return>

Wait for the reply and check the status byte to see if the pump is ready or busy. Do not send the queries too fast. Checking at a rate of eight times per second or less is adequate for nearly all systems. This also permits the controlling software to discover error conditions much sooner than would otherwise occur.

Two situations deserve special acknowledgment: valve moves and traps. If a valve motor stall occurs, the valve will automatically attempt a recovery on the next valve move. The recovery attempt can take up to several seconds. An error trap causes the program to go to a user program to handle the error. This can extend the time required for a program sequence to complete.

A second way to determine when the pump has completed a task or operational cycle is to program the pump to set a User Output to On when each instance of the task is done. A controller can sample the controller input to which the pump output is connected. This technique avoids the communications overhead.

In general, it is better to send commands as groups rather than individually, as each command will execute immediately after the preceding command has finished, without any controller overhead. Periodic status queries are still a good idea.

```
 OPEN  "COM2: 9600, N, 8, 1, CS0, DS0, CD0, RS" FOR RANDOM AS #1
PRINT #1, "/1~V"; CHR$(13)              !--- Query valve type (add <CR>)
FOR n = 0 TO 1000: NEXT n               !--- Pause to receive entire reply
In$ = INPUT$(LOC(1), #1)                !--- Get reply string from pump
 IF MID$(In$, 4, 1) <> "6"THEN '        !--- If the parameter not correct, then set it to the correct value.
PRINT #1, "/1~V6";CHR$(13)
 END IF
CLOSE #1
```

4.2.3. Using Stored Subroutines

Many tasks are repetitive. These include priming cycles, wash cycles, and some fixed I/O routines. The essential nature of such command sequences is that they never vary; they always execute exactly the same way. Such a routine is a good candidate for a stored program. By storing such a routine in the NVM, the overhead of repeating it in a program or sending a long command string is avoided. A program can call a stored program with the call command "**j*n***", where ***n*** is the stored program number 1 to 99. A host software program can call the routine with the stored program run command "**r*n***".

Calling stored routines is a very efficient way to handle repetitive tasks. With the use of general variables, even tasks which vary only in the numbers, but not in the sequence of events, can be stored and called, with the numbers set when the routine is called.

4.3. Pump Programming Tips

This section offers techniques for programming the pump using pump command strings. These techniques extend the usefulness of the pump.

4.3.1. Programming Very Slow Moves
Available with R6 Firmware Pumps Only

The lowest speed at which the syringe rate can be directly programmed by setting the top speed using the "**V*n***" command is 15 steps/second. Much lower speeds are possible using a "step-and-delay loop". Such a loop consists of the following command string:

gD1MpGn

The command string breakdown is:

g...Gn	Repeats the commands between "**g**" and "**G**" for ***n*** times
D1	Dispenses one step
Mp	Pause ***p*** milliseconds (adjust value to acheive desired dispense rate)

This command string can be sent for immediate execution by appending the "**R**" command. For example, in the command string "/**1gD1M13G24000R**" the commands do not execute until the "**R**" is entered.

Dispense speed = 1/Period, where

Period = (*n* + 13) milliseconds.

For example the "**gD1M82G4300**" command dispenses 4300 steps at a speed as found below:

Period = (82 + 13) msec = 95 msec

Dispense speed = 1/95 msec = 10.53 steps/second.

The Start Speed must be in the range of 710 to 1000. The "**v*n***" command sets the start speed (default = 750). Its value can be queried with the "**?1**" command. Start speeds below 710 steps/sec result in large speed and distance errors by a factor of 4.

4.3.2. Programming Error Traps

Errors can occur during operation of the pump. Errors can range from incorrect commands to motor overloads. The pump can detect most error conditions. For some systems, the robustness of the design can be enhanced by programming the pump to take corrective action automatically. This is called "trapping" and the part of the user program designed to handle the error is called the "error handler" or "exception handler".

Error Trapping Example

Trap a syringe overload error. Save the value of the syringe position, initialize the syringe to the input port, then return to the stall position and continue the dispense. This assumes the dispense was intended to deliver all the contents of the syringe.

In the main user program, the trap is set by declaring "**x9V**". If a syringe overload occurs, go to label "**V**".

At the end of the program, where error handlers are normally located, the handler might be as follows:

:V	Identifies the start of the handler (label used in trap instruction above).
k@7	Stores the current syringe position in the software counter for later use.
k^1	Exchanges with the counter memory #1.
k@6	Saves the current valve port position in the software counter.
Y4	Initializes the syringe to reservoir port. The port is set by the "**~Yn**" command.
o@5	Moves the valve back to the previous port position.
k^1	Places the previous syringe position back in the software counter.
A@5	Moves the syringe back to the stall position.
A0	Completes the dispense.
t1	Resumes the program execution with the next instruction.

Limitations

There are some limitations on error trapping. The error trapping feature is designed to provide a graceful recovery from errors, but it cannot fix system errors. Any error induced by mechanical or fluidic problems cannot be fixed by a program. Such things must be fixed at the external root cause. In some cases, a syringe overload might be handled by reducing the syringe speed and trying again.

Example:

Recover from a valve overload by initializing the valve and then repeating the valve move. An "error cycle counter" is included to prevent a run-away loop.

In the main program, zero the counter memory #2 and declare a trap:

k^2 Exchanges the counter with counter memory #2 (preserves k value).

k0 Sets the current counter to zero.

k^2 Restores the current counter and places the zero in counter memory #2.

x10p If a valve overload (error #10) occurs, goes to program label "**p**".

At the end of the program, where error handlers are normally located, the handler might be as follows:

:p Identifies the start of the handler (label used in trap instruction above).

o1 Initializes the valve position.

k^2 Gets counter #2.

k+1 Increments the error loop counter.

k>5q If there are more than 5 valve errors, goes to label "**q**".

k^2 If not, restores the previous counter value.

t4 Tries the valve move again.

:q Label "**q**" means there are more than 5 valve errors (from **k>5q**).

k^2 Restores the previous counter value.

U3 Signals a terminal error (via User Output #3).

t3 Does a normal program error exit (stops the program and makes an error message).

In the preceding example, the "**t4**" exit retries the instruction which caused the error. If the initialize move works, but there are more than 5 retries without success, the handler exits the program after setting an external "error" signal. If the initialize move fails, a program exit occurs. If an error occurs while in the error handler routine, a normal error exit takes place, superseding any user error handler.

NOTE: If an error occurs while an error handler is executing, the program will perform a normal program error exit, regardless of the error handler.

Example:

If any error occurs, set User Output #2 to low, set User Output #1 to high, save the current Digital Voltmeter input value, and then exit the program.

Set the error trap in the main program:

x*s if any error (*****) occurs, go to program label **s**.

At the end of the program, where error handlers are normally located, the handler might be as follows:

:s Label that marks the start of the error handler.

U2 Sets User Output #2 to low.

u1 Sets User Output #1 to high.

k@3 Saves the current Digital Voltmeter input value.

t3 Does a normal error exit.

4.3.3. Setting Syringe Speeds

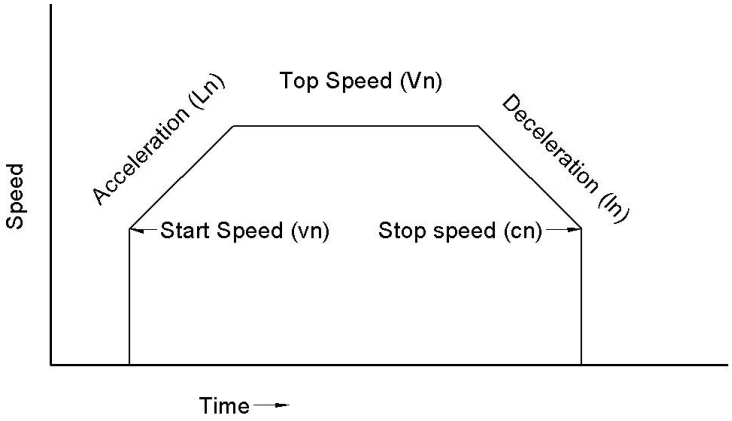
For most applications, the factory default values for syringe accelerations and speeds are adequate. Usually, only the top speed is changed for different syringe rates. If the top speed is set lower than the start speed, the pump will begin a move at the top speed. If the top speed is set lower than the stop speed, the move will end at the top speed. For this reason, values of top speed which are set lower than either the start speed or the stop speed do not require any adjustment in start speed or stop speed.

The default values of the tart and stop speeds have been set to perform well for nearly all normal applications. On occasion, it may be useful to change the speeds from the default settings. This section explains the considerations involved.

The syringe uses three speeds and two accelerations which can be set. The speeds are Start Speed, Top Speed, and Stop Speed. The two accelerations are the "**L*n***" and "**l*n***" values which set acceleration and deceleration rates, respectively.

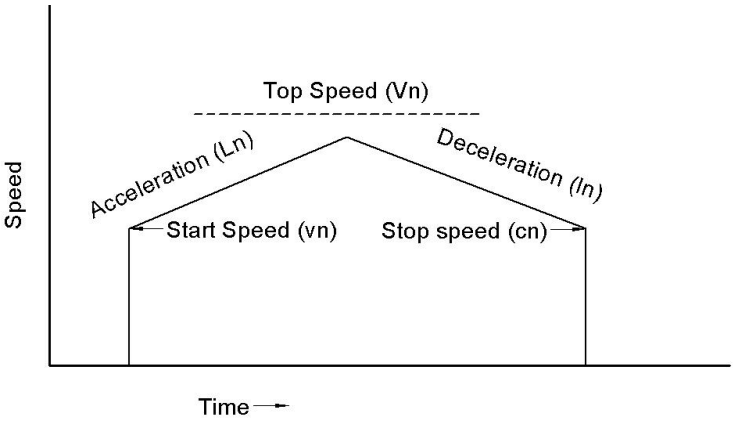
The syringe motor does not start at zero speed and accelerate smoothly to the top speed (at which syringe moves normally occur). Rather, the motor jumps abruptly from zero to the start speed and then accelerates smoothly to the top speed. The move proceeds at the top speed. As the destination is approached, the motor decelerates from the top speed to the stop speed. When the stop speed is reached, the motor performs an abrupt stop at the target position. The speed profile is thus trapezoidal.

Figure 4-1 Normal Syringe Speed and Profile



If a high top speed and low acceleration are combined with a very short move, the syringe speed may not reach the programmed top speed and the profile of the following figure will result.

Figure 4-2 Slow Acceleration or Short Move Speed Profile



In actual practice, a typical move spends nearly all the time at the top speed and the acceleration and deceleration are very small parts of the total move.

Acceleration vs. Top Speed

When selecting Acceleration and Top Speed, there is a trade-off between the two values. The acceleration of the system inertia (pump inertia + fluid inertia) uses part of the available motor power. The motion of the fluid requires additional motor power to overcome back-pressure. If the sum of acceleration power and back-pressure power exceeds the capacity of the motor, the syringe motor will stall and the pump will generate a "syringe overload" error. One or both of the values for Top Speed and Acceleration would need to be reduced.

Pressure Rise

The back-pressure of a fluid in motion is greater at the syringe than at the delivery point. The difference between the two is the "pressure rise". The pressure rise can exceed several hundred psi (tens of atmospheres) in some cases. Here are some of the factors which contribute to the pressure rise and consequent back-pressure:

Factor	Effect
Path diameter	Pressure varies as fourth power of diameter.
Path length	Pressure varies directly with length.
Fluid velocity	Pressure varies with square of velocity.
Temperature	Temperature changes viscosity, which changes back-pressure. Over 5:1 variations are possible.

All these effects are cumulative. By far the most sensitive is the path diameter. An increase of just 19% in the inside diameter of the tubing or an orifice can drop the back pressure by as much as 50%. One source of gradually increasing back-pressure is too much torque on the fittings used with the valve. The valve requires Teflon® washers to seal the fitting-to-valve connection. If the fittings are tightened too much, the pressure of the connection will cause the Teflon to "cold flow" in a way that reduces the size of the hole in the washer. In extreme cases, the hole can shrink to a pin-hole.

Higher fluid velocities have two reinforcing effects: (1) the pressure increases as the square of the velocity, and (2) the available motor power decreases with increases in speed. If syringe overloads are occurring, small reductions in Top Speed can produce major improvements in system reliability.

If a move fails to begin, the problem may be too high of a start speed or a blocked fluid path. In general, the default value of start speed is a good compromise.

If the pump generates frequent "**Z**" errors (error #26), the cause may be a stop speed set too high. Excessive momentum can overcome the magnetic braking of a stopped motor. An abrupt stop from too high of a speed may cause the syringe position to become corrupted, resulting in the error message. In nearly all cases, no value for programmed deceleration will result in this problem.

When adjusting the speeds, consider the trade-offs and consequences. Most problems are the result of too high of a top speed or too high of a stop speed. The top speed must be set to accommodate the dimensions of the fluid path, the fluid viscosity, and the decreasing thrust force as speed increases.

4.3.4. Counting Program Cycles

The number of times a given event has occurred can be determined with the Software Counter. In the example below, the number of times a programmed dispensing sequence has occurred is counted so a controller can query the pump to determine the number of dispenses which have occurred since the program was initiated.

For example, count the number of times a dispense has been made in an automated dispensing cycle:

K0	Sets the counter to zero (start of dispense program).
:B	Marks the start of the filling of the syringe.
o-1	Moves the valve to the reservoir port.
A48000	Fills the syringe (48000-step model).
O3	Moves the valve to the dispense port.
:A	Marks the start of the dispense loop.
Y<1500B	If there is not enough left to dispense, refill (goes to label "**B**").
D9600	Dispenses 20% of a syringe (48000-step model).
K+1	Increments the software counter.
JA	Does another dispense loop.

In the example above, the program begins by setting the counter to zero.

The syringe is refilled each time the syringe position is too small to do another dispense. After each dispense, the counter is incremented by 1. The counter can be queried at any time to read how many dispenses have occurred. The complete program string is:

k0:Bo-1A48000o3:Ay<1500BD9600k+1JA

4.3.5. Converting Volume to Steps

The conversion of syringe volume to steps (syringe increments) can be easily done using a proportion.

Steps required / Total steps in full stroke = Desired volume / Total volume of full stroke

For example, assume a total syringe volume of 5mL (5000uL), a desired volume of 250uL, and a 48000 step model syringe drive. The required number of steps can be found by solving:

Steps required = 48,000 * (250/5000)

Steps required = 2400

The preceding proportion can be used to find speeds also if the "steps required" is replaced with "steps per second" and the "desired volume" is replaced with "volume per second".

For example, assume a totally syringe volume of 5mL (5000uL), a desired .5mL per second (500 uL/second) dispense, a 48000 step syringe drive. The required speed, steps per second, can be found by solving:

Steps per second = 48,000 steps * ((500 uL/second) / 5000 uL)

Steps per second = 4,800 steps per second to achieve the example desired flow rate, 500 uL per second.

4.4. I/O Interface Programming

User Inputs and User Outputs (I/O) can be used to perform a variety of interfacing tasks. In stand-alone operation, without a serial communications controller, the I/O can be used to trigger program operations and to indicate operational status. In multiple-pump operations, the I/O can be used to coordinate and synchronize the operations of the pumps. One special case is the synthesis of continuous fluid flow using two pumps. This section provides some interfacing techniques to aid in using the pump I/O.

4.4.1. Waiting for an Input

In many applications, it is desirable for a pump to wait for an external input signal to start an operation. Test-and-jump commands are used to sense the state of an input signal at a User Input and control the program operation.

There are two basic scenarios:

- → Wait for a high level
- → Wait for a low level

Example: Wait for a High Input Level

:A	Program label "**A**"
i2A	If User Input #2 is low, goes back to label "**A**"

When User Input #2 goes high, the "**i2A**" instruction is false and the jump back to label "**A**" is not taken. The next instruction in the line is executed. It should be noted the User Inputs have internal pull-up resistors, so in the absence of a signal connection, the default input level is high.

Example: Wait for a Low Input Level

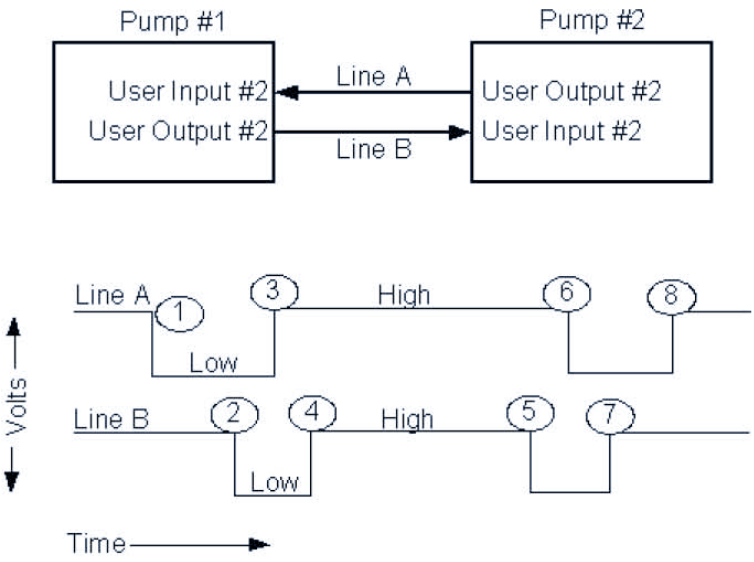
:r	Program label "**r**"
i3T	If input #3 is low, goes to label "**T**"
Jr	Always jumps to label "**r**"
:T	Label "**T**"

While the input is high, the test for User Input #3 to be low fails and the jump to "**T**" is not taken. The next command, "**Jr**", is therefore executed. The "**Jr**" command sends the program execution back to the "**r**" label and the test repeats. When User Input #3 goes low, the jump to label "**T**" is taken. Label "**T**" then leads to the next instruction in the program.

4.4.2. Handshaking Between Pumps

Handshaking between pumps uses the techniques in the previous section to synchronize the operations of two or more pumps. Each pump sends a signal to other pumps while monitoring the signals from the other pumps. Each pump therefore waits for another pump to trigger its next operation. The example below illustrates the nature of the bidirectional trigger signaling, called a "handshake".

Figure 4-3 Handshake Operation



This figure shows the wire connections for a handshake. It is assumed the necessary ground connection between the pumps is present. Each pump has one output connected to and input on the other pump. This allows each pump to send a signal to the other pump. The sequence of operations is illustrated in the voltage-time diagram of this figure.

First Handshake:

At the beginning, both outputs are idle at the high level.

At (1), pump# 2 sets its User Output #2 low to signal pump #1 to begin its next operation.

At (2), pump #1 sees the signal and sets its User Output #2 low to indicate "signal seen".

At (3), pump #2 sees the response from pump #1 and resets its User Output #2 back to high.

At (4), pump #1 sees the return high from pump #2 and sets its User Output #2 back to high.

After (4), pump #1 begins its next operation and both User Outputs are in the initial state.

Second Handshake:

Initially, both User Outputs are in the idle high state.

At (5), pump #1 sets its User Output #2 low to signal pump #2 to begin its next operation.

At (6), pump #2 sees the signal and sets its User Output #1 low to indicate "signal seen".

At (7), pump #1 sees the response from pump #2and resets its User Output #2 back to high.

At (8), pump #2 sees the return high from pump #1 and sets its User Output #2 back to high.

After (8), pump #2 begins its next operation and both outputs are in the initial state.

The handshake command strings are the same for each pump. Any of the User Inputs and Outputs could have been used. The handshake command strings for the preceding example are as follows:

Starting a Handshake:

u2 Sets User Output #2 to signal "Go" to the other pump.

:A Label **A**.

i2B If the input is low, goes to label **B**.

JA If the input is not low, goes to label **A**.

:B Input is low.

u2 Sets User Output #2 high again to say "signal is seen".

Receiving a Handshake:

:C Label **C**.

i2D If User Input #2 is low, goes to label **D**.

JC Otherwise go to label **C**.

:D User Input #2 is low.

u2 Sets User Output #2 to low to say "signal is seen".

:E Label **E**.

i2E If User Input #2 is still low, checks it again.

u2 User Input #2 went high, so set User Output #2 back to high.

4.4.3. Programming Continuous Flow

When a continuous fluid flow is required, the handshake dispense commands can be used to synthesize the flow using two pumps. The resulting flow is continuous, with no gaps in delivery. The input ports of both pumps are connected with a **T**-fitting so a single source line is used. The output ports of both pumps are connected together with a **T**-fitting to sum the two outputs into a single delivery line.

As one pump is dispensing, the other pump is refilling. When pump A approaches the end of its dispense, it signals pump B to begin to dispense. While pump B is dispensing, pump A refills itself. When each pump completes its dispense, it sends a handshake signal that triggers the dispense for the other pump and the fluid flow is thus continuous.

The syntax of the handshake dispense is "**h*n***" (handshake dispense), where ***n*** is the User Input number.

When this instruction is encountered, the pump begins testing User Input **#*n*** for a low level. When a low level is seen, a dispense begins. The dispense is always from the current syringe position to zero (the full-dispense position). As the dispense nears the zero position, the User Output **#*n*** is set low just before the dispense ends. This is used as advance notice to the other pump to start its own handshake dispense. All these I/O operations are automatic.

To begin a handshake dispense without an external trigger signal, use the syntax "**h-*n***". This is useful to start a handshake dispense operation or to resume one that has been stopped.

Handshake Sequences

The handshake sequences below assume the syringes are both filled and the valves are both at the dispense port.

Initiating pump:

V2000	Sets the dispense speed.
h-n	Begins the handshake dispense immediately.
:S	Label **S**.
o1	Moves the valve to the input port.
V6000	Sets the filling speed.
A48000	Fills the syringe.
o3	Moves the valve to the dispense port.
V2000	Sets the dispense speed.
hn	Does a standard handshake dispense.
JS	Goes to label **S** for another cycle.

Other pump:

:S	Label **S**.
o1	Moves the valve to the input port.
V6000	Sets the filling speeds
A48000	Fills the syringe.
o3	Moves the valve to the dispense port.
V2000	Sets the dispense speed.
hn	Does a standard handshake dispense.
JS	Goes to label **S** for another cycle.

Note the initiating pump differs only in the first two lines, which are needed to start the overall process. Once begun, both pumps use the same handshake command sequence. Special note should be taken of the speeds. The refill speed must be faster than the dispense speed by a difference that allows the refilling pump to make two valve moves and refill the syringe before the dispensing pump completes its dispense.

4.4.4. The DVM as a Selector Switch

The Digital Voltmeter (DVM) input can be used as a selector switch. If repeatable voltage levels can be applied to the DVM input, a series of tests can be made to determine what the program should do next. These levels can be generated from a digital-to-analog converter or from a series of resistors soldered around a selector switch to form a tapped voltage divider.

In the following command sequence, each test level is chosen to be half-way between the actual voltage levels, not at the voltage levels. This is done for noise immunity.

For example, assume six discrete voltage levels at 0, 1, 2, 3, 4, and 5 volts. The test thresholds are at 0.5, 1.5, 2.5, 3.5, and 4.5 volts for a total of five possible selections.

The actual test numbers are found as: number = volts / 0.02

Selection 1: 0.5 volts = 25

Selection 2: 1.5 volts = 76

Selection 3: 2.5 volts = 127

Selection 4: 3.5 volts = 178

Selection 5: 4.5 volts = 229

The test sequence uses the numbers calculated above.

:A Labels that mark the start of the test loop.

i>229B If the selection is 5, goes to label **B** (start of selection 5).

i>178C If the selection is 4, goes to label **C** (start of selection 4).

i>127D If the selection is 3, goes to label **D** (start of selection 3).

i>76E If the selection is 2, goes to label **E** (start of selection 2).

i>25F If the selection is 1, goes to label **F** (start of selection 1).

JA If there is no selection, tests the loop again.

The order of the tests is critical. If the order were reversed, all selections would satisfy the test for selection 1. Each label **B** through **F** denotes the place in the program at which the commands for each different selection begins. The command sequence of reach selection must end with a jump back to Label **A** ("**JA**") so the selection process continues when each selection is done.

Voltage conversions are not exact, so a range should be used when testing instead of a precise value. There is some inherent jitter in both the conversion process and the tolerances in the components used to generate the voltage. The example above uses the midpoints between discrete voltage levels to define the test range.

4.4.5. Position Snapshots

The pump provides a means to record the precise location of the syringe at the time of an external event. Such information is useful in titrations and other applications in which an external sensor detects an event while the syringe is in motion. Although the syringe position can be queried while the syringe is in motion, the communications overhead prevent the exact syringe position from being determined through "on the fly" position queries.

The snapshot feature overcomes these limitations and provides exact measurements. The snapshot feature uses the User Input #3. Each time the User Input #3 transitions from high-to-low, the current position of the syringe is stored in memory. The snapshot memory retains the positions for the two most recent input transitions. The syringe positions for these two points can be queried at any time with the "**?29**" query. This feature is always available regardless of the Input #3 operating mode.

The snapshot feature is very useful for titrations using optical detection, where the detector outputs a double pulse and the two positions at the time of the pulses must be known. It is also useful in any application in which the amount to be dispensed is unknown in advance and must be determined "on the fly" during the process.

4.4.6. Using the Expansion Port

The V6 pump includes three digital inputs, three digital outputs, and a Digital Voltmeter input as part of the standard unit. For those applications that need more I/O or different I/O, an expansion I/O port is also included. The expansion I/O port is an SPI (Serial peripheral Interface) synchronous serial port that can increase the I/O pins by either one byte (eight bits of input + eight bits of output) or two bytes (16 bits of input + 16 bits of output). Kloehn makes an I/O Expander Board that implements the two-byte expansion.

The I/O Expander Board is an example of the I/O increase possible with the expansion port. This provides 16 extra inputs which can be operated as two bytes or bit-by-bit, and 16 extra outputs, each of which can switch up to 250 milliamperes at up to 40 VDC. These can be used to control solenoid valves, relays, indicator lights, or other motors.

The expansion I/O port consists of four SPI signals and two power leads that provide +5 VDC power for external circuits.

I/O Commands

The Expansion I/O is managed by extensions of the basic I/O commands as described in I/O Commands, section 5.3. The relevant commands are "**sn**", "**sn,m**", "**Un**", and "**un**". The outputs are controlled by these commands, which can be included in a program.

Query Commands

The inputs are read by the "**?10**", "**?n**", and "**?20**" commands, which are executed immediately upon receipt and cannot be included in a program. For more information on these commands, see *Input Query Commands*, section 5.3.2.

Program Control

Program control using the Expansion I/O is possible with the "**i*np***" command, which can be included in a program:

For example, if the input level is true (low input level), the "**i*np***" command jumps to label **p**. This checks if an expansion input bit is at a low level.

n: 11...18 bit 1...8 in Expansion input byte #1
21...28 bit 1...8 in Expansion input byte #2
p: a...z, A...Z

The byte value of an Expansion I/O byte can be used to control program flow with the Software Counter test-and-jump commands using an indirect variable.

Example:

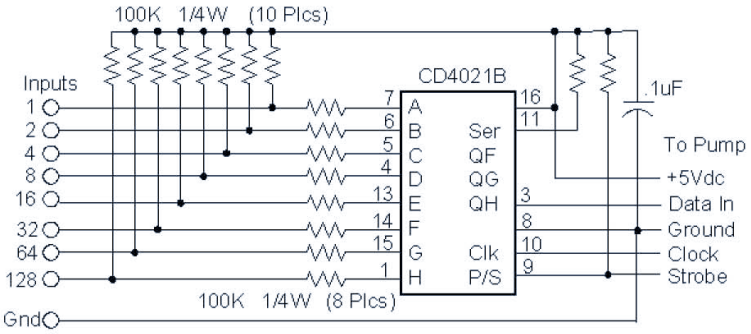
k@1 Loads the Software Counter with the first Expansion input byte.

k<np If the counter is less than ***n*** (value of first byte) then the command goes to label ***p***.

Expansion Hardware Design

Expansion hardware design is simple, as most common shift registers can be used to transfer data into or out of the Expansion I/O port, as shown in this figure.

Figure 4-4 Sample 8-bit Input Circuit for Expansion I/O



In the circuit of this figure, the user inputs are labeled with their equivalent binary weights, with "128" as the most significant bit. The inputs are shown with pull-up resistors (4.7K) and series input protection resistors (100K). These resistors are highly recommended. This circuit provides an extra eight bits of input in the one-byte mode.

If a two-byte version is needed, add a second CD4021B. Connect the QH of the second chip to the serial input of the first chip. Also connect the two Clock inputs together.

4.4.7. Generating External Pulses

In some applications, it may be useful to generate pulses on one of the User Outputs. This can be done with a simple repeat loop as shown below:

gU2M10u2M20G16

Where:

g Start of the repeat loops.

U2 Turns on User Output #2.

M10 Makes the negative pulse width 10 milliseconds.

u2 Turns off User Output #2.

M20 Makes the high pulse width 20 milliseconds.

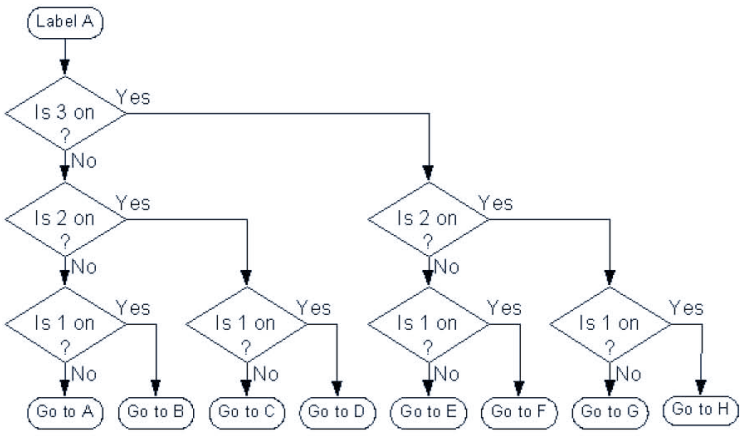
G16 Generates 16 pulses.

If the two time delays are omitted to obtain the maximum pulse rate, the resulting pulse rate is about 400 pulses per second.

4.4.8. A Binary Input Selector

In some applications, one of several selections must be made via a selector switch or PLC logic lines. The most economical approach is to encode the inputs as binary numbers. This section describes a way to program a binary selection tree. This is a way of making the input bits act as if they have a binary weighting (e.g., 1, 2, 4, etc.). This illustration uses all three inputs as a seven-way selector. The expansion I/O could also have been used to input the bits with the same instructions. This figure shows a flow chart of the algorithm.

Figure 4-5 Binary Selection Tree Flow Chart



The corresponding commands are:

:A Label **A** marks the start of the selection tree.

i3K Is User Input #3 on? If yes, goes to label **K**.

i2L Is User Input #2 on? If yes, goes to label **L**.

i1B Is User Input #1 on? If yes, goes to label **B**.

JA Goes to label **A** (no selection).

:L Label **L**.

i1D Is User Input #1 on? If yes, goes to label **D** (selection = D).

JC Goes to label **C** (selection = C).

:K Label **K** (second half of the binary tree).

i2M Is User Input #2 on? If yes, goes to label **M**.

i1F Is User Input #1 on? If yes, goes to label **F** (selection = F).

JE Goes to label **E** (selection = E).

:M Label **M**.

i1H Is User Input #1 on? If yes, goes to label **H** (selection = H).

JG Goes to label **G** (selection = G).

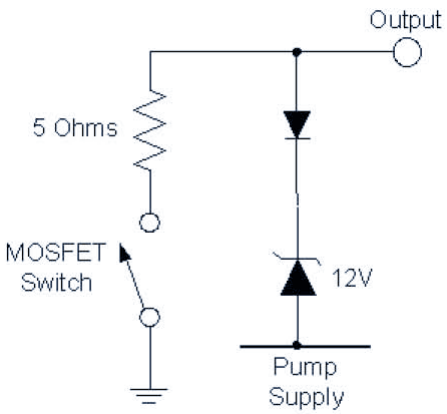
A three-way selection tree uses only the commands from "**:A**" through "**JG**". Each of the labels "**B**" through "**H**" is the start of the command sequence which corresponds to the input selection. Each selection command sequence must end with a "**JA**" (jump to label "**A**") so the input selection procedure can continue when a process is done. Note the selection A goes back to the start of the selection tree. This is done to provide a "no selection" position. If some other means of initiating the selection process (other than a valid selection) is used, then label "**A**" can be used for an eighth selection.

4.4.9. Connecting the User Outputs

The User Output #1, #2 and #3 are suitable for driving a variety of loads. The open drain MOSFET outputs can drive up to 170 milliamps (mA) and withstand voltages to 12 Volts greater than the pump supply voltage, up to a peak of 50 VDC.

Inductive loads, devices like relays or solenoids which incorporate a coil can discharge a significant amount of current when turned off. This is called an "inductive load". Each User Output includes integral protection against inductive turn-off transients (Figure 4–6). The zener diodes trigger at about 13 volts across an inductive load during the current decay to speed up the inductive load turn-off. Typical inductive loads are solenoids, solenoid valves, relays and small DC motors.

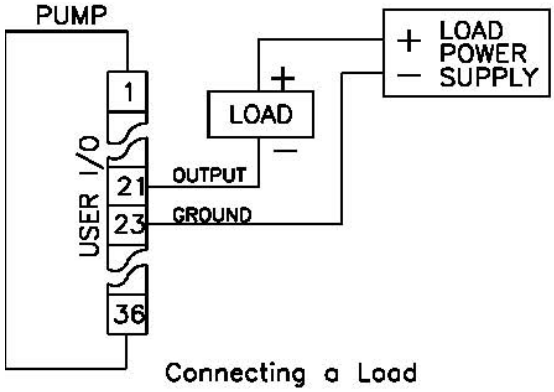
Figure 4-6 Equivalent Circuit of Output

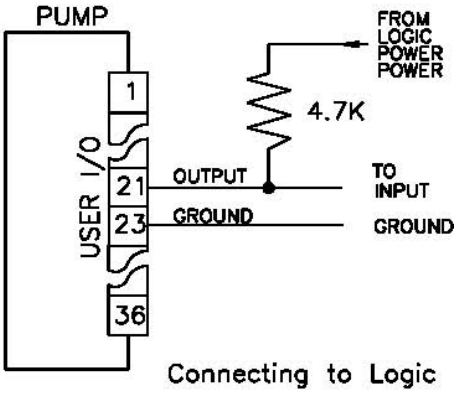


Logic Loads

Logic loads, such as CMOS and TTL inputs, require a pull-up resistor to the logic supply voltage, as shown the following figure. Other loads such as indicator lights, relays, optoisolators and solenoids usually operate from higher voltages. The connection for such loads is also shown in the following figure. Note the load is connected from the output to the load and from the load to the load supply. The ground connection from the load supply ground to the pump ground is essential.

Figure 4-7 Connecting Output Loads





5 Commands

This chapter presents the commands supported in the V6 pump.

Table 5-1 Commands

Commands	Type	Section
Syringe	Position Commands	5.1.1
	Motion Variables	5.1.2
	Initialization Commands	5.1.3
	Syringe Queries	5.1.4
Valve	Valve Type Setting	5.2.1
	Valve Position Commands	5.2.2
	Valve Queries	5.2.3
I/O	Output Commands	5.3.1
	Input Query Commands	5.3.2
	Input Test and Jump Commands	5.3.3
User Program	Program Storage Commands	5.4.1
	Program Execution Commands	5.4.2
	Program Control Commands	5.4.3
Variables	Setting a General Variable	5.5.1
	Using a General Variable	5.5.2
	Indirect Variables	5.5.3
	List of Commands Using Variables	5.5.4
Configuration	–	5.6
Query	–	5.7
Error Trapping	Trap Declarations	5.8.1
	Trap Exits	5.8.2
	Error Trap Query	5.8.3
Miscellaneous	Software Counters	5.9.1
	Flags	5.9.2
	Set Home Button Control	5.9.3
	External Syringe Motion Limit Input	5.9.4
	Motor Power Control	5.9.5
	Repeat Command String	5.9.6

When reviewing the commands in the following sections, keep note of the following items:

- Values in parenthesis () indicate the range of values.
- Values in brackets [] represent the factory default values.
- The notation **@*n*** signifies that the argument may be an indirect variable.

NOTE: The non-volatile memory is limited to 10,000 update cycles. For this reason, use configuration commands only when a specific operating configuration must be changed. A configuration setting is stored for life on the pump or until it is changed.

5.1. Syringe Commands

5.1.1. Position Commands

Position commands cause the syringe to move to a commanded position along its range of motion:

- An absolute position is a specific point.
- A relative position is a distance offset from the current position.

Absolute versus relative positions are illustrated in Figure 5-1.

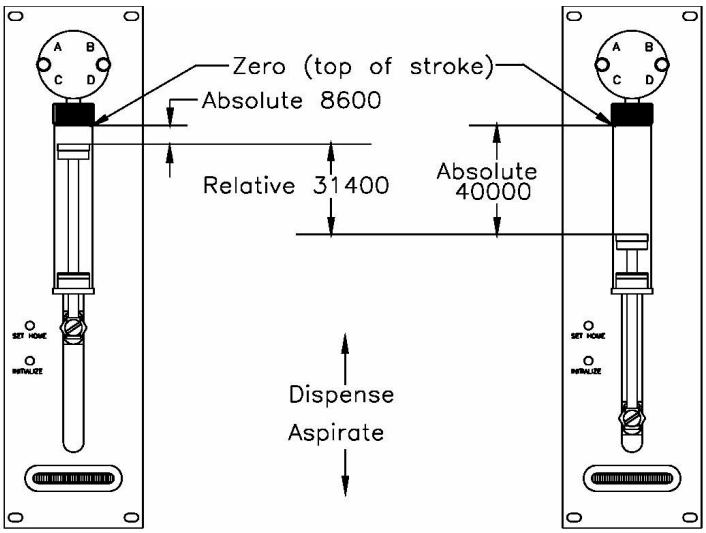
It is highly recommended that only the upper case form of the "**A*n***", "**B*n***", and "**C*n***" commands are used so that the Busy status can be ascertained. Lower case versions of these commands do not reveal the Busy status upon query.

In each of the commands in this table, the ***n*** value is expressed in steps, where "***0***" is at the top-of-stroke (volume).

Table 5-2 Position Command Definitions

Command	Description
A*n*	Go to absolute position ***n***, with the BUSY status bit set to "busy". (*n*: 0....12000 steps, 0....24000 steps, or 48000 steps, @n) [n/a]
a*n*	Go to absolute position ***n***, with the BUSY status bit set to "ready". (*n*: 0....12000 steps, 0....24000 steps or 48000 steps, @n) [n/a]
D*n*	Dispense ***n*** steps from the current position, with the BUSY status bit set to a "busy". The dispense direction is upward, towards the valve. (*n*: 0....12000 steps, 0....24000 steps or 48000 steps, @n) [n/a]
d*n*	Dispense ***n*** steps from the current position, with the BUSY status bit set to a "ready". The dispense direction is upward, towards the valve. (*n*: 0....12000 steps, 0....24000 steps or 48000 steps, @n) [n/a]
P*n*	Aspirate (pick up) ***n*** steps from the current position, with the BUSY status bit set to "busy". The aspirate direction is downward, away from the valve. (*n*: 0....12000 steps, 0....24000 steps or 48000 steps, @n) [n/a]
p*n*	Aspirate (pick up) ***n*** steps from the current position, with the BUSY status bit cleared to "ready". The aspirate direction is downward, away from the valve. (*n*: 0....12000 steps, 0....24000 steps or 48000 steps, @n) [n/a]

Figure 5–1 Relative vs. Absolute Positions



A relative position is measured from the current position to the target position. Absolute position is measured always from the zero position (top-of-stroke). In Figure 5–1, a move from the upper position at 8600 absolute to the lower position at 40000 absolute can be done with either a relative aspirate (move downward) or an absolute (go to position) command as follows.

Absolute move: **A40000** (final position measured from zero)

Relative move: **P31400** (final position measured from 8600)

Both commands will result in the syringe moving to the position shown on the right side of the figure.

In general, any move which goes to the zero or maximum (full-stroke) positions should use an absolute positioning command such as "**A0**" or "**A48000**". A move which goes from one position to another position which is not at either end of the stroke would use a relative positioning command such as "**P*n***" or "**D*n***". Absolute positioning command can also be used.

→ Use an absolute position command for all moves to the zero or to the full-stroke position. For example, "**A0**" or "**A48000**".

→ Use the capitalized version of the "**A*n***", "**P*n***", and "**D*n***" commands. Lower-case versions will not report a Busy status if the pump is queried while moving.

Handshake Dispense Commands

Table 5–3 Handshake Dispense Command Definitions

Command	Description
h*n*	Do a handshake dispense, using User Input #n and User Output #*n* for the handshake signals. (*n*: 1...3, @n)
h-*n*	Trigger a handshake dispense immediately, without an external input stimulus. (*n*: 1...3, @n)

The handshake dispense is a coordinated sequence between two pumps in which one pump dispenses while the other pump aspirates. When one pump completes its dispense, the other pump begins its dispense. By summing the outputs of the two pumps, a continuous flow can be synthesized.

The coordination between the two pumps is done automatically by the pumps when their inputs and outputs are connected. The value of **n** in the handshake dispense commands determines which of the user inputs will be used for the handshake coordination. For example, if the command is "**h2**", then the pump will use User Input #2 and User Output #2.

As each pump nears the end of its dispense, it provides an advance trigger signal to the other pump, which immediately begins its own dispense. The timing on the trigger signal is automatically adjusted to compensate for different acceleration settings.

5.1.2. Motion Variables

The syringe axis uses the following variables to determine the speeds, acceleration, and drive compensation moves. During power up of the pump, default values in the operational memory are recalled from the NVM. The operational values can be set at any time a move is not in progress. Top speed is an exception, as it can be set "on-the-fly".

Except as noted, all of these commands require an "**R**" (Run) command to execute immediately.

Table 5-4 Motion Variable Command Definitions

Command	Description
C*n*	Sets the stop speed to ***n*** steps per second (sps). (*n*: 40...10000, @n)[750]
c*n*	Sets the stop speed to ***n*** steps per second (sps). (*n*: 40...10000, @n)[750]
K*n*	Sets the number of syringe backlash steps to ***n*** steps. Backlash compensates for mechanical slack in the drive system. Too little backlash compensation results in an error in the first dispense movement that follows an aspirate move. (*n*: 0...1000, @n)[100]
L*n*	Sets the acceleration and deceleration slope to ***n***, where the actual acceleration value in steps per second per second equals **n** x 2500 sps/sec. (*n*: 1...20, @n)[7]
l*n*	Sets the deceleration slope independently to ***n***, where the actual deceleration value in steps per second per second equals **n** x 2500 sps/sec. (*n*: 1...20, @n)[7] Examples: **L12** Sets the acceleration and deceleration parameters both to 12. **l15** Sets only the deceleration parameter to 15.
S*n*	Sets a predefined syringe speed. The speeds in the table below are in steps per second (sps). (*n*: 0...36, @n)[n/a]
V*n*	Sets the syringe top speed to **n** steps per second (sps). The top speed is the rate at which dispenses and aspirates move. Top speed can be changed "on-the-fly" during a syringe move. For speeds above 100 sps, changes in speed which are too large may stall the syringe motor. Do not use an "**R**" command when this command is sent by itself. (*n*: 40...10000,@n)[5000]
v*n*	Sets the start speed to **n** steps per second. (*n*: 40...1000,@n)[750]
!	Stores the current value of the top, start, stop speeds and backlash as the default values to be used after each power-up or reset. This command cannot be stored within a program.

S*n*	sps	S*n*	sps	S*n*	sps	S*n*	sps	S*n*	sps
0	6400	9	1800	18	190	27	100	36	15
1	5600	10	1600	19	180	28	90		
2	5000	11	1400	20	170	29	80		
3	4400	12	1200	21	160	30	70		
4	3800	13	1000	22	150	31	60		
5	3200	14	800	23	140	32	50		
6	2600	15	600	24	130	33	40		
7	2200	16	400	25	120	34	30		
8	2000	17	200	26	110	35	20		

For most applications, only acceleration and top speed are adjusted. The remaining parameters are left at the default settings.

Top Speeds Below Stop or Start

If the top speed is set to a value lower than the start speed, the pump begins to move at the top speed. If the top speed is set lower than the stop speed, the move will end at the top speed. For this reason, values of top speed which are set lower than either the start speed or the stop speed do not require any adjustment in the start speed or stop speed.

5.1.3. Initialization Commands

The syringe position must be initialized (calibrated) after each power-up, reset, or syringe overload condition. Until the syringe is initialized no other syringe movement commands are accepted. This is because the syringe position cannot be absolutely known after the preceding conditions occur.

An initialization command causes the syringe to go to the Initialize position. This is the only absolutely known location on the syringe stroke when position information is lost or corrupted. All other positions can be determined once this position is known.

Initialization uses the "**W*n***", "**Y*n***", or "Z***n***" commands. Each operates in the same way except for the definition of the valve port positions used during initialization. For all initialization commands, the argument n denotes an initialize move (***n*=4**) or a set home operation (***n*=5**).

The "**W4**" command always initializes the syringe using port A. The "**Y4**" and "**Z4**" commands initialize the syringe using the valve port which has been set by the "**~Yn**" or "**~Zn**" commands, respectively. This permits three different ports to be used for syringe initialization.

The **Initialize** button on the front panel executes only the "**W4**" command. The **Set Home** button executes all "**W5**", "**Y5**", and "**Z5**" commands, as there is no practical difference between the three versions.

The initialize commands require an "**R**" command to execute immediately.

Table 5-5 Initialization Command Definitions

Command	Description
W*n*	Initializes the syringe. This must be used on systems with on valve. (*n*: 4 or 5)[n/a] • *n*=4 Moves the syringe to the Initialize position after moving the valve to port A. • *n*=5 Sets the current syringe position to the zero position. The result is automatically stored in non-volatile memory (NVM).
Y*n*	Initializes the syringe after moving the valve to the port corresponding to the number stored for the "**~Y*n***" configuration command. (*n*: 4 or 5, see the "**W*n***" command for an explanation of n values)[1]
Z*n*	Initializes the syringe after moving the valve to the port corresponding to the number stored for the "**~Z*n***" configuration command. (*n*: 4 or 5, see the "**W*n***" command for an explanation of n values)[1]
~Y*n*	Selects the valve position that the valve uses while initializing the syringe. This parameter value is used by the "**Y*n***" command. This command checks the "**~V**" parameter to determine which port numbers are acceptable. (*n*: 1...12)[1]
~Z*n*	Selects the valve position that the valve uses while initializing the syringe. This parameter value is used by the "**Z*n***" command. This command checks the "**~V**" parameter to determine which port numbers are acceptable. See the "**~Y*n***" command above for the number-to-port letter translation. (*n*: 1...12)[1]

n	Port	*n*	Port	*n*	Port
1	A	5	E	9	I
2	B	6	F	10	J
3	C	7	G	11	K
4	D	8	H	12	L

5.1.4. Syringe Queries

Table 5-6 Syringe Query Definitions

Query	Description
?	Queries the absolute position of the syringe and returns the value in steps from zero position (top-of-stroke).
?1	Queries the start speed of the syringe (v*n* value) in equivalent steps per second.
?2	Queries the top speed of the syringe (V*n* value) in equivalent steps per second.
?3	Queries the stop speed of the syringe (c*n* value) in equivalent steps per second.
?29	Queries the contents of the syringe position snapshot memory. Returns the value in steps from zero position (top-of-stroke).
?30	Queries the acceleration and deceleration number (L*n* and I*n* values). The first return number is acceleration and the second is deceleration.
?31	Queries the number of syringe backlash steps.
~Y	Queries the valve port number used by the Y*n* syringe initialization.
~Z	Queries the valve port number used by the Z*n* syringe initialization.

5.2. Valve Commands

The valve ports are not inherently directional. The actual direction of fluid flow at any port is determined by the relative motion of the syringe. An *aspiration* draws fluid into a port and a *dispense* ejects fluid from a port.

For non-distribution valves, some valve positions block the syringe port, preventing fluid from entering or leaving the syringe. The pump does not allow syringe moves in those positions with such valves.

For each move command, the argument **n** determines both the destination port and the direction of valve rotation. The default direction is clockwise.

5.2.1. Valve Type Setting

The VersaPump 6 uses a universal valve position encoder that accommodates different valve types. The valve type is selected by sending the "**~Vn**" valve configuration command. The value of the parameter ***n*** is automatically stored into the non-volatile memory (NVM) when the command is received. Once stored, do not set it again unless the valve type is changed. The valve type is stored even when power is removed from the pump. The valve configuration command cannot be placed within a program.

Table 5-7 Valve Type Setting Command Definitions

Query	Description
~V*n*	Set the valve type. If no ***n*** is entered, the current value of the parameter is returned. The factory default is **0** for units ordered without valve drive. (n: 0...12)[1]

n	Valve Type	*n*	Valve Type
0	no valve		
1	3 way non-distribution	2	3 way distribution
3	4 way non-distribution	4	4 way distribution
5	undefined	6	5 way distribution
7	6 way non-distribution	8	6 way distribution
9	8 way non-distribution	10	8 way distribution
		11	12 way distribution
		12	2 way distribution

5.2.2. Valve Position Commands

The valve position commands require the "**R**" (Run) command to be appended to cause immediate execution.

Three-way Non-distribution Commands

These three commands are used with a three-way non-distribution valve only.

- **B** – Moves a three-way standard valve to the bypass position (port A-to-port B).
- **I** – Moves a three-way standard valve to the input position (port A-to-syringe).
- **O** – Moves a three-way standard valve to the output position (port B-to-syringe).

Discrete Valve Position Command

The "**o*n***" command moves the valve to the position selected by ***n***. This command is the preferred command for all valves moves. The values of n must be consistent with the configured valve type. Positive numbers cause counterclockwise rotation as viewed from the front.

(*n*: -12...12, not including 0, where 1= port A, 2= port B, etc., @n)[n/a]

For example, the "**/1o4R**" command moves the valve on pump #1 ("**/1**") clockwise to port 4 (port D) and does it immediately ("**R**").

For example, the "**/3o-2R**" command moves the valve on pump #3 ("**/3**") counterclockwise to port 2 (port B) immediately ("**R**").

Valve Stalls

When a valve fails to turn the commanded amount, a valve stall has occurred. The next valve move command will cause the valve to move to Port A (1) before moving to the new commanded port if the destination port was other than A. The move to port A is a recalibration move.

NOTE: Due to automatic valve retries after a valve motor stall, the time for a valve move can vary significantly. Do NOT use timing loops in controller software to assume a valve move has completed within the nominal time.

When writing software to control the pump, do not assume a valve move will complete in a certain time. In the event of a motor stall and subsequent automatic error recovery attempt, the time required for the valve to complete a valve move can increase substantially beyond the normal time for a move. Instead, query the pump status with a carriage return (hex 0D, decimal 13) character to determine if the pump is busy or the move has finished.

For example, enter "**/1**" and press **Enter** to query the busy status. The reply that comes back is "**/0@**" which indicates that the status is OK and busy. Enter the "**/1**" command again. The reply" **/0`**" is returned, which indicates an OK status and no longer busy.

5.2.3. Valve Queries

Valve queries do not require the "**R**" command in order to execute. They are executed immediately after they are received by the pump. These commands cannot be embedded within a program.

Table 5-8 Valve Query Definitions

Query	Description
?8	Queries the current valve position and returns the ASCII numerical value. (1= port a, 2= port b, etc.)
$	Queries for valve stalls and returns an ASCII number. 0= no error, 2= error
%	Queries the number of valve movements since the last power-up or reset and returns an ASCII number.
~V	Queries the valve type and returns the value of the "**~V*n***" parameter.

5.3. I/O Commands

5.3.1. Output Commands

Output commands set a User Output logic level or send an output byte via the Serial Expansion I/O port. All of these commands requirethe "**R**" command in order to execute immediately.

The sn Command

The "**s*n***" command sends a byte to the expansion port. The value of the ASCII number ***n*** is the base 10 representation of the value of a binary byte. For transmitted bytes, positive logic applies (1 equals the high logic level).

(*n*: 0...255, @n)[n/a]

For example, the "**s85**" command sends the decimal number 85 in binary format. The serial I/O device receives the binary number 01010101 (=85 in decimal-base 10-format). **1** represents a high logic level and a **0** is a low logic level.

The sn,m Command

The "**s*n,m***" sends two bytes to the expansion port, byte ***m*** first and then byte ***n*** second. The values of ***n*** and ***m*** are expressed as the ASCII base 10 representation of the binary bytes.

(n: 0...255, m: 0...255, @n)[n/a]

Table 5-9 sn,m Command Examples

Example Command	Description
s85,129	Sends the decimal number 85 and 129 as binary numbers. The external serial I/O device will receive the binary number 01010101 10000001. The 1 represents a high logic level and a 0 is a low logic level.
s@10,35	Send two numbers. The first is the same first number sent when this command was last used and the second number is "35" (00100011). See Variables, section 5.5, for an explanation of @n usage.

The Un Command

The "**U*n***" command sets User Output **#*n*** to **On** (low logic level) or turns on the expansion I/O port output bit.

Where ***n*** equals:

1...3 equals parallel outputs 1...3

11...18 equals byte 1, bit 1...8 of the expansion port

21...28 equals byte 2, bit 1...8 of the expansion port)[n/a]

Table 5-10 Un Command Examples

Example Command	Description
U2	Turns off (sets to low logic level) User Output #2.
U25	Turns on bit 5 in byte #2 of the expansion I/O.

The Un Command

The "**u*n***" command turns the user parallel output ***n*** to Off (open-circuited) or turns off the expansion I/O port output bit.

Where ***n*** equals:

1...3 equals parallel output 1...3

11...18 equals serial byte 1, bit 1...8 of the expansion port

21...28 equals serial byte 2, bit 1...8 of the expansion port)[n/a]

A special syntax is available for controlling the output while the pump is executing other commands or a program. This allows immediate, real-time control of the outputs. The syntax is a variation of the preceding output commands.

The syntax is:

U#*n* Turns On (set low) an output immediately.
u#*n* Turns Off (set high) an output immediately.

These commands use the same values for ***n*** as the "**U*n***" and "**u*n***" commands.

5.3.2. Input Query Commands

Input query commands are sent from a host controller and request a status reply from the pump. These commands are executed when they are received by the pump and do not require an "**R**" command. These commands cannot be embedded in a program string.

Table 5-11 Input Query Command Definitions

Command	Description
?4	Queries the User Input **#1** status and sends the value to the host as an ASCII. A **1** represents true (low logic input level) and an ASCII **0** represents false (high logic input level). For example, the "**/5?4**" command queries the User Input #1 status on pump #5. The reply of "**/0`1**" indicates the following: **/0** = host address (fixed assignment) **`** = status is "not busy" and "no errors" **1** = input is "true" (low input logic level)
?5	Queries the User Input #2 status and sends the value to the host (see the "**?4**" command above).
?6	Queries the User Input #3 status and sends the value to the host (see the "**?4**" command above).
?7	Queries the analog input value at the Digital Voltmeter input and sends the value to the host controller as an ASCII base 10 number. The voltage equals the number multiplied by 0.02 volts. For example, the "**/1?7**" command queries the Digital Voltmeter input on pump #1. The reply of **/0`157** indicates the following: **/0** = the host address (fixed assignment) **`** = the status is "not busy" and "no errors" **157** = the 157 x 0.02 = 3.14 Volts
?10	Queries the value of the first byte received from the expansion I/O port input. An input byte is read (MSB first), and the numerical value of the first input byte is reported in a base 10 ASCII format. The value uses a negative logic convention (low level = 1, high level = 0). In 1-byte mode, this is the only byte. In 2-byte mode, this is the first of two bytes. For example, the inputs for the first byte are 11001001 (201 decimal). The "**/2?10**" command queries pump #2 expansion input, byte #1. The reply of "**/0`201**" indicates the following: **/0** = the host address (fixed assignment) **`** = the status in "not busy" and "no errors" **201** = the decimal = binary 11001001
?20	Queries the value of the second byte received from the expansion I/O port input in 2-byte mode. Two inputs are shifted in (MSB first) and the numerical of the second input byte is reported in a base 10 ASCII format. See the "**?10**" section above for an example.
?*n*	Queries the state of the expansion I/O port input bit designated by ***n***. A byte is read (MSB first) and the state of the designated bit is reported as an ASCII value of **0** if the bit is false (high input logic level) or an ASCII value of 1 if true (low input logic level). (*n*: 11...18 bits 1...8 in expansion input byte #1 21...28 bits 1...8 in expansion input byte #2)

5.3.3. Input Test and Jump Commands

The value of an input can be checked and its status can be used to cause a program to change the path of the instructions to be executed. This is called a conditional jump. The general format is that if the input is true then jump to the place marked by the program label.

These commands are used to control the way a program executes, depending upon the state of an input variable. The commands are intended to be embedded within a program string and not to be executed alone.

The Digital Input Test Command

For the "**i*np***" command if the input level for bit ***n*** is true (low input level) then jump to label ***p***. This checks if a user input pin on the card edge connector is at a low level. There are three user inputs. If an I/O Expansion Board is used, the number of inputs increases by 16, organized as two bytes of eight bits each.

(***n***: 1...3 User input 1....3
11...18 bit 1...8 in Expansion input byte #1
21...28 bit 1...8 in Expansion input byte #2
p: a...z, A...Z) program label

For example the "***i18b***" command means that test bit #8 is in expansion byte #1. If it is a low level, the command begins to execute the instructions at program label "**b**". If it is not at a low level, it continues with the next instruction after this one.

The DVM Input Test Commands

For the "**i<*np***" and "**i>*np***" commands if the analog input (DVM) value is less than ("<") or greater than (">") ***n*** then jump to label ***p***. The input voltage range of 0 to 5V is converted into one of 255 levels. The number ***n*** is the numerical value of the level. The DVM uses a scale of .02 volts per increment.

(*n*: 0...255, @n
p: a...z, A...Z)

For example the "**i<124s**" command tests the Digital Voltmeter input and if the voltage is less than 2.48 V, then it goes to program label "**s**".

(124 = integer part of 2.48 / 0.02.)

5.4. User Program Commands

Individual commands, commands strings, and programs are executed in the pump RAM (temporary) memory. The pump can also store programs in non-volatile memory (NVM). The NVM acts like a solid-state disk drive. For details on the pump's internal memory, see *Program Memory*, section 4.1.

There are three types of program commands:

- → User program storage commands can load, save, run, or erase user programs in the pump memory.
- → Program execution commands are used to stop or start programs.
- → Program control commands determine the order of execution (flow) of a program.

5.4.1. Program Storage Commands

These commands control the storage, retrieval, and erasure of a user program in the non-volatile user program memory. These commands execute when received and cannot be placed within a program. The "**R**" command is not required.

The maximum length per program string (if there is sufficient NVM space) is 390 characters. This is the limit imposed by the size of the input buffer. The total available space for standard NVM is 400+ characters, and for expanded NVM is 8000+ characters. The exact size of available space may vary according to firmware release.

Table 5-12 Program Storage Command Definitions

Command	Description
E*n*	Stores the current command string into NVM as program #n. (*n*: 1...10 for standard NVM, 11...99 for expanded NVM)
e*n*	Erases a stored command string in NVM. (*n*: 1...10 for standard NVM, 11...99 for expanded NVM)
q*n*	Returns a copy of a program stored in NVM. (*n*: 1...10 for standard NVM, 11...99 for expanded NVM)
?9	Queries the number of unused bytes (characters) in standard and expanded NVM. This query returns two numbers: standard available space (400+ maximum) and expanded aailabel space (8000+ maximum).
?19	Queries which program numbers are currently used to store a program and returns a list of the numbers in use, separated by a space between numbers.
?33	Queries the contents of the program RAM buffer. This is the program string executed when an "**R**" or "**X**" command is entered, or saved with an "**E*n***" command.

5.4.2. Program Execution Commands

The external Stop input function is an added feature on Kloehn pumps. Only the "**H**" command can be used within a program. These commands do not require the "**R**" command for immediate execution.

Table 5-13 Program Execution Command Definitions

Command	Description
~A*n*	Enables or disables auto start for a program in NVM. If ***n*** is not zero, the command begins executing the numbered program when power is applied after a reset. If auto start is disabled, a stored program must be started with the "**r*n***" command. (n: 0 = disable, 1...10 = enable, (@n)[0] (See Section 7.2.3)
~A	Queries the auto start state. It returns a 0 if disabled and the auto-start program number if enabled.
H	Halts the executing command string or program. This command is used to create breakpoints in program execution. The "**H**" command is for inclusion within a program string and cannot be used as a command sent externally to a running program. A program stopped by the "**H**" command within the program may resume execution with the command following the "**H**" command if an "**R**" command is subsequently sent.
r*n*	Runs stored program ***#n*** in the NVM. (External command) (*n*: 1...10 for standard NVM, 11...99 for expanded NVM)
j*n*	Runs stored program ***#n*** and then resumes the present program. (*n*: 1...10 for standard NVM, 11...99 for expanded NVM)
R	Runs the command string in RAM. If the command string has been stopped by an "**H**" command, it resumes execution.
T	Terminates the execution of the current command string. An externally sent command only, it is executed when received. A valve move in progress will go to completion when it receives a "**T**" command. If the "**T**" command is used while the syringe is moving above about 2000 sps, the pump may generate a "**Z**" error when it passes up through the Initialize point. **Note:** The "**T**" command may cause the syringe to "loose steps" and generate a "**Z**" error if the "**T**" command is used to stop the syringe motion at a high speed

5.4.3. Program Control Commands

Jumps and Labels

A program jump provides a means to change the order of execution of program commands. The point from which a jump occurs is a *jump* command. Program execution is changed from the location of the jump command to the destination label (p) specified in the jump command.

A jump can be unconditional, which executes every time it is encountered, or it can be conditional. Conditional jumps are "if...then" commands. The jump to a label occurs only if the specific test condition in the command is true.

Table 5-14 Jump and Label Command Definitions

Command	Description
:*p*	Declares the program label ***p*** (case sensitive). (*p*: a...z, A...Z)
J*p*	Jumps unconditionally to program label ***p***. This is the only unconditional jump command. (*p*: a...z, A...Z)
f*np*	If the flag ***#n*** is set (=1), then this command clears it and jumps to label ***p***. This is useful to change the way a program executes if the path has already been done once before. The program jumps the first time this instruction is encountered but not when it is encountered after the first time. A flag is a bit which can be set (turned on), cleared (turned off) or tested (if...then). There are eight flags, numbered 1 through 8. (*n*: 1...8 p: a...z, A...Z)
f-*np*	If the flag is clear (=0), the command jumps to label p. (*n*: 1...8 p: a...z, A...Z)
i*np*	If the input level is true (low input level), the command jumps to label ***p***. This tests if a user input pin on the card edge connector is at a high or a low level. There are three user inputs. If an I/O Expansion Board is used, the number of inputs increases by 16, organized as two bytes with eight bits each. (***n***: 1...3 User input 1....3 11...18 bit 1...8 in Expansion input byte #1 21...28 bit 1...8 in Expansion input byte #2 ***p***: a...z, A...Z)
i<*np*	If the analog input (DVM) value is less than the number ***n***, the command jumps to label ***p***. The input voltage range of 0 to 5 V is converted into one of 255 levels. The number ***n*** is the numerical value of the converted level. This can be found as n= input volts / 0.02, truncated to an integer number. (*n*: 0...255, *p*: a...z, A...Z)
i>*np*	If the analog input (DVM) value is greater than the number ***n***, this command jumps to label ***p***. The input voltage range of 0 to 5 V is converted into one of 255 levels. The number ***n*** is the numerical value of the converted level. This can be found as n= input volts / 0.02, truncated to an integer number. (*n*: 0...255, *p*: a...z, A...Z)
k<*np*	If the software counter is less than ***n***, this command jumps to label ***p***. A software counter is internal to the pump and can be set to a number, added to, subtracted from, and tested. It is useful for counting program events and for the temporary storage of internal variables such as syringes or valve position. (*n*: 0...65535, @n *p*: a...z, A...Z)
k=*np*	If the software counter is equal to ***n***, the command jumps to label ***p***. (*n*: 0...65535, @n *p*: a...z, A...Z)
k>*np*	If the software counter is greater than ***n***, the command jumps to label ***p***. (*n*: 0...65535, @n *p*: a...z, A...Z)
s<*np*	If the expansion input byte has a value less than ***n***, the command jumps to label ***p***. This command reads in the value of the first I/O Expansion input byte and compares the numerical value of the byte against ***n***. (*n*: 0...255 *p*: a...z, A...Z)
s>*np*	If the Expansion input byte is greater than ***n***, the command goes to label ***p***. (*n*: 0...255,@n *p*: a...z, A...Z)
y<*np*	If the syringe position is less than ***n***, the command jumps to label ***p***. This is useful in loops which repeatedly aspirate or dispense until some event occurs. This test can prevent the error that occurs if a move is commanded beyond zero or full-stroke. (*n*: 0...12000 or 0...24000 or 0...48000, @n *p*: a...z, A...Z)
y=*np*	If the syringe position is equal to ***n***, the command jumps to label ***p***. (*n*: 0...12000 or 0...24000 or 0...48000, @n *p*: a...z, A...Z)
y>*np*	If the syringe position is greater than ***n***, the command jumps to label ***p***. (*n*: 0...12000 or 0...24000 or 0...48000, @n *p*: a...z, A...Z)

Repeat Loops

A program *loop* causes a group of commands to repeat. A loop can be constructed from a jump command and a label. This type of loop repeats indefinitely unless a conditional jump is included within the loop to cause an exit from the loop. The repeat command offers a better way when the number of repeats is known.

The repeat command causes a group of instructions to repeat a specific number of times. The syntax is "**g...G*n***". The "**g**" command marks the beginning of the group of commands to be repeated and the "**G*n***" command marks the end of the group. The value ***n*** denotes the number of times the loop is to be repeated.

Table 5-15 Repeat Command Definitions

Command	Description
g	Marks the beginning of a group of commands to be repeated.
G*n*	Marks the end of a repeat string to be repeated n times. (n: 0...32767) [n/a] **Note:** Do NOT use a number greater than 32767.

For example, for the command "**go1P6000o3A0G10**":

g	Indicates the beginning of a loop command.
o1	Moves to the port 1 location.
P6000	Aspirates 6000 steps.
o3	Moves the valve to port C (3= C).
A0	Dispenses all the contents of the syringe (go to zero).
G10	Closes the loop command string and repeats the loop 10 times.

For the "**o1P6000o3A0**" command, the command string between "**g**" and "**G10**" will be repeated ten times.

Time Delays

A *time delay* is a pause in a program. These are useful for timing events such as generating pulses, very slow syringe moves, and event synchronization.

The "**M*n***" command delays (pauses) for ***n*** milliseconds. Internally the command uses a fixed 1 millisecond clock tick to count down the delay. This may cause an inaccuracy for the first millisecond, since it may be counted down as a partial clock tick. To guarantee a minimum delay add one to the count to account for the partial first millisecond. 1000 milliseconds = 1 second.

(*n*: 1...60000)[n/a]

5.5. Variables

A variable is command argument (command value) that permits a command to use a value which is determined at the time the command executes within a program, rather than being set to a fixed value when the program is written. This permits more general programs to be written and stored.

The two types of variables are:

→ General variables are set by the user and are declared before a program is run.

→ Indirect variables use values obtained from hardware inputs or internal pump values.

All variables use the syntax "@***n***", where "@" denotes a variable and "***n***" denotes the source of the variable.

5.5.1. Setting a General Variable

The value of a general variable is set with the syntax "***zn=m***". General variables are referenced in other commands by the "**@1*n***" syntax, where ***n*** is the variable number.

Where:

z denotes a general variable

n is the variable identification number

m is the value assigned to the variable

For example, in the "**z3=4500**" command the general variable is "@**13**" and the value of "**4500**" is assigned to it.

The value of a general variable can be declared at any time prior to the execution of the program using the variable. The value of the variable must be valid for the command in which it is used. If an invalid value is used, an error message results.

Example of a stored program:

The "**z1=45z2=4500z5=12000r7**" command is a typical run-time declaration of a stored program.

Where:

z1=45 Sets general variable @11=45.

z2 Sets general variable @12=4500.

z5 Sets general variable @15=12000.

r7 Runs the stored program #7 in NVM using these variables settings.

The "***zn***" command executes as soon as the pump recognizes the command. Therefore, the "***zn***" command cannot be stored as part of a program. The values of all "***zn***" commands are stored into RAM and are lost if the power is removed from the pump or if the pump is reset. The default value of all "@**1*n***" variables is zero after a reset.

5.5.2. Using a General Variable

The syntax for a general variable is "@**1*n***".

Where:

@ Denotes a variable.

1 Denotes a general variable.

n Denotes which general variable to use.

(*n*: 1...8)

Table 5-16 General Variables and Argument Values

Argument	Variable	Argument	Variable
@11	z1	@15	z5
@12	z2	@16	z6
@13	z3	@17	z7
@14	z4	@18	z8

For example, the "**D@13**" command dispenses an amount equal to the value of the general variable set in the "**z3=*n***" command.

5.5.3. Indirect Variables

Indirect variables are determined by the read-only value of an input or by some internal condition. The variable syntax is "**@*n***", where "**@**" denotes a variable and n denotes the source of its value. The indirect variables are listed in this table:

Table 5-17 Indirect Variable Definitions

Variable	Description
@1	Numerical value of expansion input byte #1, read as a two-digit packed BCD number (0...99)
@2	Numerical value of expansion input byte #2,#1, read as a two-digit packed BCD number (0...99)
@3	Digital Voltmeter input (0...255)
@4	Digital Voltmeter input (two-digit BCD, normalized to 0...99. Normally used for external displays driven from the expansion port.)
@5	Current Software Counter value (0...65535)
@6	Current valve position (1... number of ports for valve type)
@7	Current syringe position (steps, normally used in other commands)
@8	Current syringe position (two-digit BCD, normalized to percent of full stroke, 0...99. Normally used for external displays driven from the expansion port.)
@9	Most recent value of the byte #2 sent with the "***sn,m***" command
@10	Most recent value of the byte #1 sent with the "***sn***" or "***sn,m***" command

5.5.4. List of Commands Using Variables

The commands listed in this section may use variables. The column labeled "Scaled" indicate if the variable is scaled for use with a particular command. Variables which are not scaled are used as the actual numeric value of a command argument. Values which are scaled are used to compute a proportional amount of the argument's range. The proportion is:

Argument value = (variable value/maximum variable) x maximum argument

Example (Not Scaled): **o@3**

Use the number as the value. If the number in "**@3**" were "6", the command would be "**o6**".

Example (Scaled): **V@3**

The value used for the command will be proportional the maximum value of the number. In this case, if the value of "**@3**" were 127, the actual argument would be 4980. This is computed as follows:

Top syringe step rate (used by "**V**" command) is 10000, maximum analog value (**@3**) is 255.

Value= (127/255) x 10000 (The value is to 10000 as 127 is to 255.)

In the preceding example, the DVM input accepts a dc voltage from 0 to 5 volts and converts this voltage to a parameter value. A parameter value may be scaled. Thus, if a potentiometer is used to input a voltage, the potentiometer dial could be calibrated to read from 0 to 100 percent and would then be applicable to any called parameter.

A given variable is not restricted to use by one command or to one instance of a command. However, the value of a variable must be compatible with all commands which use it. The following commands can use a variable in place of a fixed value for ***n***. Variables cannot be used for labels.

NOTE: The commands in Table 5 18 are described in further detail elsewhere in this section. For a full list of the commands found in this manual, see Table 5-1.

Table 5-18 Commands that use Variables

Command	Instruction	Scale
A*n*, a*n*	Move syringe to absolute position	Yes
c*n*	Set stopping speed	Yes
D*n*, d*n*	Dispense, relative to current position	Yes
g...G*n*	Repeat loop	No
i*np*	If digital input bit ***n*** is true, jump to ***p***	No
i>*np*	If analog input >***n***, jump to ***p***	No
i<*np*	If analog input<***n***, jump to ***p***	No
j*n*	Program #***n***, then return	No
K*n*	Set number of backlash steps	Yes
k*n*	Set software counter = ***n***	No
k+*n*	Add n to software counter	No
k-*n*	Subtract ***n*** from software counter	No
k<*np*	If counter < ***n***, then jump to ***p***	No
k=*np*	If counter = ***n***, then jump to ***p***	No
k>*np*	If counter > ***n***, then jump to ***p***	No
L*n*	Set acceleration slope	No
l*n*	Set deceleration slope	No
M*n*	Delay ***n*** milliseconds	Yes
o*n*	Move valve to position ***n***	No
P*n*, *pn*	Aspirate, relative to current position	Yes
S*n*	Set top speed	No
sn	Send expansion byte	No
sn,*m*	Send expansion double byte	No
s<*np*	If serial input<***n***, then jump to ***p***	Yes
s>*np*	If serial input>***n***, then jump to ***p***	Yes
V*n*	Set top speed (steps/second)	Yes
v*n*	Set start speed (steps/second)	Yes
y<*np*	If syringe position<***n***, jump to ***p***	No
y=*np*	If syringe position=***n***, jump to ***p***	No
y>*np*	If syringe position>***n***, jump to ***p***	No
~A*n*	Set auto start program number	No
~B*n*	Set communication baud rate	No

5.6. Configuration Commands

Configuration commands are used to determine the operating parameters of the pump. All configurations commands begin with a tilde ("**~**") and have two forms: the set form and the query form. The set form uses a numerical argument to set the value of a parameter. The query form is the command with no number attached. The query form reports the present value of the parameter.

All configuration parameters are automatically saved into the NVM when they are set. The settings will be remembered even without power for the life of the pump or until they are changed.

Configuration commands execute when they are received and do not require an "**R**" command. They cannot be used within a program. Either upper or lower-case letters may be used.

Table 5-19 Configuration Command Definitions

Command	Description
~A*n*	Select the program to auto-start when the power is turned on. A selection of **0** means no program is selected for an auto-start. If the value is not zero, the number is the program number to be auto-started after power-up or Reset. If no parameter ***n*** is entered, the current value of ***n*** is returned. (*n*: 0...10, @n)[0]
~B*n*	Select the communication baud rate. If no parameter **n** is entered, the current value of ***n*** is returned. (*n*: 0...7)[3] ***n*** **Baud Rate** ***n*** **Baud Rate** 1 38,400 5 2,400 2 19,200 6 1,200 3 9,600 4 4,800
~H*n*	Set the **Set Home** button mode on the faceplate. If no parameter **n** is entered, the current value of **n** is returned. This parameter determines if the front panel Set Home button is working of not working. This is useful for preventing users from resetting the Home position of the syringe. (*n*: 0= the button operation is enabled 1= the button operation is disabled.)[0]
~I*n*	Enable or disable the valve power-up initialization mode. If no parameter ***n*** is entered, the current value of the parameter is returned. When power is first applied to a pump or immediately after a Reset, the valve normally performs a move to home (port A). In systems with several pumps, it may be desired to inhibit this initialization move to prevent a large current demand on the power supply due to all pumps moving at the same time. If the power up move is inhibited, the first valve move command or the first syringe initialize command will cause the valve to perform an initialization move as the first part of the command. (*n*: 0= the move is enabled and will occur 1= the move is inhibited and will not occur)[0]
~L*n*	Set User Input #3 operating mode. If ***n*** is omitted, the current value of the operating mode will be returned. The default is 0. Regardless of the operating mode, the syringe position snapshot feature is active. ***n*** Operating mode ***0*** Normal The input is a normal logic input. ***1*** Limit A syringe dispense will stop when the input goes true (low), and the input can still be read and tested like a normal logic input.
~P*n*	Select the communication protocol. If no parameter ***n*** is entered, the current value of the parameter is returned. (n: 1=DT, 2= OEM0[1]
~S*n*	Select the Expansion I/O mode. This determines if one or two bytes will be sent and received for each I/O operation. If the I/O Expander Board is used, the two-byte mode must be selected. If no parameter ***n*** is entered, the current value of the parameter is returned. The factory default is 1-byte. (*n*: 1=1-byte transfers, 2=2-byte transfers)[1]
~V*n*	See the "**~V*n***" command in *Valve Type Setting*, section 5.2.1, for details.
~Y*n*	See the "**~Y*n***" command in *Initialization Commands*, section 5.1.3, for details.
~Z*n*	See the "**~Z*n***" command in *Initialization Commands*, section 5.1.3, for details.

5.7. Query Commands

Query commands are executed when they are received. They return a value or set of values to the query. A query command can be sent at any time, even if the pump is busy doing something else, and a reply will be sent.

All query commands are executed when they are received and cannot be placed within a program. Query commands do not require an "**R**" command to execute immediately.

Table 5-20 Query Command Definitions

Variable	Description
Q	Returns the status byte. The simplest form of status query is a Carriage Return (hex code 0D). The command /***address*** returns only a single-character ASCII status byte. Example: /**1**'<**CR**>
q*n*	Lists the command string stored into NVM user program #*n* memory. This is used to view programs stored in NVM.
x?	Reports the last error that was trapped by an error trap instruction.
F	Reports the communication buffer status. 1 = command in buffer, 0 = buffer empty
&	Reports the firmware version as <letter><reference P/N>-<ID letter><rev> (i.e., "P54022-R9").
*	Reports the supply voltage in decimal volts, rounded to the nearest 1/10 volt. The value is averaged over not less than 8 readings.
~A	Queries the NVM user program auto start state and returns **0** if disabled. If enabled, reports the program number to auto start.
~B	Queries the communications baud rate and returns the baud number ***n***. See the "**~B*n***" command in *Configuration Commands*, section 5, for an explanation of the return values.
~H	Query the operating mode of the front panel SET HOME button. See the "**~H*n***" command in *Configuration Commands*, section 5, for an explanation of the return values.
~I	Query the mode of the power-up (post Reset) valve initialize move. See the "**~I*n***" command in *Configuration Commands*, section 5, for an explanation of the return values.
~L	Query the User Input #3 operating mode. See the "**~L*n***" command in *Configuration Commands*, section 5, for an explanation of the return values.
~P	Query the communication protocol. The default is the DT protocol.
~S	Query the Expansion I/O operating mode. See the "**~S*n***" command in *Configuration Commands*, section 5, for an explanation of the return values.

For more information on the following query commands, see *Valve Queries* in section 5.2.3, *Input Query Commands* in section 5.3.2, and *Program Storage Commands* in section 5.4.1.

5.8. Error Trapping Commands

Errors can occur during pump operation, in the structure of a user program, during communications, or in the way a command is given. The pump recognizes these errors. Normally, an error causes a program or instruction to halt and generate an error message to be reported in reply to the next received command. This normal response to an error can be redefined by a user program using a trap command.

A trap is an instruction that directs the pump to go to a label in a program if a particular error occurs. The commands following the label then determine what actions will be taken as a result of the error. An exit command marks the end of the error-handling command string (the "handler") and determines what happens next.

A user error handler is composed of these three parts:

- → The label which marks the beginning of the handler
- → The commands which are the body of the handler
- → The exit command which marks the end of the handler

5.8.1. Trap Declarations

A trap instruction takes effect when it is declared in the program. It remains in effect as written unless it is changed afterwards. Thus error traps can be redefined "on the fly" in a program. The syntax for an error trap is "**x*np***".

Where:

- **x** Denotes an exception (trap) instruction.
- ***n*** Denotes the error number to be trapped.
- ***p*** Denotes the program label which starts the error handler routine.

If error #**n** occurs after a trap for error #***n*** is set, the program jumps to the label ***p***. By declaring the same trap with a different label, different error handling routines can be used for the same type of error in different parts of a program.

A trap operates any number of times if the error occurs externally to the error handling routine. If the error recurs while executing the error handler (before the error handler exit), the program terminates with a standard error exit. In general, if an error persists in recurring, it cannot be solved with a trap.

NOTE: Traps can provide graceful recovery of controlled exits from occasional error conditions. A trap cannot fix system problems or overcome serious mechanical difficulties.

5.8.2. Trap Exits

The last instruction of an error handler (exception program) MUST be a trap exit command. Trap exit commands mark the end of an error handler and specify what action the program is to take when exiting the error handler.

The general syntax is "**t*n***", where ***n*** is the error code number.

Where:

- **t** Denotes a trap routine exit.
- ***n*** Specifies the action to be taken after exiting:
  - 1 - Returns program execution to the instruction following the instruction which caused the error.
  - 2 - Restarts the program from the beginning.
  - 3 - Performs a normal error exit with an error message.
  - 4 - Retries the instruction which caused the error)[n/a].

If exit type 4 is used some means must be used to prevent an "endless loop" of error-> handler-> error-> handler.

5.8.3. Error Trap Query

The pump can be queried at any time to report the last error encountered by a trap. The syntax is "**x?**" for the command. This command executes when received and does not require "**R**" command. Do not place this command within a program.

5.9. Miscellaneous Commands

5.9.1. Software Counters

The pump contains eight software counters. A software counter is a register in the RAM which can hold an integer number. Simple arithmetic operations and tests can be done with counters. These counters are useful for holding numbers for other uses, for counting program cycles, for tracking external events, or for doing simple arithmetic on program values.

The counters are organized as one active and eight exchangeable memory locations. All operations are performed on the active counter. To use one of the other counter memories, that memory must be exchanged with the active counter. The value of the active counter is then placed in the memory location and the prior content of the same memory location is placed into the active counter.

The symbol for the counters is the command "**k**". The counter instructions require the "**R**" command in order to execute immediately. The test-and-jump instructions must be used within a program.

Table 5-21 Software Counter Command Definitions

Command	Description
k	Queries the current value of the active counter.
k*n*	Sets the active counter equal to the number ***n***.
k+*n*	Adds the number ***n*** to the active counter.
k-*n*	Subtracts the number ***n*** from the active counter.
k^*n*	Exchanges the contents of counter memory ***n*** with an active counter.
k<*np*	If the active software counter is less than ***n***, this command jumps to label ***p***. A software counter is internal to the pump and can be set to a number, added to, subtracted from, and tested. It is useful for counting program events and for the temporary storage of internal variables such as syringe or valve position. (*n*: 0...65535, @n *p*: a...z, A...Z)
k=*np*	If the active software counter is equal to ***n***, the command jumps to label ***p***. (*n*: 0...65535, @n *p*: a...z, A...Z)
k>*np*	If the active software counter is greater than ***n***, the command goes to label ***p***. (n: 0...65535, @n p: a...z, A...Z) **Example:** counter 1 (#1) =13, counter 3 (#3) = 45, active counter (ac) = 122 Increment #3 **k^3** #1 = 13 #3 = 122 ac = 45 **k+1** #1 = 13 #3 = 122 ac = 46 **k^3** #1 = 13 #3 = 46 ac = 122 **Example:** Count cycles and jump when a limit is reached: **k+1** Increment the active counter **k>250A** If the count exceeds 250, jump to program label "**A**"

5.9.2. Flag

Flags are software switches which can be set (turned on), cleared (turned off), and tested. Flags are used to indicate the status of something or to change the way something is done after the first time.

There are six general-purpose flags (f1 through f6) and three special-purpose flags (f7 through f9). The flag instructions require the "**R**" command in order to execute immediately. The test-and-jump instructions must be used within a program.

Table 5-22 Flag Command Definitions

Command	Description
f*n*+	Sets flag ***n***. (*n*: 1...9)[n/a]
f*n*-	Clears flag **n**. (*n*: 1...9)[n/a]
f*n*?	Queries the status of flag **n** and returns a **0** if cleared or a **1** if set. (*n*: 1...9)[n/a]
f*np*	Tests the status of flag ***n***. If it is set, then the command clears it and jumps to program label ***p***. This is useful for altering the way something is done the first time this instruction is encountered. The first time will see the flag set and subsequent times will see it cleared. (*n*: 1...9)[n/a]
f-*np*	Tests the status of flag ***n***. If it is clear, then the command jumps to program label ***p***. This is useful for altering the way something is done after the first time it is encountered. (*n*: 1...9)[n/a]

5.9.3. Set Home Button Control

The Set Home button on the front panel can be inhibited, preventing it from being inadvertently activated by a user. There are two ways this can be done, one of which is a long-term control and one of which is intended to be dynamically set "on-the-fly".

When the button is enabled, pressing the button launches the "**W5**" command function, which sets the current syringe position as the zero, or top-of-stroke position and stores the result into non-volatile memory. Only do this operation when the valve, syringe, or washer has been changed or when the top-of-stroke position needs to be changed.

When the button is disabled, pressing the button has no effect. The four ways of controlling the activation of the Set Home button are:

Table 5-23 Set Home Button Command Definitions

Command	Description
~H*n*	Sets the Set Home button mode. If no ***n*** parameter is entered the current value is returned. Use this as a long-term enable or inhibit. The number of times the Set Home function can be done is limited to 10,000. (n: 0 = the button operation is enabled. 1 = the button operation is disabled.) [0]
f9+	Inhibits the Set Home button operation. This command operates in RAM (temporary memory) and is effective as long as power is applied. Use this command to inhibit button operation "on-the-fly".
f9-	Enables the Set Home button operation. This command operates in RAM (temporary memory) and is effective as long as power is applied. Use this command to enable button operation "on-the-fly".
f9?	Queries the status of the button flag f9. The flag reports 1 if disabled or 0 if enabled. The "**~H*n***" and f9 flag interact. If the "**~H**" is set to inhibit, the button will be inhibited regardless of the state of the f9 flag. One useful application of the flags is to set f9 when the system initializes and clear it only when a technician enters an access code into a controller or activates a User Input to service the pump.

5.9.4. External Syringe Motion Limit Input

For applications in which a dispense must be terminated by some external event or condition, configure the User Input #3 an external limit input. Such situations can include dispensing to a fixed level, dispensing to a pH, or creating an external safety stop button.

The normal operating mode for User Input #3 is a logic input. Use the "**~L*n***" pump configuration command to set User Input #3 into the "limit" mode. In the limit mode, a true (low) input halts a syringe move in the dispense direction, but has no effect in the aspirate direction. The behavior of the external limit input can be modified by the action of either flag #7 or flag #8.

Once stopped, the syringe cannot be moved further in the dispense direction unless the Limit input changes to an Off (high) condition or unless flag #7 is set. Moves in the aspirate direction can still be made at any time. If flag #7 is set, then a dispense can be initiated against an active limit input. When the move begins, the flag is automatically cleared so that the next limit input transition from high-to-low causes the syringe to stop again. This is useful in applications where successive dispenses are made under the control of an external signal, such as filling containers using a fill-until-signal condition.

For some applications, it is desired to stop a dispense after the second high-to-low limit input transition. This is accomplished by setting flag #8. This limit action is useful for titrations where the transition is measured optically and a pair of pulses is generated through the transition region. If this feature is used in conjunction with the "snapshot" feature, fully-automatic titrations can be done.

Even in Limit mode, the User Input #3 can still be used for the normal logic functions of reporting User Input #3 status to a host and for "test-and-branch" decisions within a program.

5.9.5. Motor Power Control

The syringe and valve motors can be individually turned on and off by command.

Syringe Motor

The syringe motor normally idles at half-power to conserve energy and reduce idle motor heating. The syringe motor automatically goes to full power at the beginning of a move and returns to half-power after the move completes.

Valve Motor

The valve motor is normally off. The valve motor automatically turns "on" at the beginning of a valve move and turns "off" at the end of a valve move. There are no rotary forces acting on a valve and the combination of the internal friction in a valve and the motor detent torque are sufficient to hold the valve in place between moves.

The mn Command

Use the "**m*n***" command to set the status of a motor, where n denotes the motor and action to take.

n: 0 - Turns off the syringe motor.
1 - Turns on the syringe motor.
2 - Turns off the valve motor.
3 - Turns on the valve motor.

When a move takes place in either the syringe or valve motor, the normal motor power operation for the move overrides the current state of the "**m*n***" command.

Examples:

m0	Turns off the syringe motor.	(motor is off)
A1200	Sends the syringe to position "1200".	(motor is on during and after moving)
m0	Turns off the syringe motor again.	(motor is off)

5.9.6. Repeat Command String

A command string can be repeated in its entirety by sending the string repeat command. The syntax is "**X**". There is no limit to how often this command can be used to repeat the previous command string.

This command does not work for queries and configuration commands.

Example :

o2A12000o-1A0R	Moves the valve to port B, fill the syringe, move the valve to port A, empty the syringe.
X	Does all of the above commands again in the same way.
X	Does all the commands listed above again.

5.9.7. Miscellaneous Queries

Firmware commands change over time. A program string can test for particular release levels to ensure new features are present. The "**?32**" query returns the current firmware release level.

Table 5-24 Syringe Query Definitions

Query	Description
?32	Queries the firmware modification level ("-R9" for firmware release R9).

6 Status and Error Messages

A status byte is returned after about 12 milliseconds following receipt of the carriage return I each command string sent t the unit. Every pump reply contains a status byte immediately following the host address "/0".

Example:

/0@125	**/0**	Host address (fixed).
	@	Status byte (ok, busy).
	125	A value return in response to a query.

Each status byte has two forms:

Busy	The device is executing a command of program.
Ready	The device is ready to receive another command.

Table 6 1 Status Messages

ASCII		Error	Decima		Binary	Status	Description
Busy	**Ready**	**#**	**Busy**	**Ready**	**76543210**	**–**	**–**
@	`	0	64	96	01x00000	No error	The device is operating normally. No error condition has been detected.
A	a	1	65	97	01x00001	Syringe failed to initialize	An attempt to initialize the syringe has failed. No syringe move command other than an initialize command will be valid until initialization is complete. This is usually the result of a syringe overload. Check the fluid paths for restrictions or obstructions. Check the valve ports for alignment. If a reduced speed succeeds, try a shorter fluid path or larger diameter fluid path.
B	b	2	66	98	01x00010	Invalid command	A command just sent was not recognized as valid. A character was sent that is not part of the command set. A typing error may have occurred. For example, "**N1000**" is not valid because "**N**" is not a legal command.
C	c	3	67	99	01x00011	Invalid argument	The number sent with a command is not valid. The command itself was valid, but the value sent with it was out of the allowed range of values for that command. For example "A55000" has a value 55000, which is not within the range of available values. This most frequently occurs when a series of relative dispenses is commanded, and the last dispense command exceeds the remaining volume.
D	d	4	68	100	01x00100	Communication error	The checksum or sequence number was incorrect (OEM protocol only). Retransmit the message with a new sequence number. The sequence number is adjusted each time a command is repeated to prevent a repeated command from being executed more than once.
E	e	5	69	101	01x00101	Invalid **R** command	The "**R**" (run) command was sent with a command which does not require it.
F	f	6	70	102	01x00111	Supply voltage too low	The device supply voltage is too low. The conditions may by a low power supply voltage or voltage transients on the power supply wiring near the pump.
G	g	7	71	103	01x00111	Device not initialized	A move command was sent while the device was in an un-initialized state. The device must be initialized before a move command will be accepted. An initialized state results after a power-up, a reset, or a syringe overload error.
H	h	8	72	104	01x01000	Program in progress	A program or command is executing. A new command (other than queries, "**V*n***" and "**T**") cannot be sent until the present command completes.

ASCII		Error	Decima		Binary	Status	Description
Busy	Ready	#	Busy	Ready	76543210	–	–
I	i	9	73	105	01x01001	Syringe overload	The load on the syringe drive axis was too great. The syringe motor stalled. A stall does no damage to the drive system there are several possibilities: 1) A fluid path is constricted or blocked. Check for kinks in the tubing valve washers which may have a shrunken hole, other valves sticking, and other sources of constriction. 2) The syringe velocity is too high. Back pressure increases with the square of velocity. Available thrust force also decreases with velocity. Reduce the syringe top speed using the "**V*n***" or "**S*n***" commands. 3) The acceleration is too high. High acceleration places a power demand upon the syringe motor in addition to the friction and fluid back pressure demands. Decrease the acceleration value using the "**L*n***" command. 4) The inside diameter of the flow path is too small for the fluid flow rate. A larger diameter, shorter path, or lower velocity may be required.
J	j	10	74	106	01x01010	Valve overload	The valve motor failed to complete a commanded move. Some possibilities are: • The valve is wearing out and needs replacement. • There is particle contamination in the valve. Particles may come from another place and be transported into the valve or may result from crystal deposits in the valve. • The chemistry results in swelling or softening of the valve materials, resulting in binding. Check the chemical compatibility. • The fluid or operating environment is too hot. This results in the swelling and binding of the valve. • There is misalignment of the valve drive and the valve stem. This typically results in binding at one point in the valve rotation. • A fitting is screwed in too tight, distorting the valve and causing binding.
K	k	11	75	107	01x01011	Syringe move not allowed	A syringe move command was not allowed because the valve is in a bypass position (syringe port blocked), or because the supply voltage is too low.
L	l	12	76	108	01x01100	Cannot move against limit	The User Input # 3 limit input is active. The syringe cannot dispense against an active limit input.
M	m	13	77	109	01x01101	Expanded NVM failed	The expanded non-volatile memory (NVM) has failed. Erasing a program and restoring it can sometimes overcome this error.
O	o	15	79	111	01x01111	Command buffer overflow	A command was sent while another was executing. The last command which was sent was not executed and was discarded.
P	p	16	80	112	01x10000	Use for 3-way valve only	An instruction was sent which applies only to a three-way non-distribution valve only ("**O**","**I**","**B**") with the valve type set other than a three-way non-distribution.
Q	q	17	81	113	01x10001	Loop nested too deep	There are too many nested loops in the program.
R	r	18	82	114	01x10010	Program label not found	A program label called by a jump instruction was not found in the program. Labels are case-sensitive. A label may have been changed or deleted when editing a program.
S	s	19	83	115	01x10011	End of program not found	A program stored in the non-volatile memory (NVM) does not have the required end-of-program indicator. The NVM may have been corrupted.
T	t	20	84	116	01x10100	Out of program space	There is not enough program memory to hold the entire program.
U	u	21	85	117	01x10101	Home not set	The Home (zero) position has not been set The syringe axis requires calibration.
V	v	22	86	118	01x10110	Too many program calls	A called program has called another program. Only one program can be called at the same time.
W	w	23	87	119	01x10111	Program not found	The program called or commanded to execute cannot be found in the memory.

ASCII		Error	Decima		Binary	Status	Description
Busy	Ready	#	Busy	Ready	76543210	–	–
X	x	24	88	120	01x11000	Valve position error	The valve position is not valid. This error occurs when the valve position code wheel sensor fails to see a slot after all valve motion has ceased. Some possible causes are: • The valve failed to complete the preceding move. • The valve position has been displaced since the last move completed. • Contamination has blocked either the optical sensor or a sensor disk slot.
Y	y	25	89	121	01x11001	Syringe position corrupted	The current computed syringe location is in error. The position value is outside the range of acceptable values. The syringe position memory has likely been corrupted.
Z	z	26	90	122	01x11010	Syringe may go past home	The syringe might try to move up past the Home (zero) positions. This message is generated whenever a check of the computed syringe position exhibits an out-of-tolerance error. Syringe crashes are avoided by this means. When the syringe position is calibrated, the distance from the Initialize position to the top-of-stroke (zero) is stored into NVM. Each time the syringe passes up through the Initialize point, the computed position is compared to the stored value. If the two values do not match within a given error band, the syringe is stopped and the error message is generated. The two most common sources of this error are: • Having the home set at the Initialize point. • Having lost (not count) steps during a move. Steps can be lost when the "**T**" command is used to stop a syringe which is moving at speeds above 2000 steps per second or when a syringe overload occurs.

7 Sample QBasic Communications Program

This section presents two typical driver functions: send and receive a string, and test for busy status. These functions have been written and tested in the QBasic syntax as supplied with DOS 6.x[TM]. The structures can be easily converted to Visual Basic of ANSI C. The "GOTO" statements can be used, or a more structured style can be employed.

The communications channel for the syringe drive is opened by the statement:

```
OPEN "com@: 9600, N, 8, 1, CS0, DS0, CD0, RS" FOR RANDOM AS # 1
```

When the program ends, the statement below must be used to close the channel.

```
CLOSE # 1
```

The preceding two QBasic statements will need to be developed as function calls in ANSI C or Visual Basic if an equivalent library function is not available. Windows programs should use the Windows Applications Programming Interface (AP) for I/O handling whenever possible. The driver code presented in this manual is intended as a sample of driver structures to facilitate the development of user drivers.

7.1. Send and Receive a String

The following QBasic code accepts a string and sends it to the syringe drive. It waits a short interval (→15ms) and receives the response string. The response string is then parsed for the status byte if an error status has been returned, the error is identified and an error message string is generated the string value "Pump$" contains the string to be sent , including the pump address number the response, if any, is returned in the string variable "PumpIn$". Error messages are printed from within the module.

```
'****************** UTILITY MODULE: Get Pump subroutine ******************

Send command string, gets reply, parses  reply
'         This is used by other subroutines for pumpIn$ = response string returned
'         Pump$ = command string to send, pumpIn$ = response string returned
'         "eflag " indicates  error status :o = no error , 1 = error status; set to 0 before'
          Entry into module.

GetPump:

'*********send pump command and get response *********
      PRINT # 1, pump$;CHR$(13)                        send command string with <CR>
      For n = 0 TO 1000: NEXT n                        'delay to receive entire reply
      In$ = INPUT$(LOC(LOC), #1)                       get reply string from pump
      IF MID$(In$,1,1)<> "/" THEN
                PRINT "COMM ERROR: no response from pump"
                Eflag = 1                              "set error flag ON
                GOTO gp2                               "exit module
      END IF
'**********parse reply for errors***********
     Status$ = ""                                                  'get status byte value
     PumpErr$ = ""                                                 'clear error message string
     IF ASC(STATUS$) <> AND ASC(status$)<> 64 THEN                 'IF NOT "OK" status
                SELECT CASE status$
                      CASE "a"  "A " : PumpErr$ = Syringe not initialized"
                      CASE "b", "B" :  PumpErr$ = Invalid command"
                      CASE "c", "C" :  PumpErr$ = Invalid operand "
                      CASE "d", "D":   PumpErr$ = Communication error"
                      CASE "g" , "G":  PumpErr$ = Device not initialized"
                      CASE " i ", "I " :  PumpErr$ = Syringe overload"
                      CASE " j " , "J" : Pump Err$ = Valve overload"
                      CASE " k", "K" : PumpErr$ = "No syringe move in valve bypass"
                      CASE "o" , "O" : PumpErr$ = " Command buffer full"
                END SELECT
                PRINT "PUMP ERROR :";status$; = "PumpErr$
                Eflag = 1                              'set error flag to ON

      END IF

''************ extract any response string *******************

      n = LEN (In$)  'begin from last character in reply
gp1:
      IF MID$(In$,n, 1) <> CHR$(3) THEN 'ETX character ?
      n = n - 1                                        no, check next character
                GOTO gp1
      END IF
      n = n - 4                                        adjust for length of response
      pumpIn$ = MID$ ( In$, 4, n)                      'DELETE "/O", status byte, & overhead
gp2:
      RETURN
```

When writing as ANSI C version of the preceding drivers, the statement "`PRINT #1, Pump$`" in the Listing below must be replaced with a function call which handles the character – by – character details of string transmission. Also , the "`In$ = INPUT$ = =INPUT$(LOC(1), #1)`" statement requires a function call to handle the reception and construction of the reply string .

7.2. Check for Busy Status

This module checks to see if the syringe drive is busy or is ready for another command. It returns to the calling routine when the status is "ready".

```
'************** UTILITY MODULE: Check Pump for Busy Status *************
'  wait for pump to return a "not busy" status
'  no entry parameters, no return value
PumpBusy:
       Pump$ = "/1Q"                                  ' set up status query string
       GOSUB GetPump                                  'send status query
       IF PumpErr$ <> ""THEN GOTO pb2                 'if pump status error, exit routine
       IF Status$ = "@" THEN GOTO PumpBusy            ' else check until "not busy"
Pb2:
       RETURN

'****************** UTILITY MODULE: GetPump subroutine ***********************
'         send command string, gets reply, parses reply
'         This is used by other subroutines for pump communications
'         Pump$ = command string to send, pumpIn$ = response string returned
'         eflag" indicates error status: 0 = no error, 1 = error status: set to 0 before
 '              Entry  into  module
GetPump:
'**********send pump command and get response**************
       PRINT #1, pump$: CHR$(13)                      'send command string with <CR>
       FOR n = o TO 1000: NEXT n                      'delay to receive entire reply
       In$ = INPUT$(LOC(1), #1)                       'get reply string from pump
       IF MID$(In$, 1,) <>"/" THEN
                  Print "COMM ERROR: no response from pump"
                  Eflag = 1                           'send error flag ON
                  GOTO gp2                            'exit module
       END IF

''********** parse reply for errors **************
       Status$ = MID$(In$, 3, 1 )                     'get status byte value
       PumpErr$ = """"                                'clear error message string
       If ASC(status$<> 96 AND ASC (status$) <> 64 THEN ' if not "OK" Status
       SELECT CASE status$
                  CASE "a", "A": PumpErr$ = "Syringe not initialized"
                  CASE "b", "B": Invalid command"
                  CASE "c", "C": PumpErr$ = "Device not initialized"
                  CASE "I", "I"":  PumpErr$ = "Syringe overload"
                  CASE "j", "J":  PumpErr$ = "Valve overload"
                  CASE "k","K": PumpErr$ = "No syringe move in valve bypass"
                  CASE "o", "O": Command buffer full"
       END SELECT
       PRINT "PUMP ERROR: " ; status$; " = " ; PumpErr$
       Eflag = 1                                      'set error flag to ON
       END IF
```

```
'' ************ extract any response string ***********************
      n = LEN(In$)                                  'begin from last character in reply
gp1:
      IF MIS$(In$, n, 1)<> CHR$(3) THEN             ' ETX character ?
                  N = N -1                          ' no , check next character
                  GOTO gp1
      END IF
      N = N -4                                      ' adjust for length of response
      pumpIn$ = MID$(In$, 4, n)                     ' delete "/0", status byte, & overhead
gp2:
      RETURN
```

This module checks to see if the syringe drive is busy or is ready for another command. It returns to the calling routine when the status is "ready".

```
'**************UTULITY MODULE: CHECK Pump for Busy Status *************
'  wait for pump to return a "not busy "status
' no entry parameters, no return value

PumpBusy:
Pump$ = "/1Q"                                       'set up status query string
GOSUB GETPump                                       'send status query
IF PumpErr$<> ""THEN GOTO pb2                       'if pump status error, exit routine
IF status$ = '@" THEN GOTO PumpBusy                 'else check until "not bus
Pb2:
RETURN
```