

# ViewPort

## Local Displays for LabVIEW applications

MODEL VP216L

User Manual Version 1.00



Viewpoint Systems, Inc. does not warrant that the Program will meet Customer's requirements or will operate in the combinations which may be selected by the Customer or that the operation of the Program will be uninterrupted or error free or that all Program defects will be corrected.

VIEWPOINT SYSTEMS, INC. DOES NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS THAT MAY BE OBTAINED BY USING THIS SOFTWARE. ACCORDINGLY, THE SOFTWARE AND ITS DOCUMENTATION ARE SOLD "AS IS" WITHOUT WARRANTY AS TO THEIR PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE PROGRAM IS ASSUMED BY YOU.

NEITHER VIEWPOINT SYSTEMS, INC. NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SUCH AS, BUT NOT LIMITED TO, LOSS OF ANTICIPATED PROFITS OR BENEFITS, RESULTING FROM THE USE OF THE PROGRAM OR ARISING OUT OF ANY BREACH OF ANY WARRANTY. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR THE ABOVE MAY NOT APPLY TO YOU.

LabVIEW<sup>®</sup>, FieldPoint<sup>®</sup>, and Compact FieldPoint are registered trademarks of National Instruments Inc.

All other brand and product names are trademarks or registered trademarks of their respective companies.

Copyright © 2003, Viewpoint Systems, Inc.

All Rights Reserved. Reproduction or adaptation of any part of this documentation beyond that permitted by Section 117 of the 1976 United States Copyright Act without permission of the Copyright owner is unlawful.

Viewpoint Systems Inc.  
800 West Metro Parkway  
Rochester, NY 14623  
June 2003  
VS-VUPRT-216-0100

# Table of Contents

---

Overview .....	1
System Requirements.....	1
Getting Started .....	2
Installing the Software .....	2
Software Updates .....	2
Installing the Hardware.....	2
System Setup Verification .....	2
Removing the software .....	2
Theory of Operation.....	3
Overview .....	3
Main Concepts .....	3
Display Commands:.....	3
Screen:.....	4
Display Manager:.....	4
Display Engine:.....	4
Static Screen:.....	4
Dynamic Screen: .....	4
Creating Applications .....	5
Some preliminaries .....	5
ViewPort “ <i>Hello World!</i> ” Screen .....	5
Creating a simple application.....	5
Introducing the ViewPort Engine .....	6
Simplifying with Dynamic Pages .....	7
Controlling the ViewPort Engine.....	7
Adding a Keypad .....	8
User I/O Screens .....	9
Tips on creating a new application .....	10
VI Reference .....	11
ViewPort 216 Commands .....	12
ViewPort 216 Screen Manager .....	14
ViewPort 216 Screen Engine .....	15
ViewPort 216 Xlat Keypad.....	16
ViewPort 216 Edit Double .....	17
ViewPort 216 Edit List .....	18

Performance Hints.....	19
General Considerations .....	19
Cable issues.....	19
Additional LabVIEW VIs .....	20
ViewPort Error Codes .....	21
Diagnostics.....	22
Specifications .....	23
Dimensions .....	23
Recommended Panel Cutout .....	24
Power Requirements .....	25
Environment.....	25
Physical.....	25
Connectors .....	25
Contacting Us.....	26

# Overview

---

ViewPort displays allow embedded applications to get information to the outside world. The devices are easy to install and even easier to program. The ViewPort LabVIEW library provides all the tools necessary for adding output to your applications.

ViewPort features include:

- **Simple devices** Serial port based displays can be attached to a wide variety of systems. Use ViewPort with FieldPoint, Compact FieldPoint, or PXI based controllers.
- **Code portability** The same ViewPort VIs and examples will run on LabVIEW system with a spare serial port. Code can be developed and tested on a desktop PC and then re-targeted to a FieldPoint controller.
- **Universal Power** ViewPort can work with 7-30 Volt DC sources (same as FieldPoint and Compact FieldPoint)
- **Driver software** drivers and LabVIEW VIs provided
- **Examples** LabVIEW examples show how to make the most of the ViewPort displays

## System Requirements

FieldPoint or Compact FieldPoint controller with available serial port

**OR**

PXI or CompactPCI chassis with available serial port

**OR**

IBM compatible PC with available serial port

LabVIEW 6i (6.1) or higher

# Getting Started

---

## Installing the Software

The ViewPort software is installed through an installation program found on the accompanying CD-ROM. The installation program will run automatically when the CD-ROM is inserted into the CD-ROM drive. If *AutoPlay* has been disabled on your computer, you can manually start the installation program by running the *setup.exe* program found in the CD-ROM's root directory.

## Software Updates

Software updates will be made available through the Viewpoint Systems web site. Check the ViewPort web page at <http://www.ViewpointUSA.com/ViewPort> periodically for the latest driver software and new examples.

## Installing the Hardware

It is important to follow appropriate electrostatic precautions when handling the ViewPort board. The board is shipped in an electrostatic bag. Be sure that you have removed any electrostatic hazard by attaching a grounding wrist strap or touching the computer chassis before removing the ViewPort board from its bag. Save the bag for future use when the ViewPort board is removed from the system.

ViewPort displays require power from a 7-30 Volt DC power supply. Connect the power supply to the screw terminal power connector supplied. Be sure to follow the indicated polarity (see diagram under the connectors section – later in the manual).

## System Setup Verification

One of the powerful features of the LabVIEW environment is that the same code runs on multiple target devices. The ViewPort examples we provide, and ViewPort VIs for applications you write, will run just as well on your desktop as your LabVIEW RT FieldPoint or PXI controller system.

The easiest way to verify proper operation of your ViewPort display is to simply connect to a spare serial port on your desktop PC.

## Removing the software

Please use the *Add/remove Software* applet found in the control panel to remove the ViewPort software. This ensures the complete and proper removal of the software from your system.

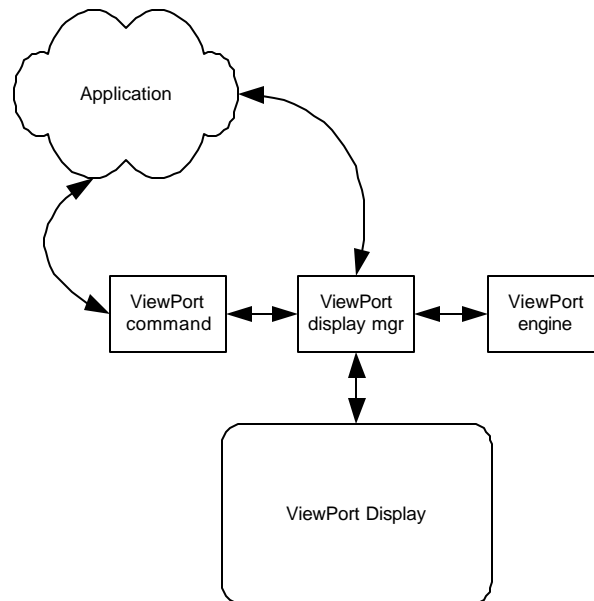
# Theory of Operation

---

## Overview

ViewPort serial displays allow you to add local display to any LabVIEW application with a spare serial port. These displays are especially useful when combined with LabVIEW RT applications deployed on Fieldpoint and Compact FieldPoint systems. The ViewPort software provides the application with everything it needs to create simple, useful, informative displays.

All interaction with the ViewPort display is through the serial port. Command sequences (strings) are built up by the application and transmitted to the display through VIs in the ViewPort library. The ViewPort library consists of low-level and high-level routines that allow you to create ViewPort display screens.



## Main Concepts

### Display Commands:

Each type of display has a number of commands it recognizes. The ViewPort library contains a VI that allows the application to generate these commands easily. These commands include simple actions like setting brightness/contrast or positioning the cursor to more complicated commands such as drawing a bar graph.

**Screen:**

A ViewPort *screen* is a predefined page of display information that can be displayed at will. A typical application will consist of a number of screens of information that is displayed sequentially on the ViewPort display.

**Display Manager:**

The ViewPort *display manager* is the main mechanism for interfacing with the ViewPort displays. The manager has actions that:

- Provides the path for all serial communication with the display/keypad.
- Caches screens for display by the display engine.
- Provides routines to control the actions of the display engine.

**Display Engine:**

The ViewPort *display engine* is a VI that runs independent of the main application and is responsible for automatically sequencing screens on the ViewPort display. The engine cycles through the screens according to its settings. The application can switch the Display Engine to manual mode and take control over the ViewPort Display at any time. This is especially useful when the system needs to alert an operator of a problem or particular system condition.

**Static Screen:**

A *static screen* is a page of information that the application creates and doesn't change. A static page is presented to the Display Manager and it will hold a copy of the screen (cache) for replay whenever it is called for. The application can overwrite a *static screen* at any time to freshen its content. This is a *push* type of operation.

**Dynamic Screen:**

A *dynamic screen* is a page whose content changes frequently. A separate VI is responsible for formatting the information for the display. The application then registers this VI with the *display manager* as the source for the content of a particular screen. The *display engine* will then call the VI anytime it needs to display that particular screen on the ViewPort display. The VI would gather any information needed for that screen from other sources in the application. This is a *pull* type of operation.



# Creating Applications

---

## Some preliminaries

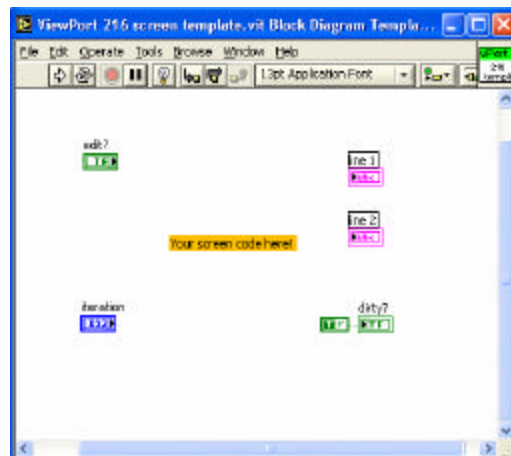
The following LabVIEW examples assume that you have successfully connected and powered your ViewPort display.

The manual assumes that you have a good working of Labview and Labview RT applications.

We recommend developing the framework for the ViewPort application under windows to allow easier access to the software. When the framework is working as expected any application specific interfacing can be done and then downloaded in the RT system for final verification.

## ViewPort “Hello World!” Screen

The ViewPort driver works by supplying a screen VI. The screen VI is generated from the screen template shown below.

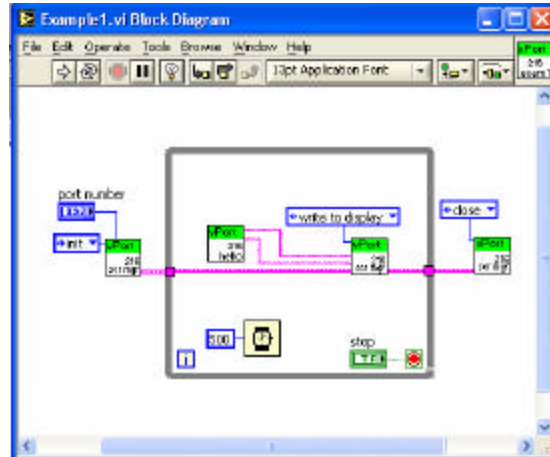


All screens take on the form of this template. The screens themselves are actually very simple; they have an output for the text to display and a control to when to update the screen. Looking at the screen VI template you can see that a call to the command VI with the “Home” action is called to put the display cursor at the home position. Also since the iteration is zero the flag “dirty” is raised to show the Screen manager that the screen has changed.

## Creating a simple application

The first example is a very basic look into using a screen template in a very simple application using the screen manager.

The first call to the screen manager is to initialize the manager. All ViewPort Applications must be initialized. This application loops while writing directly to the display.

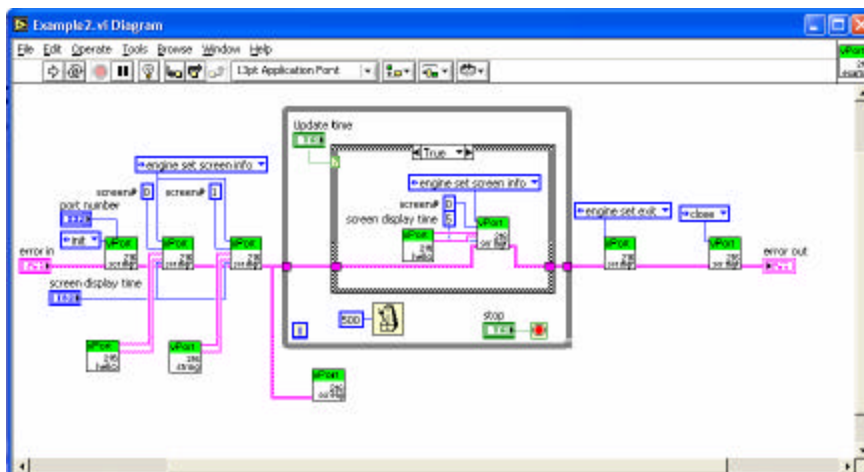


## Introducing the ViewPort Engine

In the previous example the code just used one screen and simply just sent the information directly to the screen. But what if you want multiple screens? How is this handled?

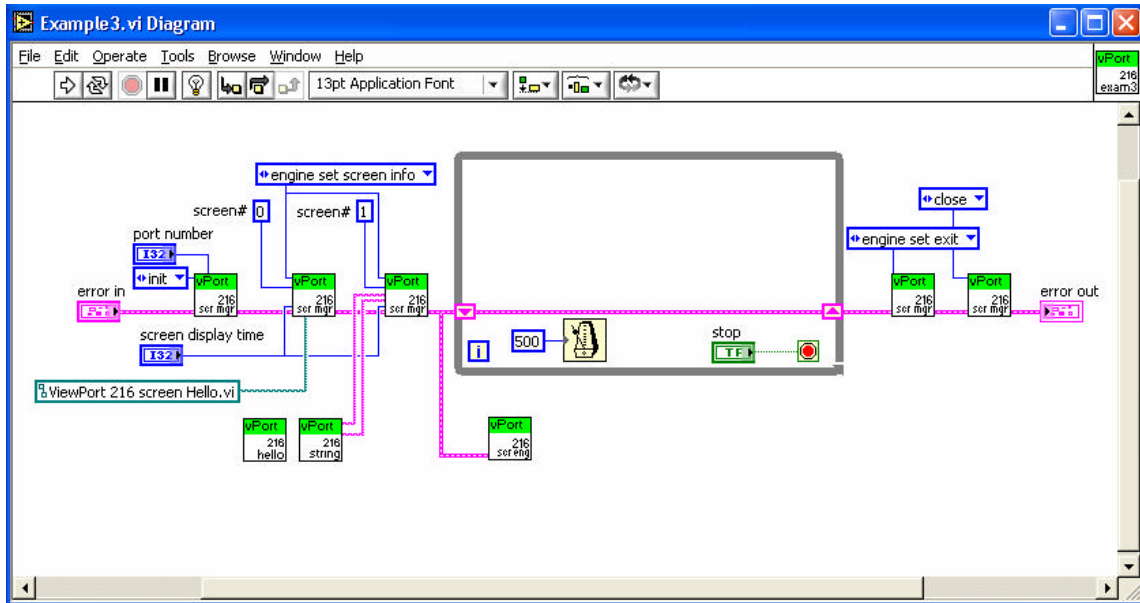
The ViewPort Screen Engine simplifies the use of multiple screens. The engine will automatically update screen information and settings by calling the screen manager. The Screen Engine can be thought of as the behind the scenes application that insures all screens get updated in a regular timeframe.

The application shown in example2.vi uses the screen engine with multiple screens and updates screen zero at an independent rate.



## Simplifying with Dynamic Pages

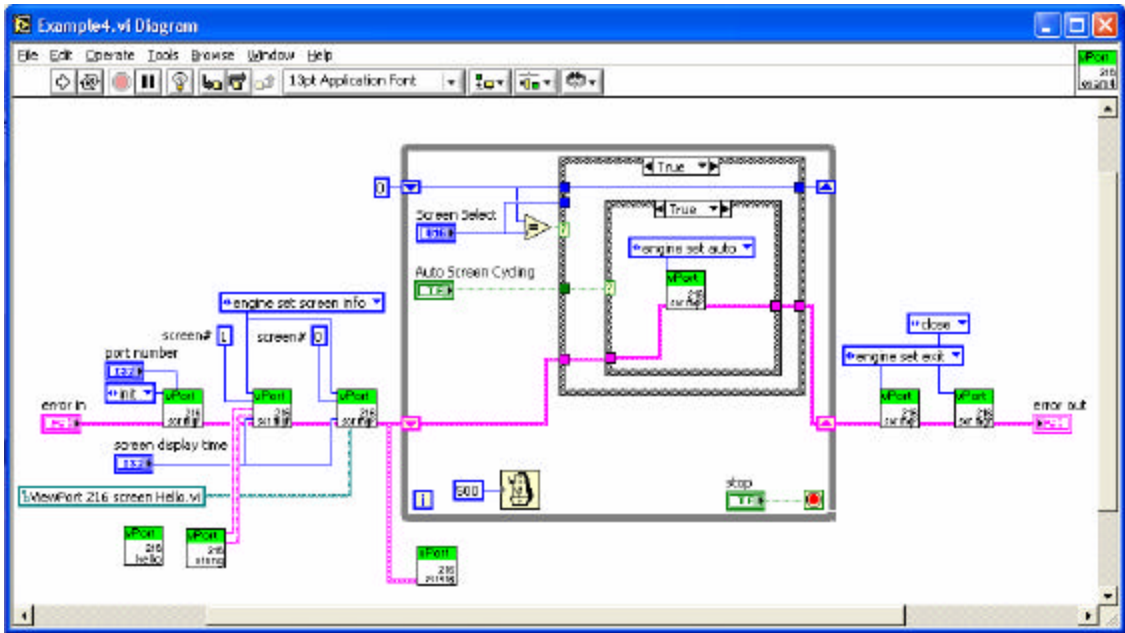
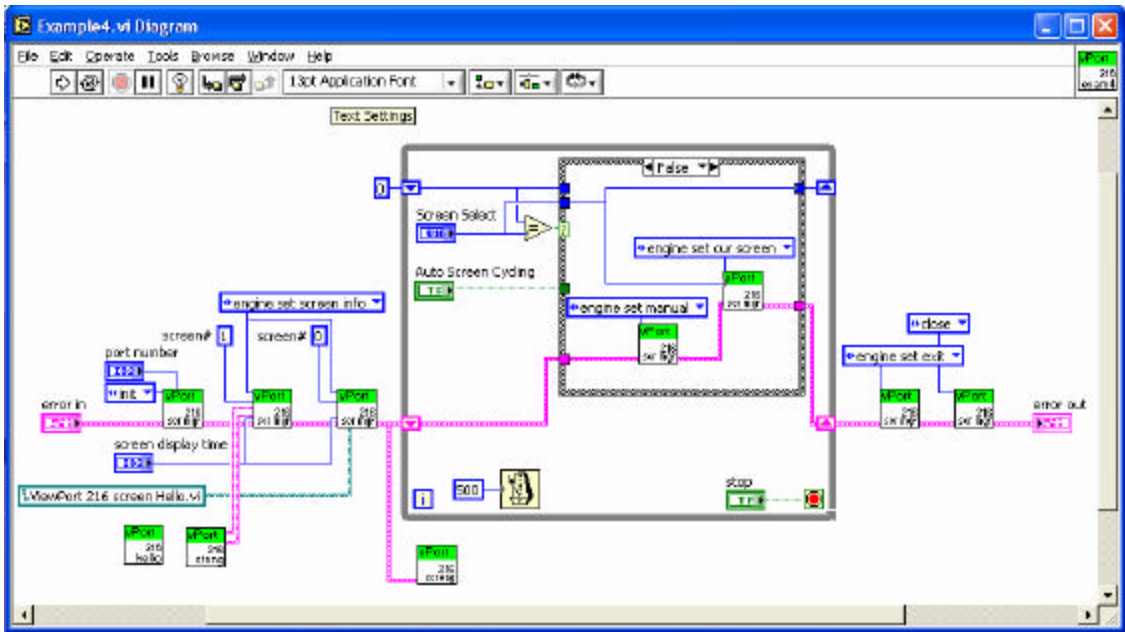
The main loop below was simplified by using dynamic calls to the hello screen VI. The concept of dynamic pages can be confusing but they allow the screen engine to call the VI without programming the application to call the screen VI to update it. The example below, Example3.VI works functionally the same as the code in example2.



## Controlling the ViewPort Engine

The ViewPort Engine doesn't always need to automatically update the screens, the engine can be placed in a manual mode allowing the user to force the display order, change the current screen or modify the timing of screens. From manual mode the Application can also enable or disable a screen on the display.

Example4.VI forces the engine in manual mode (when Auto Screen Cycling is false) allowing the application to control which screen is the currently being displayed.



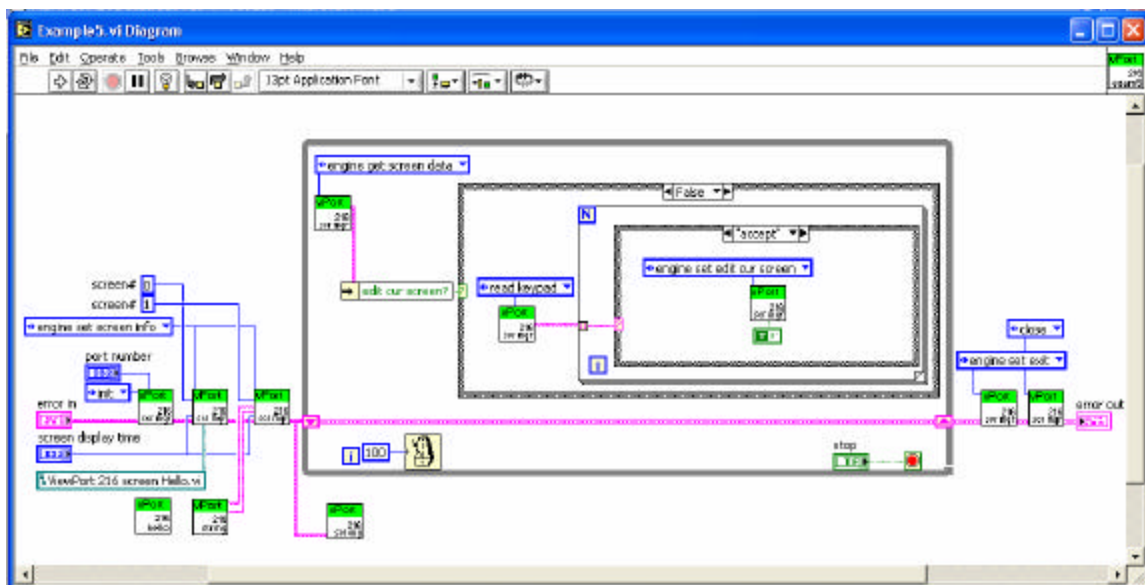
## Recognizing Keypad input

ViewPorts included with keypads may allow for user input. The screen engine looks for keys being pushed and stores them until the application can request the manager for the latest commands available.

The main loop must now look for the keypad sequences and handle these events through an event handler structure as shown below. Any keypad key that is to be handled needs a case in which the required sequence or action occurs based on that particular keypad event.

The example below looks for the up and down arrow keys to manually override the engine's cycling of the screens. The Handler also looks at the 0 and 1 characters to force the display of screens 0 and 1 respectively. Once the engine is in the manual mode the escape key is used to return the engine back to operating in the automatic mode.

This simple keypad handler can be used in a variety of applications further extending the display capabilities of a ViewPort.



## User I/O Screens

To further extend the functionality of a ViewPort, the screen VI's have the capability to supply more complex I/O interactions by the use of the Edit flag in the screen VI's. The concept of using the edit control enables anything from selecting which screens to view to allowing input from the keypad to do some physical thing to the system. The idea here is that the event structure shown in example 5 can be further extended per screen. The event handler looks for the enter key to be pressed. When the enter key is pressed and if the current screen allows editing, the main loop doesn't allow the event handler in the application to run, instead the screen event handler (user supplied) overrides this handler and does something different than the default behavior.

The example screen 6.vi installs an edit handler that simply allows the user to move between three different string lines to be output on the screen until enter or cancel are pressed, saving or restoring the string used for that screens display.

## **Tips on creating a new application**



Some of the simplest uses of the ViewPort is for just sending messages out for either debugging or user status. A good percentage of applications will fall into the example 2 and 3 scope. The data screen will be static but the data itself may or may not be updated.

The ViewPort Screen template should always be used as a starting point. The screens themselves can be pretty much debugged on a standalone basis without the use of any ViewPort calls. Once the screens operate as expected, the screen can be easily added into the top ViewPort application, keeping in mind whether or not the screen is dynamic (appear to change while being displayed) or static (doesn't change as being displayed) and having the screen manger call the VI dynamically or statically as required.

## VI Reference

---

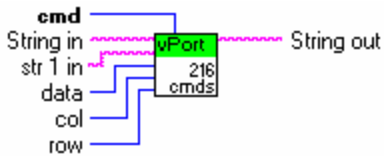
Some of the ViewPort VIs share common parameters. These parameters are defined once below, and are omitted from the individual VI descriptions.

	<p><b>error in (no error)</b> error in is a cluster that describes the error status before this VI executes. If error in indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.</p>
	<p><b>error out</b> error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, error out is the same as error in. Otherwise, error out shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using error in and error out clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.</p>








## ViewPort 216 Commands

This VI appends a ViewPort display command to the string being built up. Each type of screen has different commands available to it. Commands control a variety of aspects of the display --- from cursor control to bar graph commands. See the table below for more information on each command available.

### Connector Pane



### Controls and Indicators

-  **String in** The input string. The command will be appended to this string and passed out the string out terminal.
-  **cmd** Specifies the command to be appended to the input string. These commands are particular to the type of ViewPort display. See the table below for the particulars.
-  **data** Parameters used in certain commands. See the table below for the particulars.
-  **str 1 in** Parameters used in certain commands. See the table below for the particulars.
-  **col** Parameters used in certain commands. See the table below for the particulars.
-  **row** Parameters used in certain commands. See the table below for the particulars.
-  **String out** The resultant string combining the input string and the command indicated

Command	Parameters	Description
Ping	Any data length from 0 to 16	Pings the display. The display returns what was sent.
Get Version	none	Returns hardware version
Write user flash	16 bytes of user data	Stores data in non-volatile memory.
Read user flash	none	Returns 16 bytes of data stored in non-volatile memor.

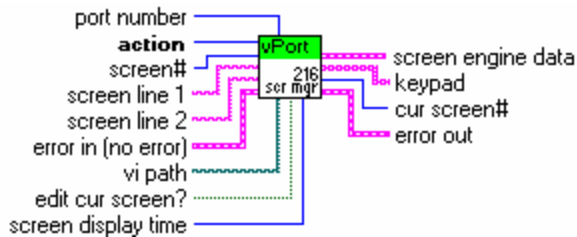


Command	Parameters	Description
Store current state	none	Stores the current state of the display as the boot state.
Reboot	3 bytes of data = 8,28,99	Reboots the Display.
Clear Screen	none	Clears LCD.
Set Line 1	16 characters	Data set to display line 1
Set Line 2	16 characters	Data set to display line 2
Set Special Character	9 bytes	Sets the font definition for one of the special characters. Byte 0 index of special character Byte1-8 bitmap of the new font for this character. Any value is valid between 0 and 63, the msb is at the left of the character cell of the row, and the lsb is at the right of the character cell. Byte 1 is at the top of the cell, Byte 8 is at the bottom of the cell.
Read LCD Memory	Byte - Address	Read LCD memory at the specified address
Set Cursor Position	2 bytes – Byte 1 column (0-15) Byte 2 row (0-1)	Moves cursor to column and row
Set Curser Style	Cursor style (0-3)	Changes the cursor style to the followinf: 0 – none; 1 – blinking block cursor; 2 – underscore; 3 – blinking bock plus underscore
Set LCD contrast	1 Byte (0-50)	Sets contrast of the Display 0 – light, 16 – about right, 29 dark, 30-50 very dark (best at cold temperatures)
Set backlight	1 Byte (0-100)	Controls the intensity of the backlighting. 0 – off, 1-99 variable brightness, 100 - on











## ViewPort 216 Screen Manager

The ViewPort Screen Manager is the main interface to the ViewPort Display hardware. The manager provides direct access to the serial communication between the application and the display. The manager is the primary interface to the screen engine, providing control over the engine's mode of operation. The manager also maintains the parameters for the Screen Engine, including information for each screen being displayed.

### Connector Pane



### Controls and Indicators

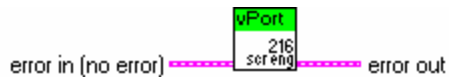
-  **action** Varies actions or method that the screen manager handles.
-  **screen line 1** Character data for line 1 on the display.
-  **screen#** Screen number to be written to.
-  **screen display time** An integer representing the number of seconds that this page should be displayed.
-  **vi path** If the page is to be a dynamic page, this path specifies the path to a VI that will provide the page content to the ViewPort Engine.
-  **edit cur screen?** An input that is used by the "engine set edit cur screen" action, specifies whether or not this page should be marked as being edited.
-  **port number** **port number** varies by platform. The parameters for serial **port numbers** depend on the whether you use Macintosh, Windows, or Unix.
-  **screen line 2** Character data for line 2 on the display.
-  **screen engine data** This Cluster contains the internal state of the ViewPort Engine.
-  **keypad** This is an Array of strings that represent the keypad information returned by the ViewPort keypad interface.

## ViewPort 216 Screen Engine

The ViewPort Screen Engine handles the automatic display of screens for the application. The engine will cycle through enabled screens based on the timing information the application provides.

The engine is implemented as an independent loop, and will only exit when the application set the engine's mode to exit via the ViewPort Screen Manager. The application can take manual control of the engine, and either force a particular screen to be displayed OR directly write to the ViewPort display.

### Connector Pane






## ViewPort 216 Xlat Keypad

The Xlat Keypad VI converts the raw keyboard codes to a common set of entries.

### Connector Pane



### Controls and Indicators

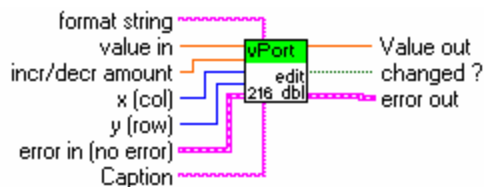
-  **raw data in** Raw keypad data received from the hardware.
-  **keypad** Translated Keypad commands (up, down right, left, accept, cancel)
-  **raw data out** Remaining Raw Keypad data after A Xlat has been run. This data is still valid and contains any command after the current translation.

Keyboard entry	
Up	0
Down	1
Right	2
Left	3
Accept	4
Cancel	5









## ViewPort 216 Edit Double

The Edit Double functions as a keypad handler that allows the user to increment or decrement a double value on the display. The calling routine supplies the increment/decrement value to apply when a user presses the Up/Down keys. The initial value of the display is based on the Caption (1st line of 216 display) and the Format string and double initially supplied (displayed on second line).

### Connector Pane



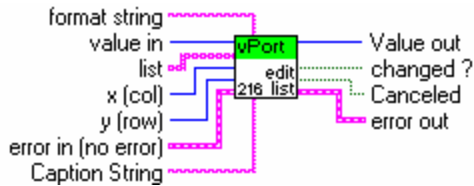
### Controls and Indicators

-  **incr/decr amount** The amount that the initial value is incremented or decremented for each corresponding up or down key pressed.
  
-  **value in** The Double value to be incremented or decremented.
  
-  **format string** The Labview format string for the 2nd line of the labview double.
  
-  **x (col)** Not used in 216 model.
  
-  **y (row)** Not used in 216 model.
  
-  **Caption** The 1st line to be displayed on the 216 during the edit double execution.
  
-  **changed ?** Boolean to test to see if value changed.
  
-  **Value out** The resultant value of the user interaction with the 216 keypad and display. If the user hit the cancel key the value will revert back to the input value, otherwise it will be the updated double value.

## ViewPort 216 Edit List

The Edit List VI allows the Display of a string list and a caption. It manages user keypad entry to scroll through the list and allows the user to either Enter (accept changes) or cancel (abort changes) of the string list selection. The Up and down Arrow keys are used for scrolling through the list.

### Connector Pane



### Controls and Indicators

**I32** **value in** Startup index into string list

**abc** **format string** Labview Format String to Format string list

**U8** **x (col)** Not used for 216 model

**U8** **y (row)** Not used for 216 model

**abc** **list** List of strings to display (up and down arrow move through the string list)

**abc** **Caption String** String placed on first line

**TF** **changed ?** Will be true if user accept the changes by pressing the accept key or false if the user presses the cancel key.

**I32** **Value out** Sting List index after Edit list runs. The value will be the same as input if cancel entered. If enter is pressed the value will be the index of the selected string.

**TF** **Canceled** True when user pressed cancel key.

# Performance Hints

---

## General Considerations

In RT applications always make sure to keep the ViewPort code out of the Time Critical loop. Avoid using dynamic variables such as arrays that require relatively slow memory manager support and are sometimes limited to the memory available on board.

Any dynamically called VI's need be either downloaded to the target with the VI path on the target specified in the application or include a case statement that never runs that calls all the Dynamic Screen VI's (and their custom subVIs) so that the VIs are included automatically in the download.

## Cable issues

Serial Cable shall be a Male to Female type, 1 to 1

## **Additional LabVIEW VIs**

---

Included on the installation disk are additional VI's that exercise many of the ViewPort Screen manager options. The "Test ViewPort 216.VI" includes screen calls that utilize all the functionality mentioned in this manual plus it adds some additional VI's just for reference and to show other capabilities of the ViewPort.



## ViewPort Error Codes

---

Error Code	Error String	Error Description
Set by Labview Serial init	ViewPort 216 Screen Manager – serial init failed	There is a problem with the Serial port initialization. Either the port does not exist or is in use already.
Set by Labview Open Reference	ViewPort 216 Screen Manager – cannot locate screen callback: <screen vi>	The Vi reference supplied to the screen manager could not be found or does not exist.
100	ViewPort 216 Screen Manager – invalid screen #	A call was made with an out of range screen number.

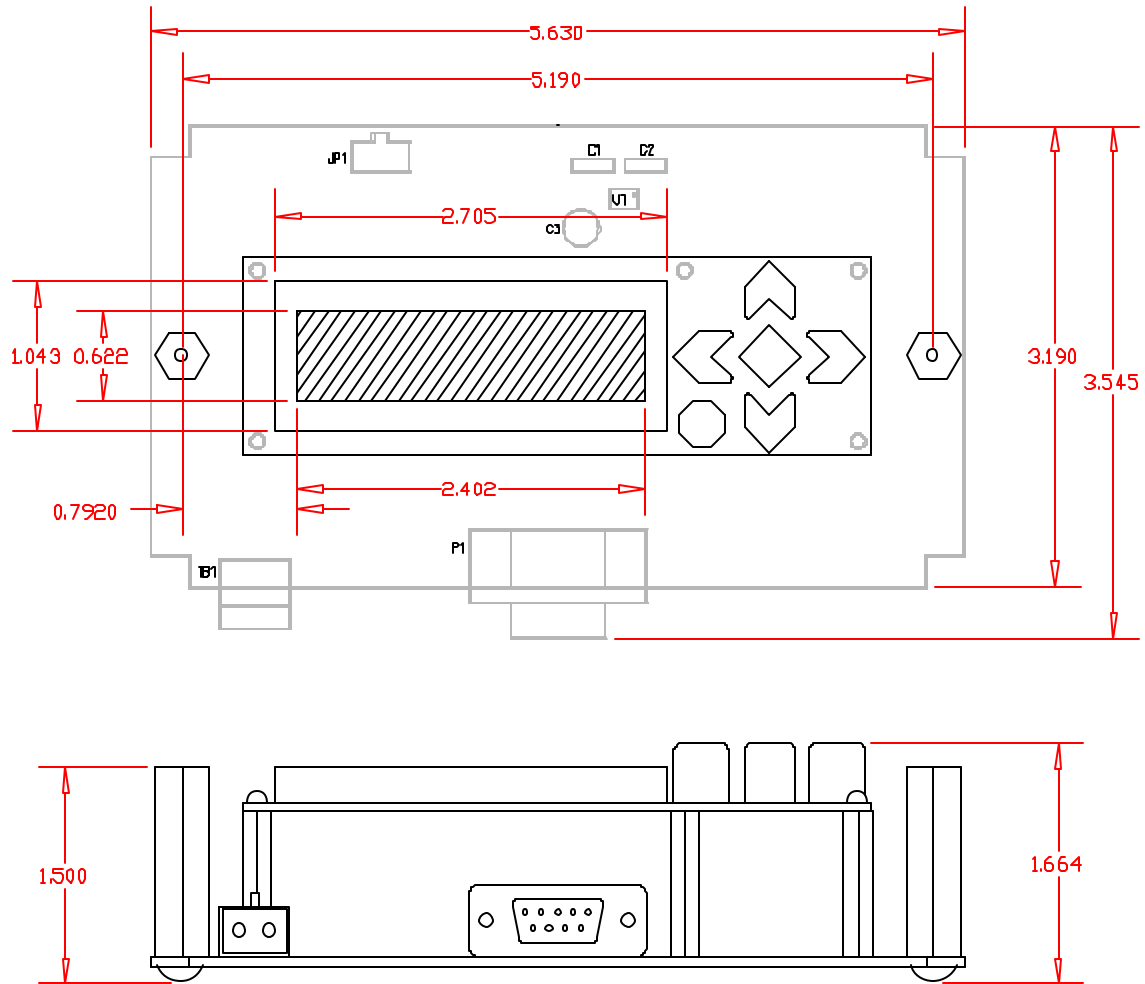
# Diagnostics

---

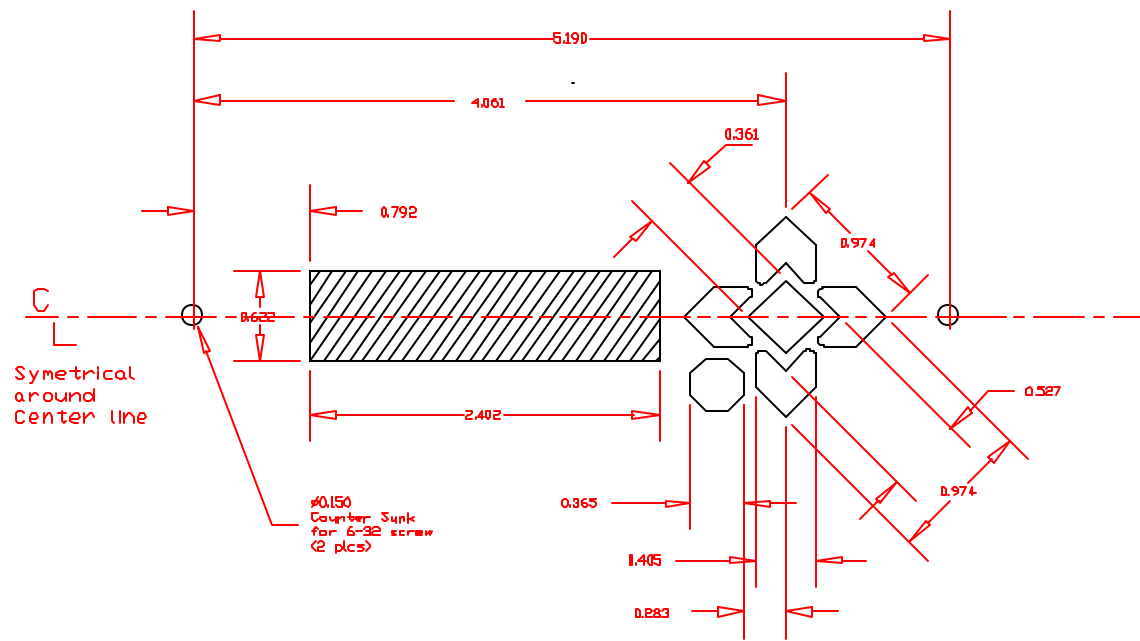
Problem	Help
No Display	Check power connections' making sure power is indeed at terminal block and polarity is correct.
No communication	Be sure that a 1 to 1 cable is being used and that the right com port number is specified.
Screen doesn't dynamically update	Be sure that the Screen VI is either being called inside the application while loop or that the application is setup to call the screen VI by reference. If the VI is called by reference, be sure that the VI is downloaded with the correct path information to the target.
Dynamic Screen doesn't show	Be sure the VI path is correct in the application. Also be sure VI is download to target as required.
Parts of the display show <i>blocks</i> instead of the expected text.	Be sure the strings that are sent to the display are defined as 16 characters long.

# Specifications

## Dimensions



# Recommended Panel Cutout



## Power Requirements

Input Power.....7-30V DC; 2.0 watts

## Environment

Operating Temperature .....0-50 °C

Storage Temperature .....-10-60 °C

Operating Relative Humidity ..... 90% max non-condensing

## Physical

Characters.....32 (2 lines by 16 characters)

Display Area .....2.4 x 0.62 inches [61.0 x 15.8 mm] (XxY)

Dot Pitch ..... 0.60 x 0.70 mm (X x Y)

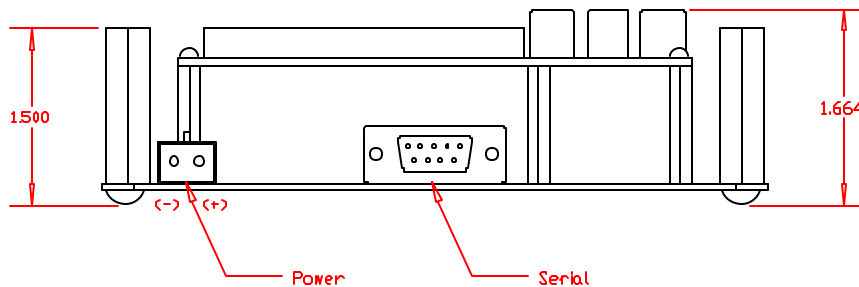
Dot Size ..... 0.55 x 0.65 mm (X x Y)

Character Size ..... 2.95 x 5.55 mm (X x Y)

## Connectors

Serial.....DB9 – Female

Power .....Terminal Block, 2 position Compression Screw, 16 AWG max



## Contacting Us

---

Viewpoint Systems, Inc.  
800 West Metro Parkway  
Rochester, NY 14623

voice: 585-475-9555

fax: 585-475-9645

e-mail: [support@ViewpointUSA.com](mailto:support@ViewpointUSA.com)

Technical support is available any business day from 9:00 AM to 5:00 PM Eastern time. Of course, you may fax or e-mail questions at any time.

