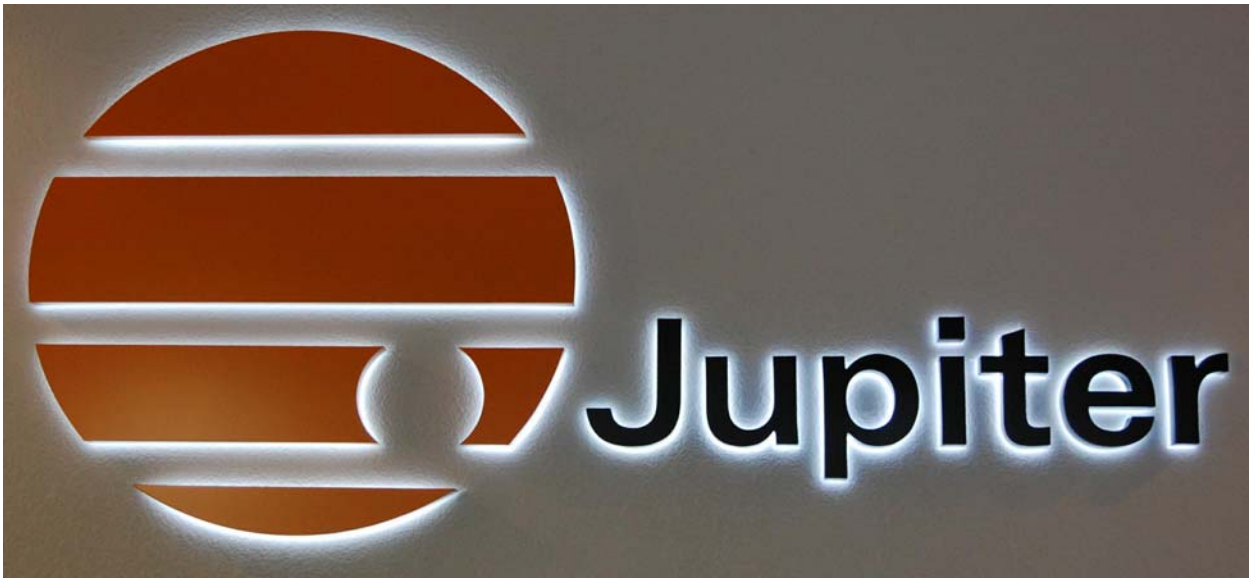
Jupiter Systems

ControlPoint Protocol Manual For Fusion Systems

3.5

October 30, 2014

A-FC0-000-01, Rev. M


Jupiter

Copyright

Copyright

Copyright © 2014 Jupiter Systems. This document is copyrighted with all rights reserved.

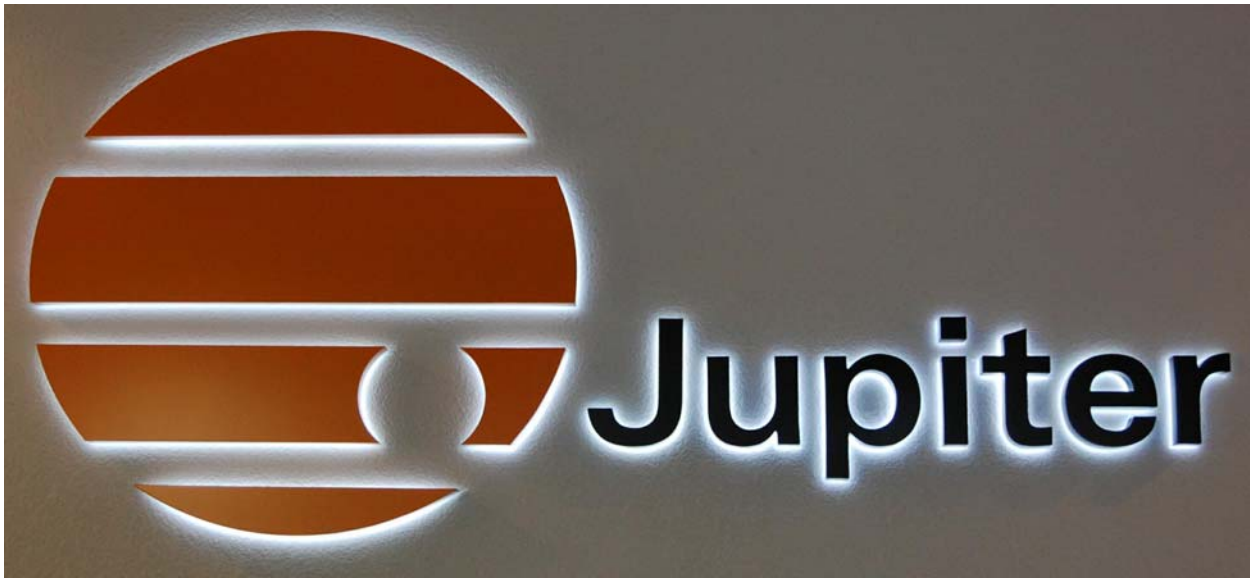
The entire risk of the use or the result of the use of this Hardware and Software and documentation remains with the User. Information in this document is subject to change without notice. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form or by any means, electronic or mechanical, including photocopying without express written permission of Jupiter Systems. See also **“Statement of Limited Warranty” on page v.**

Acknowledgements

Jupiter Systems®, Jupiter logo® are registered trademarks of Jupiter Systems. Fusion Catalyst™, SVS-8™, and ControlPoint™ are trademarks of Jupiter Systems.

All other brands and names are the property of their respective owners.

Jupiter Systems
31015 Huntwood Avenue
Hayward, CA 94544-7007
510-675-1000 (v)
510-675-1001 (f)
info@jupiter.com
support@jupiter.com
510-675-1007 (v)


Jupiter

Warranty

Software Warranty and Special Provisions

Limited Warranty

Limited Warranty. Jupiter Systems warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of sale. Any implied warranties on the SOFTWARE are limited to ninety (90) days.

Customer Remedies. Jupiter Systems' entire liability and your exclusive remedy shall be, at Jupiter Systems' option, either (a) return of the price paid, or (b) repair or replacement of the SOFTWARE that does not meet this Limited Warranty and which is returned to Jupiter Systems with a copy of your receipt or purchase order number. This Limited Warranty is void if failure of the SOFTWARE has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

No Other Warranties. To the maximum extent permitted by applicable law, Jupiter Systems disclaims all other warranties, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with regard to the SOFTWARE and the accompanying written materials.

No Liability for Consequential Damages. To the maximum extent permitted by applicable law, in no event shall Jupiter Systems or its suppliers be liable for any damages whatsoever (including without limitation, special, incidental, consequential, or indirect damages for personal injury, loss of business, profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this product, even if Jupiter Systems has been advised of the possibility of such damages. In any case, Jupiter Systems' entire liability under any provision of this agreement shall be limited to the amount actually paid by you for the SOFTWARE.

Software License Agreement

Please review the following terms and conditions before installing or using the software supplied. By opening the software package or by installing the software or by using the installed software, you indicate your acceptance of such terms and conditions. In the event you do not agree to these terms and conditions, you may not use the software, and should promptly contact Jupiter Systems.

This Software License Agreement (the Agreement) grants you a non-exclusive license to use the software supplied to you by Jupiter Systems (JUPITER), including software that may be owned by third parties and licensed to Jupiter, with the right to distribute and sublicense, which software (the Software) may be supplied to you on removable media, and/or as part of the equipment supplied by JUPITER. The Agreement also imposes certain restrictions on your use of the Software.

You may use the Software only on the equipment with which or for which it was supplied. For internal backup purposes, you may make copies of the Software on removable media but you may not use a copy on another piece of equipment and you may not transfer these copies to any other party. You may not make any other copies of the Software and you may not make any other copies of the written materials accompanying the Software and/or the equipment supplied by JUPITER.

You may not sublicense any rights granted in the Agreement. You may not transfer the Software, except upon: (i) a transfer of this License Agreement; (ii) a transfer of the JUPITER hardware equipment with which the Software was supplied; (iii) your providing the transferee with a copy of this Agreement; (iv) the transferee accepting the terms and conditions of the Agreement. If you transfer the Software to another party, you must at the same time either transfer all copies whether in printed or computer readable form to the same party or destroy any copies not transferred. Include all modifications and portions of the Software contained or merged into other programs. You must also reproduce and include the copyright notice on any copy. You agree to comply with all laws of the United States regarding the export and/or re-export of the software.

All intellectual property rights in the Software and user documentation are owned by JUPITER and/or its licensors and are protected by United States copyright laws, other applicable copyright laws, other applicable proprietary rights laws (including but not limited to trade secret laws) and international treaty provisions. **YOU MAY NOT USE, COPY, MODIFY OR TRANSFER THE SOFTWARE OR ANY COPY THEREOF, IN WHOLE OR**

PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS AGREEMENT. TO THE EXTENT PERMITTED BY APPLICABLE LAW, YOU MAY NOT DECOMPILE, REVERSE ENGINEER, OR DISASSEMBLE THE SOFTWARE, IN WHOLE OR IN PART. IF YOU TRANSFER POSSESSION OF THE SOFTWARE, ANY COPY OR ANY PORTION THEREOF, TO ANOTHER PARTY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

JUPITER retains ownership of the Software (except those portions that may be owned by third parties which retain the ownership thereof) and no rights are granted to you other than a license to use on the terms expressly set forth in the Agreement.

As defined in FAR section 2.101, DFAR section 252.227-7014(a)(1) and DFAR section 252.227-7014(a)(5) or other foreign government regulations regulating the use of commercial software by such government or otherwise, all software and accompanying documentation provided in connection with this Agreement are “commercial items”, “commercial computer software” and/or “commercial computer software documentation”. Consistent with DFAR section 227.7202, FAR section 12.212, and other applicable foreign government regulation, any use, modification, reproduction, release, performance, display, disclosure or distribution thereof by or for the U.S. or other foreign government shall be governed solely by the terms of this Agreement and shall be prohibited, except to the extent expressly permitted by the terms of this Agreement. You shall ensure that each copy used or possessed by or for the government is labeled to reflect the foregoing.

Upon any violation of any of the provisions of the Agreement, your rights to use the Software shall automatically terminate and you shall be obligated to return the Software to JUPITER or to destroy all copies of the Software. If you destroy such Software, you agree to send JUPITER written notification of such destruction. This Agreement shall be governed by California law, other than its provisions concerning the applicability of laws of other jurisdictions.

The only warranties are those specifically granted by JUPITER pursuant to its Standard Terms and Conditions of Sale, which are expressly incorporated herein. The liability of JUPITER, and its licensors, is specifically limited as set forth in these Standard Terms and Conditions of Sale.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS; YOU FURTHER AGREE IT IS THE COMPLETE AND

EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN AND BY ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

Using this Manual

Using this Manual

Introduction

Chapter titles are at the top of every page to assist you in finding sections.

The Table of Contents is a section, chapter, and heading outline of the manual; whereas, the comprehensive index at the end of the manual guides you through a search of subjects, figures, and tables.

Note and Caution

This manual uses three special entries to get your attention:

- Note
- Caution
- Warning

These entries are listed in their ascending order of importance. The examples shown are found throughout this manual.

Note	Notes are entries that bring your attention to specific items that you must see, read, and understand before continuing.

Caution	Cautions are entries that alert you to items that may cause the operating system to not operate properly. For instance, tasks that were either done out of sequence or not supposed to be done at all may cause the system to malfunction. Cautions also alert you about physical connections that can cause the system to not operate properly.

Warning	Warnings are entries that demand your attention regarding damage to the system.

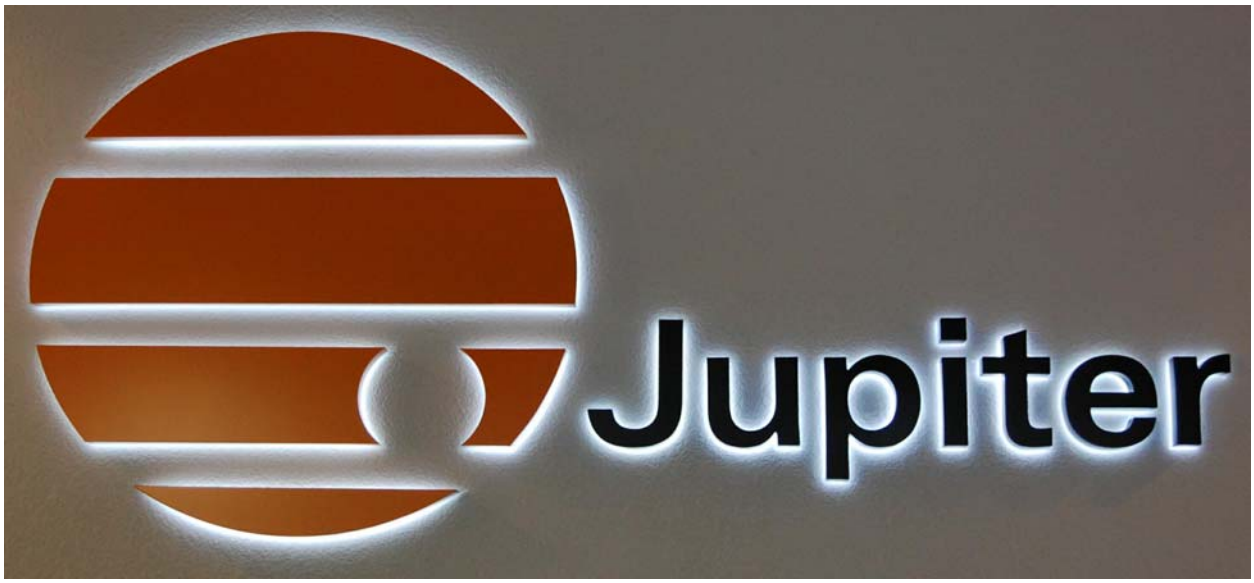

Jupiter

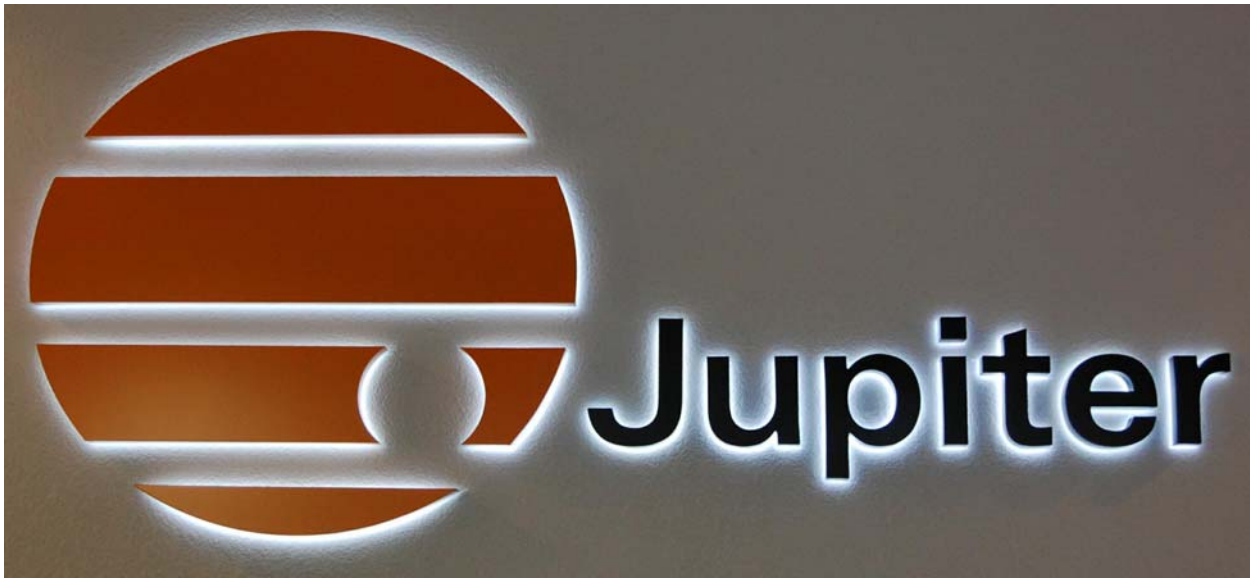
Table of Contents

Contents

Contents

Contents

Contents


Jupiter

Chapter 1—Getting Started

1. Getting Started

This chapter discusses the use of the GalileoConnect Serial and TCP/IP remote control protocol and its use with ControlPoint. The ControlPoint protocol will be used when connecting a touch panel controller to the Wall Controller System for remote control, or when using third-party created software for controlling the Wall Controller System.

A knowledge and understanding of the Binary and Hexadecimal number systems will be necessary in understanding these command structures as they use the binary bit position. Hex will be needed for setting up the commands while all responses will be in decimal notation. You will need to convert these responses to Hex in order to apply them to the command structure.

The ControlPoint Server

The ControlPoint Server allows for many types of connections.

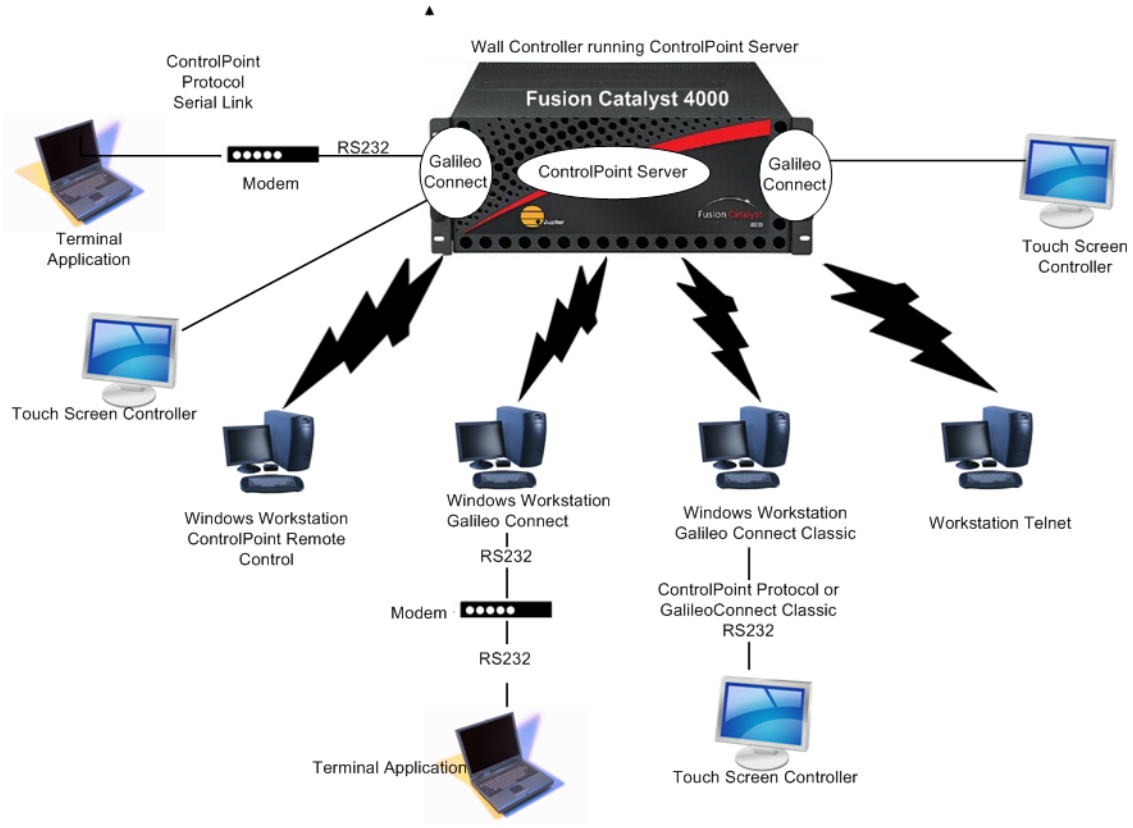


Figure 1 - Remote Connection Options

1.1 Types of Possible Connections

- Remote System Running ControlPoint over modem link (RS-232) to Wall Controller ControlPoint Server
- Touch Panel connected locally (RS-232) to Wall Controller ControlPoint Server using GalileoConnect
- Windows Workstation Running ControlPoint Client (TCP/IP) to Wall Controller ControlPoint Server
- Remote System over modem link (RS-232) to a Windows Workstation Running ControlPoint Client (TCP/IP) to Wall Controller ControlPoint Server

- Touch Panel connected to remote workstation (RS-232) running ControlPoint Client (TCP/IP) to Wall Controller's ControlPoint Server and GalileoConnect or GalileoConnect Classic
- Windows Workstation Running TCP/IP to the Wall Controller's ControlPoint Server

ControlPoint is a client/server pair that work together through TCP/IP over RS-232 or your network (or locally on the Wall Controller) to allow complete control and manipulation of your ControlPoint windows. The server is part of the ControlPoint Software that was installed on your Wall Controller System when it was manufactured. The ControlPoint Client must be installed on any workstation that will be used to control the wall.

ControlPoint uses the TCP/IP protocol to operate over your network and uses port 25456 (0x6370 hex) to communicate.

1.2 Communicating with the ControlPoint Server

Communication can be established with the ControlPoint Server over the network or through GalileoConnect with an RS-232 link between the serial device and the Wall Controller.

1.2.1 Network Communications

ControlPoint can be used to control the Wall Controller System over the network. GalileoConnect uses the TCP/IP protocol to operate over your network and uses port 25456 (0x6370 hex) to communicate. GalileoConnect is a bridge that will listen on COMM PortX for input from your touch panel (**"Figure 4" on page 9**) and relay that information to the ControlPoint Server on the Wall Controller System via the network.

1.2.1.1 Getting Connected – Ethernet – Windows 2000

Follow the procedure below for getting connected with an Ethernet connection under Windows 2000.

1. From the **Start** button click **Run** and enter **Telnet**
2. Type **set local_echo,** so you can see what you type.
3. Type the command **open [Wall Controller Name—or IP address] 25456.**

Caution After the **Connect** command, you will see “Connecting...”, with no further cursor action go to #4. Do not type anything else at this time.

4. Once connected type **admin.**
5. Press **Enter.**
6. Press **Enter** again (no password).
7. You can now enter one of the + commands available in the ControlPoint Protocol for controlling the display wall - all commands **MUST** start with a plus sign and end with a **Return**.
8. If you do not enter the plus sign ControlPoint will disconnect you.
9. Enter a plus and a few random characters and press **Enter** and you should get a returned **=80040505** error message (bad parameter error). Check **Appendix B** for a listing of all error messages.

Once you have connected and logged in successfully, you should be able to send a command and see the returned response within the **Telnet** window.

The user name and password must be sent immediately after connecting (default is **admin** and no password).

1.2.1.2 Getting Connected – Ethernet – Windows XP

Follow the procedure below for getting connected with an Ethernet connection under Windows XP.

1. From the **Start** button click **Run** and enter **cmd**
   Run **Telnet**.
2. Type **set local_echo,** so you can see what you type.
   Type the command **open [Wall Controller Name—or IP address] 25456.**
3. Once connected type **admin.**
4. Press **Enter.**
5. Press **Enter** again (no password).
6. You can now enter one of the + commands available in the ControlPoint Protocol for controlling the display wall - all commands **MUST** start with a plus sign and end with a **Return**.

7. If you do not enter the plus sign ControlPoint will disconnect you.
8. Enter a plus and a few random characters and press **Enter** and you should get a returned **=80040505** error message (bad parameter error). Check **Appendix B** for a listing of all error messages.

Once you have connected and logged in successfully, you should be able to send a command and see the returned response within the **Telnet** window.

The user name and password must be sent immediately after connecting (default is **admin** and no password).

1.2.1.3 Login with a Network Connection

Once you are connected, the first two lines must be the user name and password:

1. Type or send **user name** and **Enter** – default is user name is **admin**
2. Type or send **password** and **Enter** – default is password is no password (just press **Enter**)

   Default login is simply **admin** followed by **Enter** followed by another **Enter.**

1.2.2 Serial Communication

Serial communication (RS-232) with the ControlPoint Server is accomplished through GalileoConnect. GalileoConnect acts as a bridge between ControlPoint and the serial port.

The GalileoConnect is covered in detail in the following section.

1.3 Introduction to GalileoConnect

GalileoConnect is included in the ControlPoint Software Suite to provide an interface for control of the Wall Controller System from third-party software or remote control hardware such as touch panels (i.e. AMX or Crestron). GalileoConnect provides a means for RS-232 communications between either the serial device (touch panel) or a remote system or workstation and the ControlPoint Server. This communication is in the form a bridge between the Server and an RS-232 communication port.

GalileoConnect allows for asynchronous bi-directional communications, such that serial devices can utilize the full potential of the ControlPoint Protocol as needed. Bi-directional communications allows the serial device to receive error messages and react to them as deemed necessary by the programmer.

GalileoConnect makes it possible for the user to control the Wall Controller System over an RS-232 connection from a serial device such as a remote workstation by using the **ControlPoint Client**, or a customer-written application, or with a touch panel controller. The Wall Controller System can also be controlled remotely from a modem as shown in the Figure 1.

GalileoConnect is a separate process that can be run either locally on the Wall Controller System, or on a remote workstation connected on the network to the wall-controller using TCP/IP. GalileoConnect acts as a bridge between the serial port and the ControlPoint Server or Client.

The serial device must be connected to a local serial (COM) port of the system on which GalileoConnect is running. GalileoConnect takes exclusive control of the specified serial port. Multiple instances of GalileoConnect can be run simultaneously on the same system, provided there is a separate serial port for each GalileoConnect instance running. When each instance is run, the specific serial port must be specified as a command-line parameter of the process.

On the other side of the communications bridge, GalileoConnect opens a TCP/IP socket to a user-specified ControlPoint Server (Wall Controller). The ControlPoint server host name and port address can be specified either from the command-line when GalileoConnect is started, or interactively by the serial device by using the special GalileoConnect command.

The following figure illustrates the different modules incorporated within the ControlPoint System. The **Windows Client API** makes it possible for the user to control a Wall Controller System from a remote workstation by using the **ControlPoint Client**, a customer-written application (**Customer Client**), or with a touch panel controller through the **GalileoConnect**

Client. The **Windows Client API** connects via the Ethernet to the Wall Controller System through the **ControlPoint Server**.

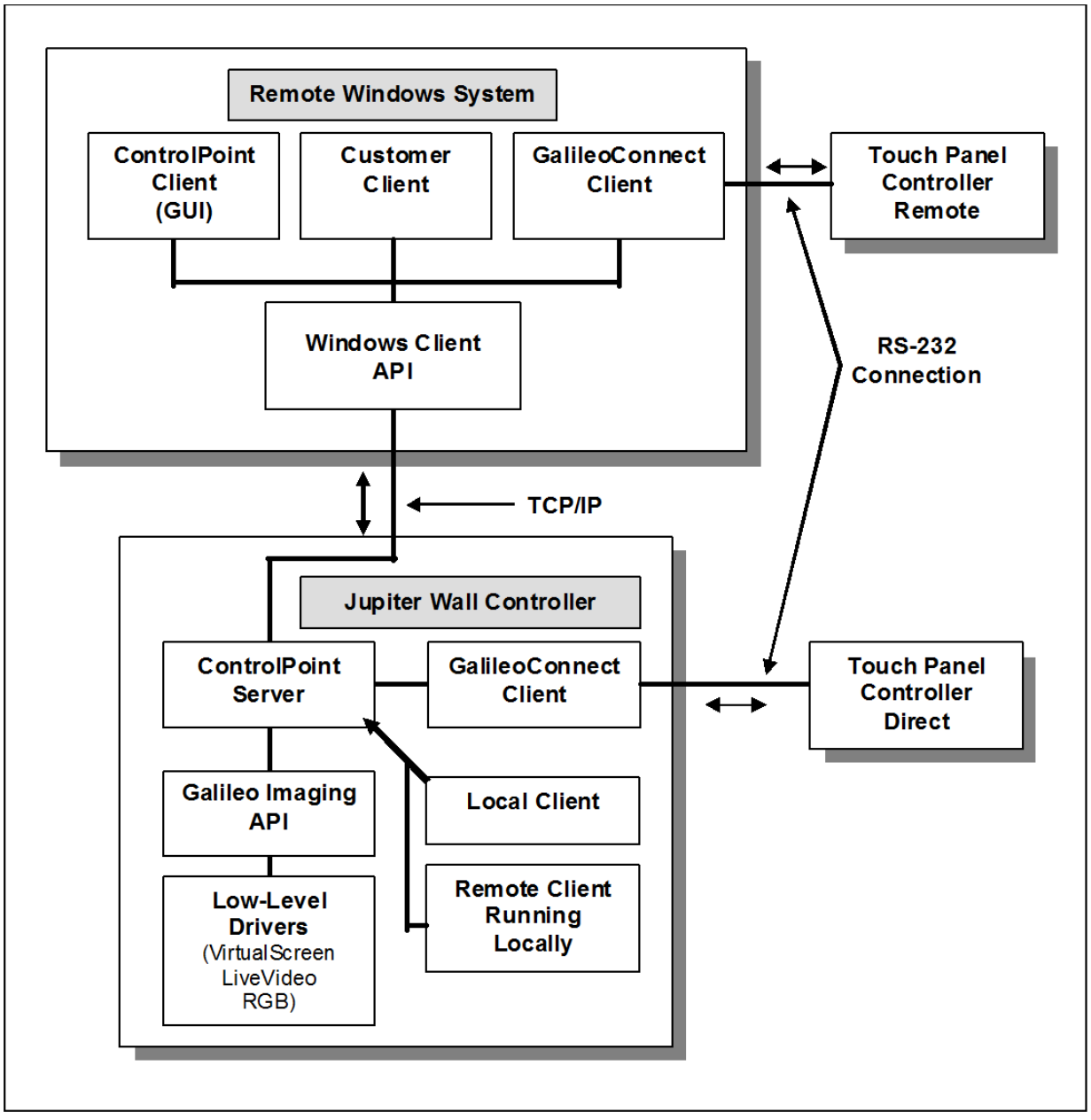


Figure 2 - ControlPoint Block diagram

1.3.1 Touch Panel Connections

There are two main ways that the serial device (touch panel) can be connected to the Wall Controller system:

- Directly through the serial port with an RS-232 cable or via the network, **or**
- Indirectly through a remote computer system, which then connects to the Wall Controller System via the network.

1.3.2 Direct Connections

The serial device (touch panel) can be connected to the Wall Controller System through its primary serial port with an RS-232 cable, or through a network connection if it has network capability.

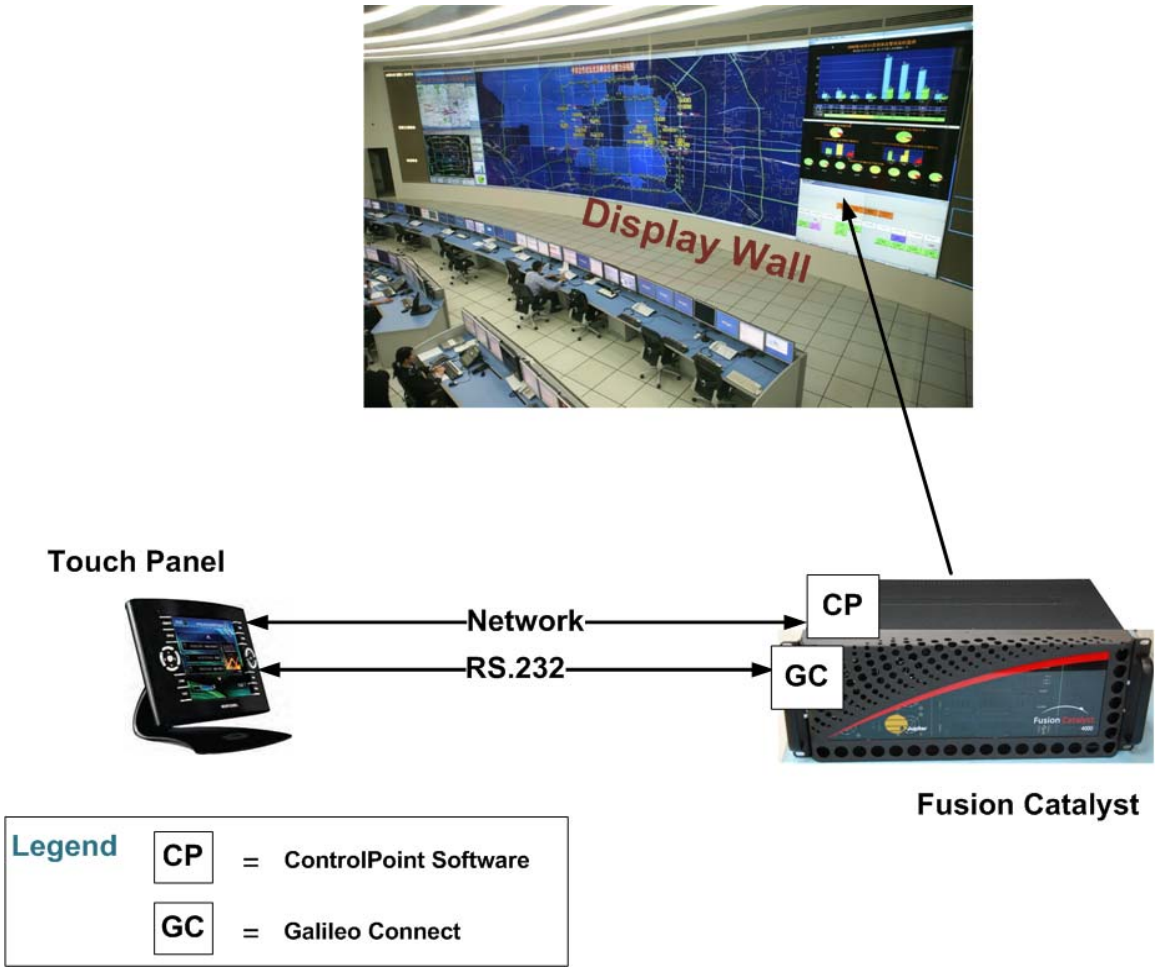


Figure 3 - Touch Panel Connection Options

1.3.3 Remote System Connection

GalileoConnect may be used to communicate from a remote system over the network to the Wall Controller System and control the wall. In this case, the touch panel controller will be connected to the serial port of the remote system. The remote system will communicate with the Wall Controller System over the network through ControlPoint. GalileoConnect and ControlPoint Client must be installed on the remote system. The following figure illustrates the remote setup described here. Refer to **"Figure 1" on page 2**.

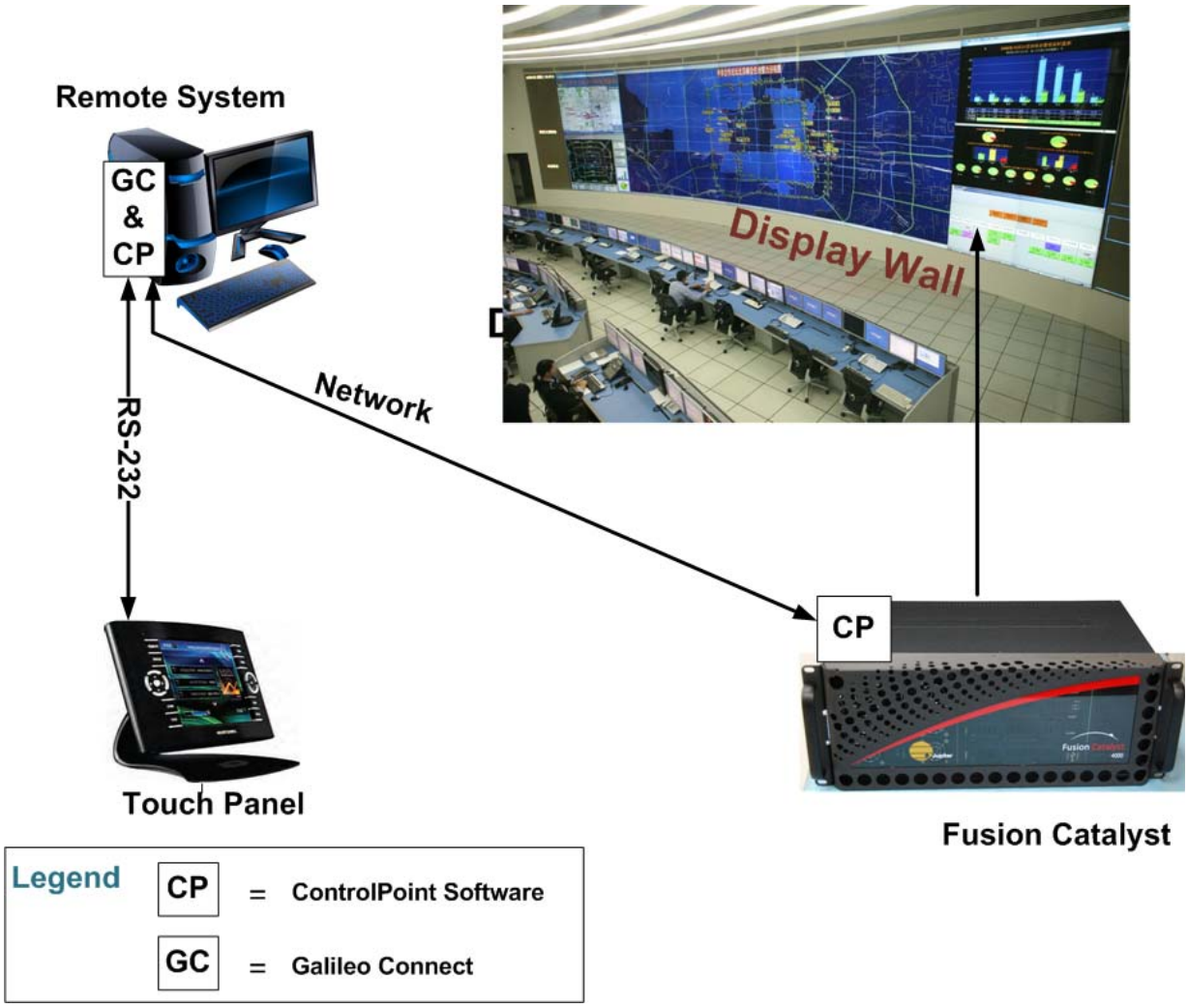


Figure 4 - Remotely Connected Touch Panel

1.4 Serial Communication (Protocol and Syntax)

By default the serial communication is configured for **9600,n,8,1** with no flow control protocol and **echo** enabled (every received character is echoed back to the sender).

While executing an interactive command, the received characters are not echoed back regardless of the echo setting.

Commands must be terminated with **CR** (ASCII code 13) and **LF** (ASCII code 10). Empty commands or commands consisting entirely of white-space characters (ASCII codes 9, 11, 12 and 32) are ignored.

Unsupported commands (commands not relevant to this version of the software) respond with an error message.

Unrecognized commands cause an error message to be returned.

All messages are terminated with **CR** (ASCII code 13) and **LF** (ASCII code 10).

A **NULL MODEM** is required to connect a touch panel to the Wall Controller System. Consult the operating manual of your serial device.

The GalileoConnect serial protocol is ASCII-based. Lines of text are exchanged as commands and responses, with **<CR><LF>** control characters marking the end-of-line. The protocol is based on command-response transactions, where the serial device sends one line of text for the command and receives one line of text as a response. The only exceptions are the asynchronous ControlPoint notifications, which do not expect a response and can be interleaved between a command and its response. The serial device can choose whether to receive asynchronous notification or not. They are not required to process asynchronous communication events, except to wait for a response before sending the next command.

1.5 Starting GalileoConnect

Before using a serial device such as a touch panel controller (or control program) with the ControlPoint Server through a serial connection, you must start GalileoConnect. If you are using a Remote System connection, you must start ControlPoint on the remote system as well as GalileoConnect. You can start GalileoConnect either from the Start Menu, executable icon (or its Shortcut) or from the command prompt. Use the Start Menu or the shortcut icon if you have a smart touch panel that can do

authentication, or use the command line to authenticate when you start GalileoConnect if you cannot authenticate from your touch panel.

Running the GalileoConnect application connects the serial port to the ControlPoint server. Once you are connected to ControlPoint, you may log into ControlPoint and authenticate. This can be done from the command line or from the serial device. You can verify your connection with the Connect command. The connection is between the serial port and the ControlPoint Server using GalileoConnect.

Hints Put a copy of GalileoConnect in the startup folder.

Note If you get an error stating that the serial port is already in use, either GalileoConnect is already running or you have an application such as HyperTerminal running that is already using the serial port.

1.6 Starting from the Start Menu

1. From the Start – Programs – GalileoConnect menu, select GalileoConnect.

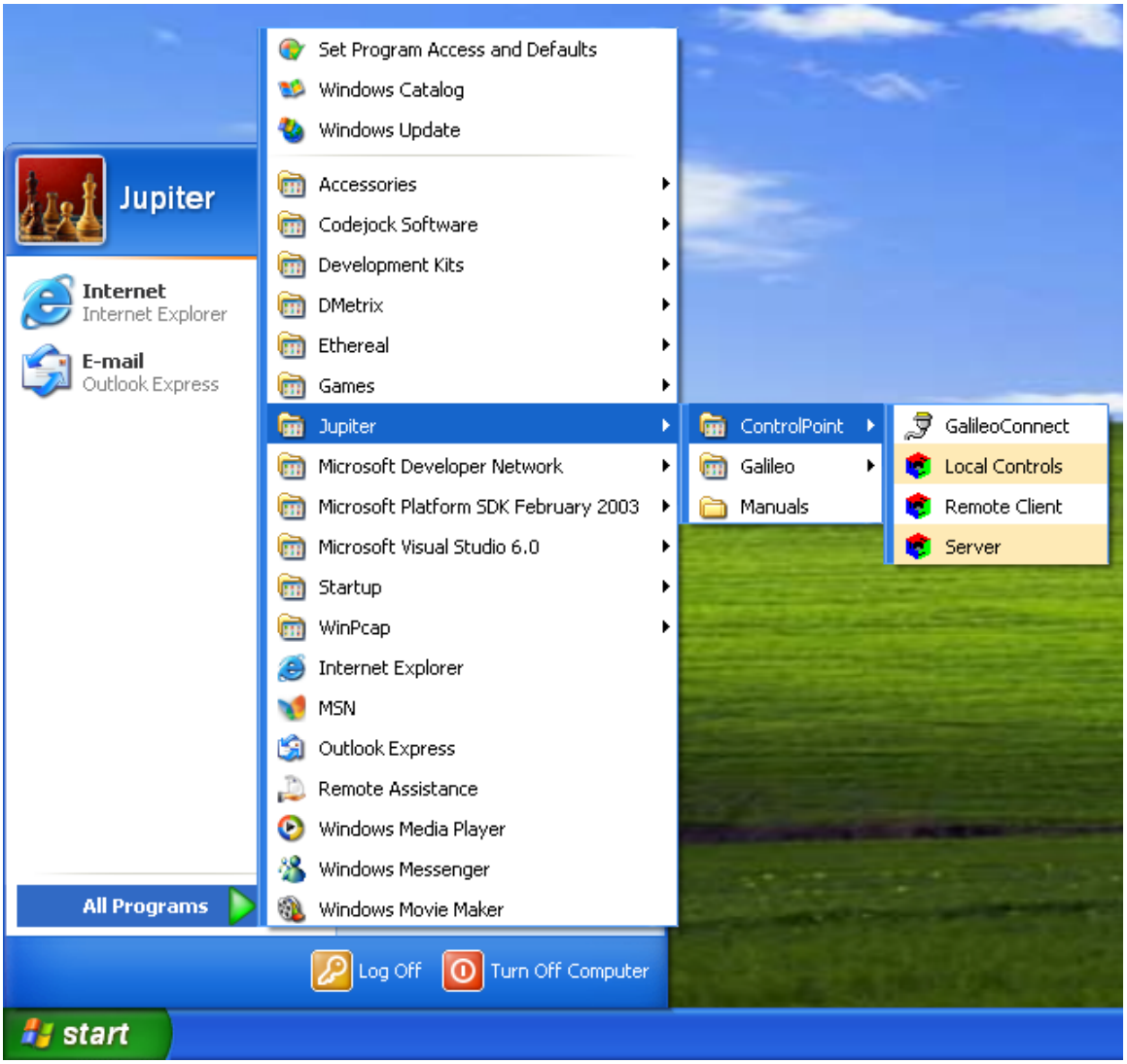


Figure 5 - Start Menu for GalileoConnect

You will notice a new icon in the system tray. This icon is shown in the following figure.

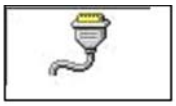


Figure 6 - GalileoConnect Icon

There are several methods that can be used to authenticate from the touch panel – these methods are discussed later in this manual in the chapter on programming. If you cannot authenticate from the touch panel you must use the command line method to start GalileoConnect, log in and authenticate.

Once you have started the GalileoConnect server, right clicking the icon in the system tray will pop up a menu where you can select About or Quit. Quit, will stop the GalileoConnect Server. About will display version and build information; as an example, see the box shown in the following figure.

Figure 7 - GalileoConnect About (Representative Image)

Hints Use a modified shortcut **Target** line with GalileoConnect command line arguments to start GalileoConnect and authenticate all at one time. Drop this shortcut icon into the **Start Up** folder to start GalileoConnect and authenticate automatically when Windows starts.

1.7 Starting from the Command Line

This section discusses how to start GalileoConnect from the command line, and the available command line options.

1.7.1 Command Line Help

There is startup help available for GalileoConnect. When GalileoConnect is started from the command line with a /? or /h switch, you will see the dialog shown below.

Examples:

C:\Program Files (x86)\Jupiter\ControlPoint\GalCon.exe /h
C:\Program Files (x86)\Jupiter\ControlPoint\GalCon.exe /?

Figure 8 - GalileoConnect Help

This dialog shows you the default parameters for GalileoConnect. These default parameters will connect the serial port to ControlPoint Server running on the local machine and talk to your COM1 port. GalileoConnect will perform authentication by using the default arguments – or the command line parameters must be entered (refer to the following section).

1.7.2 Using the Command Line

From the **Start - Run** dialog, type

C:\Program Files (x86)\Jupiter\ControlPoint\GalCon.exe

Once **GalileoConnect** is running on the system, you will see the icon in the system tray.

The GalileoConnect executable accepts many arguments from the command line when it is run. Arguments can be in any order. Omitted arguments use default values. Arguments are entered as an argument and value pair **<argument=value>.**

Arguments:

- **comport** specifies the name of the serial port to use for serial communications to the external serial device (default is **COM1**).
  comport=COM1

- **baud** – specifies the baud rate for the serial connection. Supported baud rates are: 1200, 2400, 9600, 19200, 38400, 57600, 115200 (default is **9600**).
  baud=1200

- **server** is the host name or IP address of the ControlPoint Server to which you will connect (default is **localhost**).
  server=jupiter123A

- **port** is the TCP/IP port of the target ControlPoint Server (default is 25456).
  port=25456

- **user** is the user name used for authentication with the Control-Point Server (default is **localuser**).
  user=username

- **pass** - used for authentication (default is **no password**)
  pass=

- **auto** – enables or disables the **auto-connect** feature. If a value

of **true** is specified (**auto=true**), GalileoConnect will automatically try to reconnect with the ControlPoint server if the connection breaks. An attempt is made about every 5 seconds. If the serial device will handle server connections programmatically, auto-connect can be disabled by specifying **false** for the argument (**auto=false**). Default is **true**.

auto=true

GalileoConnect checks the incoming commands for ControlPoint validity. It checks whether or not they are well-formed commands, but does not check syntax. Bad commands will not be relayed to ControlPoint; and as a result, ControlPoint will not terminate the connection on bad commands. A bad command error is returned for bad commands. If the connection is broken for some other reason, auto-connect will start up and re-establish the connection.

- **log** – enables or disables serial communication logging. If the value **true** (**log=true**) is specified, GalileoConnect will log the data exchanged via the serial link. The log is appended to a file xxxlog.txt, where xxx is the name of the serial port used for communication (i.e. **com1log.txt**). Use the value **false** (**log=false**) to disable logging. Default is **false**.

  log=true

Note	Much data is sent between the serial server and the serial device. Care should be taken to avoid the creation of excessively large log files. You must close GalCon (quit GalileoConnect) in order to write the log file and view it.

Examples:

You want to run GalileoConnect and use serial port **COM2**, connecting to ControlPoint server CPServer, authenticating user serialuser with password secretword, and auto-connect is enabled since the serial device does not handle connecting and authentication. Non-specified arguments use default values.

GalCon.exe comport=COM2 server=CPServer

user=serialuser pass=secretword auto=true

You want to run GalileoConnect on the default serial port (COM1) with auto-connect disabled, and with logging enabled. The serial device will establish connection with the ControlPoint server and do the authentication.

GalCon.exe auto=false logging=true

Once a serial connection is established with GalileoConnect, GalileoConnect relays the data stream from the serial port to/ from the ControlPoint Server directly without any translation.

Caution **All** text lines must end with a CR/LF.

1.7.3 Default Values

The command line arguments used with galcon.exe, along with their default values are listed in the table below.

Table 1: Default Argument Values

Argument	Value
comport	COM1
baud	9600
server	localhost
port	25456
user	localuser
pass	(none)
auto	true
log	false

The default values listed in the table are pre-programmed into the system. The programmed default value will be used for any non-specified arguments when the **galcon.exe** command is executed. Each argument's programmed value will be changed to the last value used on the command

line when executing the **galcon.exe** command. For example, the programmed default for the server is **localhost**. When you use the command line **server** argument with the value **jupiter123**, the programmed value of **localhost** will be changed to **jupiter123.**

1.8 Starting Automatically

The easiest way to start GalileoConnect automatically is to drop a shortcut to GalileoConnect into the Windows Startup folder on the Wall Controller. You may edit the Target line to include the argument information discussed in the previous section. Refer to, **“Starting from a Shortcut” on page 21** for more information.

If authentication is done automatically from the Start folder icon, or if it is done automatically from the serial device, then some provision must be made on the touch panel for manual or programmatically re-authenticating. There is no way for the external device to know if and/or when the Wall Controller has been rebooted, or if either the ControlPoint or GalileoConnect have been restarted. If this happens, the serial device may not be connected or authenticated. It is recommended that a **Start** button or some program be added such that when activated the external device sends the Connect command with authentication information. Refer to GalileoConnect command line option **Auto** and, **“Programming Considerations” on page 168** for staying connected.

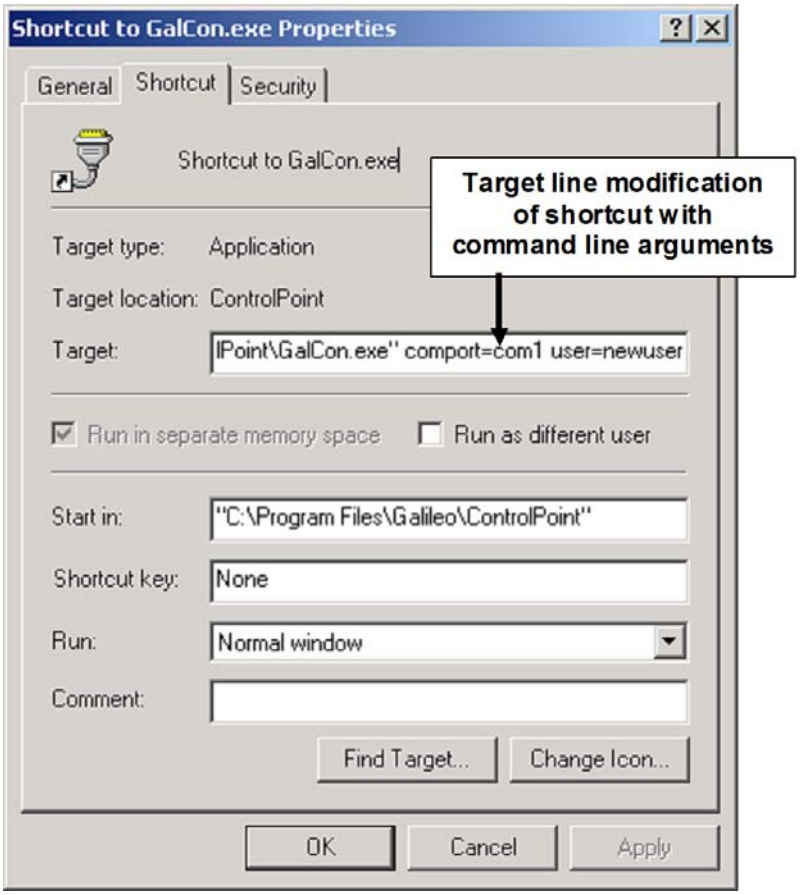


Figure 9 - Shortcut Dialog

Note	Windows 2000 does not allow you to create an editable shortcut by dragging from the **Start / Programs** menu items. You must go to the executable icon and create a shortcut from that icon – then edit the target line.

1.9 Getting Connected

This section discusses getting connected through GalileoConnect with an RS-232 connection. It is recommended that before you start working with your touch panel, you connect with a terminal program to test and understand the connection process.

1.10 Connecting to ControlPoint

There are several considerations involved in getting connected to ControlPoint through GalileoConnect. You will need to use one of the following methods to connect to ControlPoint with GalileoConnect. GalileoConnect must **always** be **started** on and from the Wall Controller (or remote workstation), but once started you can connect and authenticate either from the Wall Controller or the serial device (touch panel).

1. Starting/connecting from the Start Menu
2. Starting/connecting from a Shortcut
3. Starting/connecting from the Command Line
4. Connecting and authenticating from the Serial Device

Once a new connection is established, the serial peer may authenticate itself in the **ControlPoint Server**. Sending the user name on one line, then the password on a second line accomplishes this authentication. The server responds with **OK** when successful and **ER** if the login is rejected.

The ControlPoint Server will break the TCP/IP connection if the authentication fails. You will need to issue another **CONNECT** command and authenticate again.

1.10.1 Starting from the Start Menu Icon

You can start GalileoConnect from the Start Menu Icon. Starting GalileoConnect from the Start Menu will connect GalileoConnect to ControlPoint and use the command line default arguments listed previously to connect and authenticate. Passing the user name and password to ControlPoint may also be accomplished from the serial device. Refer to information on connecting and authenticating in **Chapter 3 “Programming” on page 119**.

You can connect by running GalileoConnect on the Wall Controller and then send login information from the touch panel – either manually (touch button) or automatically if the touch panel can do that. If you have a smart touch panel, you may be able use its programmability to do your authentication.

The method described here is the preferred method to start GalileoConnect and get logged in and authenticated automatically every time the system is booted, as in the case of a power outage and restart.

1.10.2 Starting from a Shortcut

You can create a shortcut on your desktop to start GalileoConnect. Once created, it can then be dropped or copied to the Startup folder in the Start Menu. Go to **C:\Program Files (x86)\Jupiter\ControlPoint** right click on galcon.exe and drag it to the desktop. Select Create Shortcut Here.

The Target line in the shortcut properties can be edited to provide the login and authentication arguments as described in the following section on using the command line to connect.

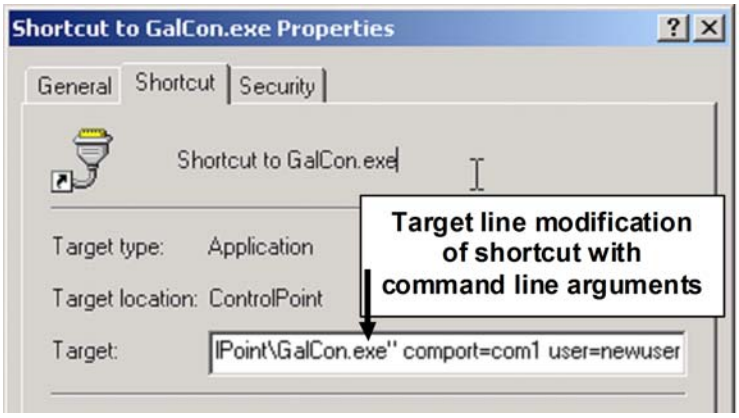


Figure 10 - Shortcut Dialog

Note that the command line arguments are typed **outside** (after) the quotation marks.

There is no password security here. If this method is used, anyone who has access to the Wall Controller system can view the shortcut properties, and with them, your login and password parameters.

Caution	No password **security** is provided if you use the shortcut method of starting GalileoConnect automatically from an icon in the **Startup** folder, or an icon on the desktop.

1.10.3 Starting from the Command Line

Use the following path to **Run** galcon.exe from the command line **C:\Program Files (x86)\Jupiter\ControlPoint\GalCon.exe**. Add the proper parameters as necessary to get logged onto the ControlPoint Server and authenticated. GalileoConnect command line options and the Connect command are discussed earlier in this section.

Example:

galcon.exe comport=COM1 server=WallServer port=25456 user=MyName pass=OpenSesame auto=true

Once you are connected and logged on, you may start passing commands from your serial device (touch panel).

Hint Click **Start** – **Run**, and type **cmd** to open a command line dialog box. This box is commonly referred to as a **DOS Box**.

Once you have started GalileoConnect, your system is connected and authenticated using either the default values or the values you entered on the command line.

1.10.4 Connecting from the Serial Device

Once GalileoConnect has been started on the Wall Controller, you may login and authenticate from the serial device. Many flat panel controllers have the ability to send this information automatically by waiting for confirmation of connection and then sending the authentication information. Also refer to **“Programming Considerations” on page 168**.

If you do not have a smart serial device or flat panel, you can make a set of buttons to connect and log in manually. The following sections discuss how to connect and login to the ControlPoint Server. While this information is not explicitly geared toward this application, it can easily be adapted to using touch buttons or other facilities in your serial device to send the same data strings necessary to logon and authenticate.

1.10.5 Testing Your Connection

The easiest way to test and verify the physical serial connection is to connect using HyperTerminal on both the Fusion system and second

Windows system. You should test by typing something on each keyboard to determine that the data is sent and received in both directions.

Once you have established serial communications in both directions, terminate HyperTerminal on your Wall Controller and run GalileoConnect. After you are connected and logged in, you can send ControlPoint commands and see the messages returned by the GalileoConnect Server in answer to these commands within the HyperTerminal window on the remote system. Refer to **“Understanding Command Structures ” on page 119** for more information on commands and response messages.

Use the settings shown below to configure both sessions of HyperTerminal.

Settings in HyperTerminal

Connect to - Configure

Baud 9600

data bits 8

parity none

stop bits 1

flow control none

Settings

Windows Keys

ASCII Setup

Send Line ends with line feeds

Echo typed characters locally

Append line feeds to incoming line ends

Wrap line that exceed terminal width

(Refer to **“The Connect Command” on page 25**)

Type a plus sign (+) and some random characters (simulated ControlPoint command) and you should get the response for a bad command.

=80040505

You have a good connection and you are working properly within the ControlPoint Server.

1.10.6 Authenticating the Connection

Authentication is always necessary with an IP connection. Authentication is only necessary under GalileoConnect if you are changing the login from the default values, as in changing the baud rate or other parameters. When

you start GalileoConnect from the command line, all defaults are used and you can start sending ControlPoint commands from your serial device.

Follow the steps below to login to ControlPoint after connecting with either IP or RS-232.

1. Type or send **user name** and Carriage Return / Line Feed (**Enter**) – default user name is **admin**
2. Type or send **password** and Carriage Return / Line Feed (**Enter**) - default password is no password (null string) (just press **Enter**)

   Default login is simply **admin Enter Enter**

   - default user name is **admin**
   - default password is **no password specified**
   - both of the above authentication parameters must be sent on separate lines followed by **<CR/LF>** (**Enter**)

   - Example:

     admin <CR/LF>
     <CR/LF>

1.11 Special Commands (RS-232)

There are two special commands for GalileoConnect that can be sent from the remote serial device:

Baud

Connect

These commands can be nested in the ControlPoint protocol data stream. Each command must start immediately at the beginning of a new text line. If the text is not a valid GalileoConnect command, it is relayed to ControlPoint. GalileoConnect will return a response to each command. This response will indicate either a success or a failure.

A success response is:

OK data

where data is optional and can be any resultant data associated with the command.

An error response is:

ER error_text

where error text is the explanation of the problem.

In all examples, **<CR><LF>** is the same as pressing the **Enter** key.

1.11.1 The Baud Command

BAUD <number> -

Set the baud rate for the serial connection. The baud rate is changed after the result of the operation is sent back to the serial device.

Supported baud rates are: **1200, 2400, 9600, 19200, 38400, 57600, 115200**.

If the command is successfully executed, GalileoConnect returns

OK new_baud_rate <CR><LF>

If the command was not successfully executed, GalileoConnect returns an error.

ER Invalid baud rate <CR><LF>

1.11.2 The Connect Command

The **Connect** command is used from the serial device to get or set the status of the TCP/IP connection to the ControlPoint Server.

Two Mode Operation

The **Connect** command operates in two distinct modes.

- Mode 1 - Without parameters – get status only - no authentication is required
- Mode 2 - With parameters – sets parameters – requires immediate authentication

Mode 1 - Get the status of the TCP/IP connection to the ControlPoint Server.

When Connect is used without parameters, it returns the current the status of the connection: **ControlPoint host name, TCP/IP port**, and

user name used for authentication. When used in this mode, no authentication is expected after use.

CONNECT
OK hostname 25456 username <CR><LF>

Mode 2 - Set the TCP/IP connection to the ControlPoint Server.

When Connect is used with any parameters, these parameters will be used and authentication will be expected immediately following on the next two lines.

CONNECT <CPServer> <PortNumber> <user> <password>
OK hostname 25456 username <CR><LF>

CPServer is the host name or IP address of the ControlPoint Server.
PortNumber is the TCP/IP port to connect to (default 25456).
User is the user name for authentication, **password** is the password for authentication.

A positive response is returned if the connection is ok,

OK hostname 25456 username <CR><LF>

A negative response is returned if the connection is broken.

ER hostname 25456 username <CR><LF>

If an asterisk **[*]** is specified for the server name (CPServer), the last server name supplied is used for the connection. In all examples, **<CR><LF>** is the same as pressing the **Enter** key.

ER * 25456 username <CR><LF>

When the Connect command is issued with parameters, it signals the ControlPoint Server that the next two sets of data are the user name and password, in that order. If a null string is supplied for the user name, GalileoConnect reuses the last user name and password supplied in the command line arguments when starting GalileoConnect. This is helpful when the serial device has **no** knowledge of users and passwords and you want to use the username/password supplied to GalileoConnect on start-up.

Example 1 – connect to server using all defaults:

connect <CR><LF>
OK localhost 25456 localuser <CR><LF>

Notes

—It is not necessary to send the **Connect** command in order to begin sending ControlPoint commands from the serial device to the ControlPoint server once the GalileoConnect Server has been started.

—It is advisable to send the **Connect** command and look for the **OK** returned to validate your connection.

—With no parameters added, **no** authentication is necessary as this was taken care of by the defaults when the **Connect** command was started. The next output from the serial device can be your first ControlPoint command.

—You must use the **Connect** command with the server name if you intend to authenticate with user name and password other than defaults. Refer to **Example 2** below.

—Whenever **any** parameter is entered after the **Connect** command, authentication is signaled and required immediately following.

Example 2 – connect to server CPServer on the default port 25456 and authenticate as user admin with password secret:

connect CPServer <CR><LF>
admin <CR><LF>
secret <CR><LF>

Example 3 – connect to server CPServer on port 123 and authenticate the default user supplied as a command line argument to GalileoConnect:

connect CPServer 123 <CR><LF>
<CR><LF>
<CR><LF>

Refer to **“Getting Connected with the ControlPoint Server” on page 168**.

After GalileoConnect receives the **Connect** command with the user name and the password, it tries to connect to the ControlPoint server and

authenticate the user. Upon success, GalileoConnect replies with a positive response, otherwise it returns a negative response.

If anything is sent after the **Connect** command that is NOT user name and password, you **must** reissue the **Connect** command with the server name in order to send login authentication again. GalileoConnect assumes that the next two lines are the user name and password, and will return an error if not correct. If you find that your login is rejected or you get other errors, then re-issue the **Connect** command, if you get an error, re-connect fully, if you get an OK, then send your authentication information to login. This process is very easily seen when using a terminal to view the process. Refer to **“Testing Your Connection” on page 22**.

1.12 Connecting to ControlPoint Using C#

To connect to ControlPoint Protocol using the C# application, run the following script:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;


namespace Jupiter
{
   /// <summary>
   /// Creates a connection between ControlPoint for communcation
   /// </summary>
   public class SocketConnection
   {   //http://msdn.microsoft.com/en-us/library/kb5kfec7.aspx
      /// <summary>
      /// Check if connection is active
      /// </summary>
      public bool IsConnected
      {
         get { return client.Connected; }
      }
      /// <summary>
      /// TCP socket connection
      /// </summary>
      private Socket client;
      /// <summary>
      /// Method for receiving data
      /// </summary>
      /// <param name="data">Received string from server</param>
```

```
        public delegate void SocketReceivedData(string data);
        /// <summary>
        /// Even is fired when data is received. Optional.
        /// </summary>
        public event SocketReceivedData ReceivedData;
        /// <summary>
        /// Credentials to login
        /// </summary>
        private string IP, Username, Password;
        /// <summary>
        /// PDC=30326. ControlPoint=25456
        /// </summary>
        private int Port;
        /// <summary>
        /// Buffer for receiving data
        /// </summary>
        private byte[] bytes = new byte[1024];
        /// <summary>
        /// Creates a TCP socket
        /// </summary>
        /// <param name="IP">IPv4 address</param>
        /// <param name="port">Port of server</param>
        /// <param name="Username">Username</param>
        /// <param name="Password">Password</param>
        public SocketConnection(string IP, int port, string Username, string
Password)
        {
            client = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
            client.ReceiveTimeout = 10000;
            this.IP = IP;
            this.Port = port;
            this.Username = Username;
            this.Password = Password;
        }
        /// <summary>
        /// Connects to server and sends username & password.
        /// </summary>
        /// <returns>Is connected to server</returns>
        public bool Connect()
        {
            client.Connect(IP, Port);
            if (!client.Connected)
                return false;
            byte[] msg = Encoding.UTF8.GetBytes(Username + "\r\n");
```

```
            int bytesSent = client.Send(msg);
            msg = Encoding.UTF8.GetBytes(Password + "\r\n");
            bytesSent = client.Send(msg);
            int bytesRec = client.Receive(bytes);
            string response = Encoding.UTF8.GetString(bytes,0,bytesRec);
            if(ReceivedData!=null)
               ReceivedData(response);
            if (response.Equals("OK"))
               return true;
            return false;
         }
         /// <summary>
         /// Releases and closes the socket
         /// </summary>
         public void CloseSocketConnection()
         {
            client.Shutdown(SocketShutdown.Both);
            client.Close();
         }
         /// <summary>
         /// Sends data to the server.  (Puts in the carriage return and line
feed automatically)
         /// </summary>
         /// <param name="data">data to be sent</param>
         /// <returns>returned data</returns>
         public string Send(String data)
         {
            byte[] msg = Encoding.UTF8.GetBytes(data + "\r\n");
            int bytesSent = client.Send(msg);
            int bytesRec = client.Receive(bytes);
            string sb = Encoding.UTF8.GetString(bytes, 0, bytesRec);
            while (client.Available != 0)
            {
               bytesRec = client.Receive(bytes);
               sb+=Encoding.UTF8.GetString(bytes, 0, bytesRec);
            }
            if (ReceivedData != null)
               ReceivedData(sb);
            return sb;
         }
      }
   }
```

1.13 The Protocol – Some Basics

This section is intended to help you get started by providing some general introductory information about the ControlPoint Protocol. A knowledge and understanding of the Binary and Hexadecimal number systems will be helpful in understanding these structures as they use the binary bit position as shown in the examples.

In all examples, **<CR><LF>** is the same as pressing the **Enter** key.

1.14 Syntax

The ControlPoint Protocol is text-based. Each command line terminates with **CR/LF.** Tokens are space separated. **All** identifiers are case-sensitive.

Your inputs start with a plus sign [+]. The replies from the server start with an equal sign [=]. All following examples have return codes included.

The following examples show proper replies for comparison. Refer to **"Error Codes" on page 175**.

1.14.1 Commands

A serial device can send two types of commands to GalileoConnect.

- ControlPoint Protocol Commands
- GalileoConnect Commands

1.14.1.1 ControlPoint Commands

Every ControlPoint command must start with the '+' character. This is how GalileoConnect knows that the command is to be relayed to the ControlPoint server for processing.

Example:

+WinServer QueryAllWindows <CR><LF>

1.14.1.2 GalileoConnect Commands

Any text line not starting with '+' is treated as a GalileoConnect command and processed by GalileoConnect. There are only two GalileoConnect commands: **Connect** and **Baud.**

Example:

connect <CR><LF>

1.14.2 ControlPoint and GalileoConnect Responses

This section discusses ControlPoint and GalileoConnect communication responses.
Possible responses to a ControlPoint command:

1. ControlPoint positive response if the ControlPoint server processed the command and successfully returned the result to GalileoConnect.
2. ControlPoint negative response if the command cannot be forwarded to the ControlPoint server or if the server failed to respond to the command.

ControlPoint Response
Every ControlPoint response starts with the equal [=] character. The serial device should expect such a response from every ControlPoint command processed by the server.

Example:
=00000000 <CR><LF> 'positive response
=80040505 <CR><LF> 'negative response

Possible responses to a GalileoConnect command

1. GalileoConnect positive response
2. GalileoConnect negative response

GalileoConnect Positive Response
GalileoConnect positive responses start with **OK**, followed by optional text data with the result of the command.

Example:
OK localhost 25456 localuser <CR><LF>

GalileoConnect Negative Response
GalileoConnect negative responses start with **ER**, followed by optional description of the error condition.

Example:
ER Socket error <CR><LF>

If GalileoConnect gets a command that is not a well-formed ControlPoint command, it matches it against its internal commands (**baud** and **connect**). If not either of these, a **Bad Command** error is returned.

Example:

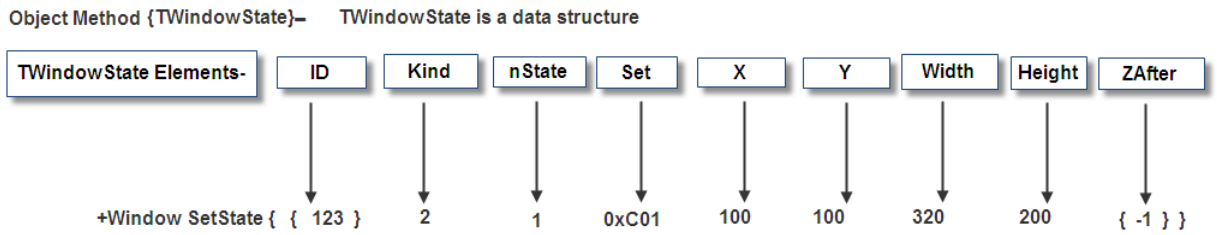
ER Bad Command <CR><LF>

1.14.3 ControlPoint Notifications

If the serial device registers to receive ControlPoint notifications, GalileoConnect will relay those from ControlPoint to the serial device. This can happen only in an idle state, when GalileoConnect is waiting for a command from the serial device or waiting for a ControlPoint response to a ControlPoint command. ControlPoint notifications always start with the colon **[:]** character and are easily distinguished from the normal command-response data flow. The serial device does not respond to such events.

1.15 Using Braces

All **Structures** must be in braces.



Note from the above that the ID is also a Structure as is ZAfter. These Structures must be within braces.

Many commands do not need braces because the argument is not a data structure.

+WinServer SetLayout “layoutname”

Many others may require at least one set and many times two or more sets of structures.

+LiveVideoSys SetChannel { ID } channel#
+LiveVideoSys SetVideoSource { ID } {CPLiveVideoSource}
+Window SetState { { 123 } 2 1 0xC01 100 100 320 200 { 1 } }

Note All braces must be preceded and followed by a space.

1.16 Window Numbering

ControlPoint provides for creating windows using either the ControlPoint Local Client or the ControlPoint Remote Client as well as a serial interface. Windows can be created with an automatic ID number or with **User-Assigned Window ID** numbers. The main purpose of **User-Assigned Window ID** numbers is for use with external serial devices. The use of **User-Assigned Window ID** numbers gives the user a way of controlling and keeping track of specific windows with their ID numbers. All instructions used by the touch panel controller or program to control specific windows on the display wall must refer to the window by its unique ID number. (Refer to the **ControlPoint Software Manual**).

All windows have a **unique** ID number. Windows created by the ControlPoint Server have Window ID numbers that are generated automatically by the server and are always **greater than 10,000**. User-Assigned ID numbers will always be **greater than 0** and **less than 10,000**. All windows are addressable by external control programs by using the window ID number. The ControlPoint software can address and control all windows.

Windows given ControlPoint generated window IDs (i.e. server-assigned IDs) will be given the **next** ID number available for the current session. If you were to open and close (delete) three windows, the next number will be 10,004. A session is the current running of the server after a reboot or the restarting of the server.

Window ID numbers can only be assigned for **User-Assigned Window ID Numbers**. If you are controlling windows from an external program or serial device, you **must** use windows **with User-Assigned ID Numbers** in order to insure you are controlling the correct or even an existing window.

1.17 Set Command Structure

The structure is:

Object Method { Set Elem1 Elem2 … }

Where **Set** validates subsequent elements in the command string. This same command structure is used in a more complex manner in the **SetState** method shown following. The **SetImgBalance** method will be used as an example for explaining this structure.

The **SetImgBalance** method sets the image balance of the specified LiveVideo or RGB window. The **SetImgBalance** method syntax and an example command are shown below.

+LiveVideoSys SetImgBalance { WinId } { Set Bri Con Gam Hue Sat }

+LiveVideoSys SetImgBalance { 101 } { 27 0 100 0 0 100 }

The **Set** element is used to validate all other elements in the subsequent list.

Table 2: Set Element Bit Positions

Set bit positions							
128	64	32	16	8	4	2	1
0	0	0	Sat	Hue	Gam	Con	Bri

In the following example, **Set** is equal to 27. This value, 27 is the sum of 16, 8, 2, and 1 (1+2+8+16=27). These values equate to the bit positions (powers of 2 in the table above) that are associated with the image balancing properties **Hue**, **Saturation**, **Contrast**, and **Brightness**. Consider the diagramed command below:

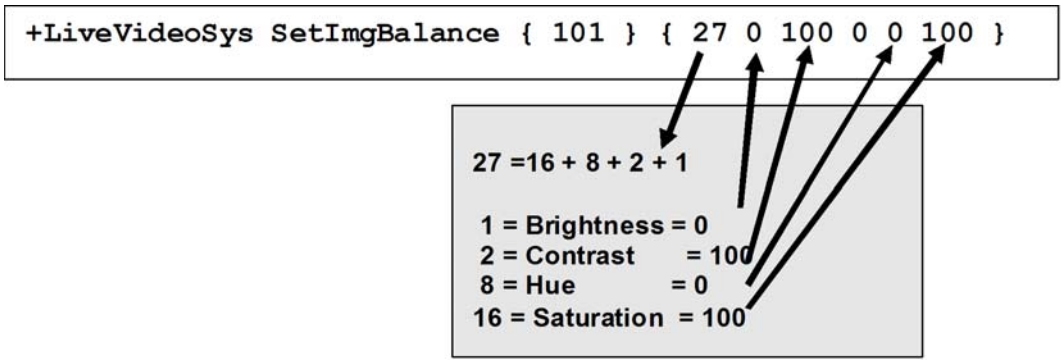


Figure 11 - SetImgBalance Method

Since **Gamma** is not used, **Set** with a value of 27 validates all four remaining elements in the list. The values shown are the default values for the parameters.

If you wanted only to set contrast and brightness (2 and 1) the **Set** element should be 24 as shown in the following program line which sets contrast to 7 and brightness to 93. All other values in the element list will be ignored except those specified by **Set**.

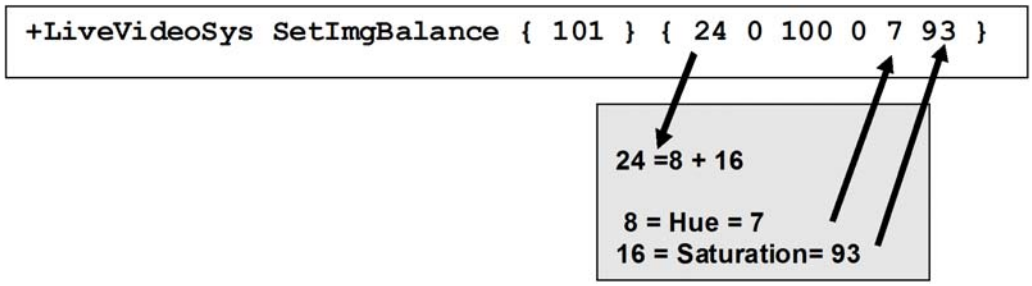


Figure 12 - Hue and Saturation

Hue will be set to a value of 7 and Saturation to 93.

The following section shows the SetState command, which uses the same structure shown here.

Note All data positions must have some value (0) as a placeholder for the data field.

1.18 Creating Windows

There are four steps required to create and display a window from scratch.

Hint	It is much easier to create your windows manually and save them as a layout and then operate on the windows using the protocol.

The list below shows the steps (Object and Method) required for the creation of a **LiveVideo** window.

1. LiveVideoSys NewWindowWithId
2. LiveVideoSys SetChannel
3. GalWinSys Start
4. Window SetState

These steps are shown in detail in **Chapter 3 – Programming**. **SetChannel** and **Start** will not be covered here as they are very simple commands to implement.

Let's look at a simple example of creating a LiveVideo window:

```
+LiveVideoSys NewWindow
=00000000 { 10001 }
```

Creating a window only *creates* it; it does not fill it nor make it visible.

Note	All newly created windows are hidden, have size and position (0,0), are given the User ID number, and are invisible (hidden).

The result shows success (**=00000000**). The return code includes the WindowID (**10001**) for this window.

To create windows with *known* and constant ID numbers, you must create windows with a specific ID number supplied by you. You will have to use this ID number whenever referring to this window. For example, to create a window with ID 123, you would send:

Step 1.

```
+LiveVideoSys NewWindowWithId { 123 }
=00000000
```

Step 2.

Select input channel 2:

```
+LiveVideoSys SetChannel { 123 } 2
=00000000
```

Step 3.

Activate the window with the **Start** command:

```
+LiveVideoSys Start { 123 }
=00000000
```

Step 4.

You can now use the **SetState** command to specify several window parameters once you have created the window as shown above. A window created with the **NewWindowWithId** command will always have this ID number associated with it.

Having created the window (123), we will now show it (it is still hidden) at coordinates 100,100 with size 320x200:

```
+Window SetState { { 123 } 2 1 0xC01 100 100 320 200 { -1 } }
=00000000 { { 10001 } 2 1 7183 100 100 320 200 { -1 } }
```

Now let's move the window to a new position: 1200, 100:

```
+Window SetState { { 123 } 0 0 0x400 1200 100 0 0 { -1 } }
=00000000 { { 10001 } 2 1 7183 1200 100 320 200 { -1 } }
```

Note the 0x400 in **Set** indicates we are only going to make the position (x, y) valid data. Refer to **Table 4 on page 46**.

As you can see, we have attempted to minimize the total number of commands, allowing you to do many things in this single command.

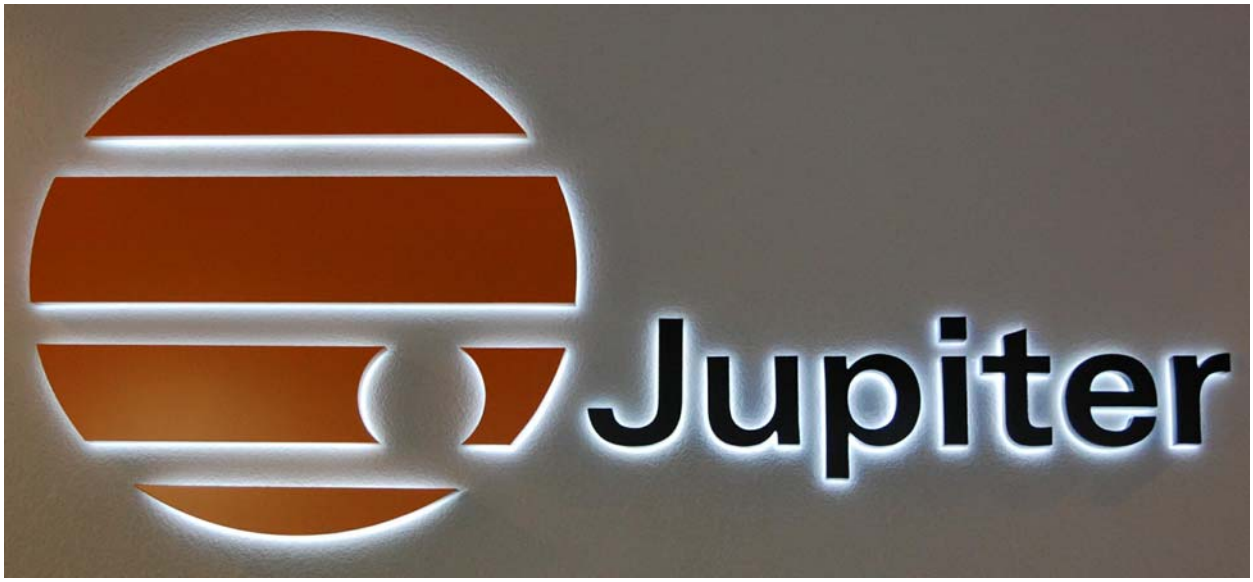
1.19 Extended Information

ControlPoint Error Codes and Structures

The protocol listing of return and error codes associated with the GalileoConnect instructions discussed in this manual can be found in **Appendix A – Error Codes**.

ControlPoint Commands

A full alphabetical listing of all Methods by Objects and Method can be found in **Appendix B and C – Command Listing**


Jupiter

Chapter 2—ControlPoint Protocol

2. ControlPoint Protocol

This chapter discusses the ControlPoint Protocol, its syntax, structure, and use.

2.1 Introduction

The ControlPoint logical protocol is developed on top of **RMC** (Remote Method Call) inter-process communication protocol, implemented over a TCP/IP connection.

RMC Protocol—Protocol Syntax
The RMC protocol is a text-based streaming protocol that allows peers to invoke methods of objects and receive asynchronous notifications. The peer that invokes a method is called **client** and the peer that services the method and returns the result of the execution is called **server**. For each client **request** there is a server **response**. Requests and responses are synchronous and a client cannot issue a new command before the current call completes. **Notifications/events** are generated by the server and are asynchronous with the request-response mechanism. Notifications are only one way and are not acknowledged by the client. Notifications can happen in-between a request-response transaction.

Please refer to **Chapter 3, "Programming" on page 119** for detailed examples of using the commands enumerated in this chapter.

2.2 Syntax

The **ControlPoint** protocol is text-based. Each command line terminates with **CR / LF**. Tokens are space separated. **All** identifiers are case-sensitive.

There are three types of command lines that can be sent with the protocol. Their first character determines the type of these commands:

1. A request which starts with a plus sign [+]
2. A response which starts with an equal sign [=]
3. A notification which starts with a colon [:]

Requests are data going *to* the Wall Controller System from a remote client or device.

Responses are incoming data notifications *from* the Wall Controller System. The request-response mechanism is synchronous. Every **Request** requires a **Response**.

Caution

A **Response** must be received before the next **Request** is sent.

Notifications are asynchronous, but cannot occur between a **Request** and the corresponding **Response**. Notifications are sent by the server to notify the client of an event (as in a window being moved).

The first character of the command line determines the type – **Request**, **Response**, or **Notification**.

request +<object name> <method> [<value list>]
response =<result code> [<value list>]
notification :<object name> <method> [<value list>]

Note, that the request and the notification syntax are similar. This is because a notification is semantically a remote method call (a request), but it does not get a response.

Any text line not starting with '+' is treated as a GalileoConnect command and processed by GalileoConnect.

connect <CR><LF>

2.3 Request Line

Every ControlPoint command must start with the plus character. This is how GalileoConnect knows if the command is to be relayed to the ControlPoint server for processing.

A **Request** command starts with a plus sign **[+]**, followed by an [object name], [method name] and [method arguments]. The method arguments maybe optional.

+[ObjectName] [MethodName] [Arguments]
+GalwinSys SetLayout "Layout-One"

2.4 Response Line

A **Response** starts with and equal sign **[=]**, followed by an 8-digit hex number of the result code, optionally followed by other response data. The result code should be treated as a 32-bit signed integer number in twos-compliment form. Positive values (and zero) mean success, negative values indicate a failure.

=00000000 [Arguments] (positive values are success responses
=80000000 [Arguments] (negative values will start with a Hex character of 8 or greater – MSB set)
(refer to **Appendix A**, **"Error Codes" on page 175**)

2.5 Notification Line

Notifications start with a colon **[:]** and follow the syntax of a request. Notifications are requests that do not get responses. You will see notification lines only when you turn on notify commands.

:[ObjectName] [MethodName] [Arguments]

2.6 Arguments and Data Types

The Argument list consists of argument tokens separated by a space. The following argument types are supported:

- **Integer number** –
  - a 32-bit integer number in decimal format;
  - a 32-bit integer number in hex format (starts with ***0x***) (0x123A) Decimal floating point number

- **String**, is a value surrounded by quotation marks - “ ” (“now is the time”)
- **Structure**, is a value surrounded by braces - { xx }
- **Array**, represented by a structure with the first field set to the number of elements to follow.
- **Boolean –** Boolean operators are either 0 or 1 (0 = false – 1 = true)
- **Bit Fields** - There are several special data constructs called **bit fields** used in the ControlPoint Protocol. One such set is **Flags**, which are described in the following section and elsewhere. The **Set** elements used in **CPLiveVideoSource** and **ImgBalance** are bit fields and are used similarly to the example shown below.

Note	All returned data is in base 10 format (decimal). Entered data may be decimal or base 16 (Hex) signified by the starting characters 0x. For example: 0x12AB3400

Bit Field Example:

ImageBalance uses five components (Brightness, Contrast, Gamma, Hue, and Saturation), which are each represented by a separate bit in the **Set** variable. If you want to set all five parameters, you must set all 5 bits. Gamma, bit 4 is not used and will be 0. 16 + 8 + 2 + 1 = 27.

Table 3: Set Variable Bit Positions

Bit Positions	0	0	0	16	8	4	2	1
	-	-	-	S	H	G	C	B
0 0 0 S H G C B				1	1	0	1	1

(0 0 0 S H G C B) = (0 0 0 1 1 0 1 1) = 27
S=Saturation - H=Hue - G=Gamma - C=Contrast - B=Brightness

+GalWinSys SetImageBalance { 123 } { Set Brightness Contrast Gamma Hue Saturation)

+GalWinSys SetImageBalance { 123 } { 27 100 100 0 100 100 }
note value 27 in the above example – this is the **Set** variable validating the four used data fields.

2.7 Data Structures

This section lists and describes the data types and structures.

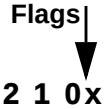
2.7.1 Flags

The following table lists the available flags. These flags are used for several arguments. The following **Set** flags will be used in the **Get/SetState** commands. Refer also to **"Understanding and Using Flags" on page 122**.

Table 4: Flags

Flags		
Get-Set Flags	wsVisible	0x0001
	wsMinimized	0x0002
	wsMaximized	0x0004
	wsFramed	0x0008
	wsLockAspect	0x0010
	wsAlwaysOnTop	0x0020
	wsPosition	0x0400
	wsSize	0x0800
	wsZOrder	0x1000
	wsTitle	0x2000
Notify Flags (read only)	wsKind	0x4000
	wsChannel	0x00010000
	wsBalance	0x00020000
	wsFormat	0x00040000
	wsCrop	0x00080000
	wsInput	0x00200000

Usage



+Window SetState { { 123 } 2 1 0x0C01 100 100 320 200 { 0 } } for **Set**, returned value of 7181 = 0x1C0D. Note bit positions for the following flags are set. **WsVisible**, **wsMaximized**, **wsFramed**, **wsPosition**, **wsSize**, and **wsZOrder**.

The table below shows the flags (listed on previous page) in their bit positions by hex digit.

Table 5: Flag Hex Positions

10000		1000		100		10		1	
80000	wsCrop	**8000**	na	**800**	wsSize	**80**	na	**8**	wsFramed
40000	wsFormat	**4000**	wsKind	**400**	wsPosition	**40**	na	**4**	wsMaximized
20000	wsBalance	**2000**	wsTitle	**200**	wsDestroyed	**20**	WsAlwaysOnTop	**2**	wsMinimized
10000	wsChannel	**1000**	wsZOrder	**100**	wsCreated	**10**	wsLockAspect	**1**	wsVisible

2.7.2 LiveVideoFormat

The following table lists the available LiveVideo Formats. These values will be used whenever you set the type of video input format.

Table 6: Live Video Format

LiveVideoFormat	
LIVE_VIDEO_NTSC	0
LIVE_VIDEO_NTSCJ	1
LIVE_VIDEO_PAL	2
LIVE_VIDEO_PALM	3
LIVE_VIDEO_PALN	4
LIVE_VIDEO_SECAM	5
LIVE_VIDEO_PALNc	6
LIVE_VIDEO_NTSC443	7
LIVE_VIDEO_PAL60	8

Usage

+LiveVideoSys SetVideoSource { 123 } { Set Format InputType }
+LiveVideoSys SetVideoSource { 123 } { 1 2 0 }

Note The Set argument determines which other arguments will be valid. 1 = Format 2 = Type 3 = Both.

2.7.3 LiveVideoType

The following table lists the available LiveVideo Types. These values will be used whenever setting the type of input source used.

Table 7: Live Video Type

LiveVideoType	
LIVE_VIDEO_COMPOSITE	0
LIVE_VIDEO_SVIDEO	1

Usage

+LiveVideoSys SetVideoSource { 123 } { Set Format InputType }
+LiveVideoSys SetVideoSource { 123 } { 2 0 1 }

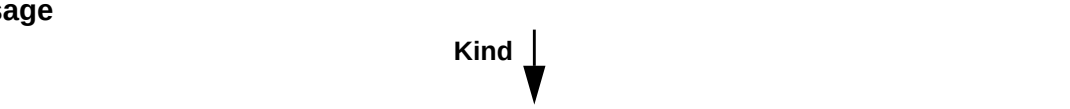
2.7.4 SubSystemKind

The following table lists the available SubSystemKind. SubSystemKind is used for several arguments. SubSystemKind is a read-only data type and cannot be used to set the type of a window.

Table 8: SubSystemKind

SubSystemKind	
SubSystemKind_None	0
SubSystemKind_Galileo	1
SubSystemKind_LiveVideo	2
SubSystemKind_RGBCapture	3
SubSystemKind_SystemWindow	4
SubSystemKind_CPShare	5
SubSystemKind_VidStream	6
SubSystemKind_CPWeb	7
SubSystemKind_PictureViewer	8
SubSystemKind_CatalystLink	9
SubSystemKind_IPStream	10

Usage



2.7.5 CaWndTitleHorizJust_t

The following table lists the available window title justifications.

Table 9: Ca Window Title Horizontal Justification

CaWndTitleHorizJust_t	Value
CaWndTitleHorizJust_Left	0
CaWndTitleHorizJust_Center	1
CaWndTitleHorizJust_Right	2

2.7.6 CaWndTitleVertJust_t

The following table lists the available window title vertical justifications.

Table 10: Ca Window Title Vertical Justification

CaWndTitleVertJust_t	Value
CaWndTitleVert_Top	0
CaWndTitleVert_VCenter	1
CaWndTitleVert_Bottom	2

Example:
The default value of zero for foth vertical and horizontal puts text at left and top.

2.7.7 CaWndTitleBarPos_t

The following table lists the available window bar positions.

Table 11: Ca Window Title Bar Position

CaWndTitleBarPos_t	Value
CaWndTitleBar_Top	0
CaWndTitleBar_VCenter	1
CaWndTitleBar_Bottom	2

2.7.8 WinId_t

The following table is a list of the available Data Structures, and their items:

Table 12: Data Structures

Type Name	Type	Elements
WinId_t_id	Long	_id

WinId_t_id

window ID number - >0 and <10,000 for User Defined ID numbers
window ID number - >10,000 for ControlPoint generated ID numbers

2.7.9 WinId_t_array_t

Table 13: Data Structures

Type Name	Type	Elements
WinId_t_array_t	Unsigned	nCount WinId_t pData[ ]

WinId_t_array_t array of window id's
nCount length of the array
WinId_t data name
pData [ ] data elements

Example of array:

Table 14: The WinID_t_array_t Array Example

Element	Window ID
0	10001
1	10002
2	123

2.7.10 TWindowState

Table 15: Data Structure - TWindowState

Type Name	Type	Elements
TWindowState	WinId_t	Id
	SubSystemKind_t	Kind
	unsigned	nState
	unsigned	nStateChange
	integer	x, y, w, h
	WinId_t	ZAfter

TWindowState

Id window ID number

Kind type of window – SubSystemKind [0, 1, 2, 3]

nState

Table 16: nState Bits

wsVisible	0x0001
wsMaximized	0x0004
wsFramed	0x0008
wsLockAspect	0x0010
wsAlwaysOnTop	0x0020

nStateChange refer to Flags - Used to specify valid data in named fields. nStateChange uses the above flags plus wsPosition, wsSize, and ZAfter. When all are set, you have 0x1C1F or 7193 decimal.

x, y X position, Y position – (0,0) to the size of wall

w, h width, height – (0,0) to the size of wall - bare image only

ZAfter WinId of window for this window to follow (-1=has focus and on top -2 for in back)

Note The values for X and W must be specified as even pixels, whereas, Y and H can accommodate both even and odd values.

Note Window SetState cannot be used to set the Kind (type) of a window. The Set field is used to validate all other data fields.

2.7.11 TWindowState_array_t

Table 17: Data Structure - TWindowState_array_t

Type Name	Type	Elements
TWindowState_array_t	unsigned	nCount TwindowState pData[ ]

TWindowState_array_t (array of window states)

nCount array size

TWindowState refer to TWindowState above

pData[ ] array elements

Example of array:

Table 18: The WinID_t_array_t Array

Element	ID								
0	10001	Kind	nState	nStateChange	x	y	w	h	ZAfter
1	10002	Kind	nState	nStateChange	x	y	w	h	ZAfter
2	123	Kind	nState	nStateChange	x	y	w	h	ZAfter

2.7.12 ImgBalance

Table 19: Data Structure - ImgBalance

Type Name	Type	Elements
ImgBalance	unsigned float	Set Brightness, Contrast, Gamma, Hue, Saturation

ImgBalance

Set * bit field – low order 5 bits (0001 1011)
Brightness brightness level – -100 to +100 (default = 0) (lsb)
Contrast contrast level – 0 to 100 (default = 100)
Gamma not supported - Set bit=0
Hue hue – 180 to +180 (default = 0)
Saturation saturation level – 0 to 200 (default = 100) (msb)
*Set argument is used to determine which parameters are to be changed (are valid).

Table 20: Image Balance Bit Positions

Set bit positions							
128	64	32	16	8	4	2	1
0	0	0	Sat	Hue	Gam	Con	Bri

2.7.13 CPLiveVideoSource

Table 21: Data Structures

Type Name	Type	Elements
CPLiveVideoSource	unsigned long long	Set Format InputType
CPRect	long	nX, nY, nW, nH
CPSize	long	cx, cy

CPLiveVideoSource

Set *	bit field – low order two bits (0000 0011)
Format	video format - (lsb)
InputType	video input type - (msb)

* The Set bits insure that the respective files contain valid data. Other fields may contain data but may not be valid.

Table 22: Input and Format Bit Positions

Set bit positions							
128	64	32	16	8	4	2	1
0	0	0	0	0	0	Inp	For

CPRect

nX, nY, nW, nH current rectangle (window) position and size

CPSize

cx, cy current size

2.7.14 CPRGBTiming

Table 23: Data Structure - CPRGBTiming

Type Name	Type	Elements
CPRGBTiming	boolean	bValid
	short	nWidth
	short	nHTotal
	short	nHOffset
	short	nHeight
	short	nVTotal
	short	nVOffset
	long	nPhase
	short	nVFreq
	long	nSyncType
	boolean	bHSyncNeg
	boolean	bVSyncNeg

CPRGBTiming

bValid	is valid – Boolean 0 or 1
nWidth	width of RGB image
nHTotal	horizontal total of RGB Image
nHOffset	horizontal offset
nHeight	height of RGB image
nVTotal	total vertical pixels
nVOffset	vertical offset
nPhase	RGB Pixel Phase
nVFreq	scan frequency
nSyncType	sync type
bHsynNeg	horizontal negative sync – Boolean – 0 or 1
bVsyncNeg	vertical negative sync – Boolean – 0 or 1

2.7.15 CPScreenConfig

Table 24: Data Structure - CPScreenConfig

Type Name	Type	Elements
CPScreenConfig	int	TotalWidth
	int	TotalHeight
	int	SingleScreenWidth
	int	SingleScreenHeight

CPScreenConfig

TotalWidth	total width of screen (horizontal resolution times displays)
TotalHeight	total height of screen (vertical resolution times displays)
SingleScreenWidth	horizontal screen resolution
SingleScreenHeight	vertical screen resolution

2.7.16 CPServerInfo

Table 25: Data Structure - CPServerInfo

Type Name	Type	Elements
CPServerInfo	unsigned long	dwVersionMS
	unsigned long	dwVersionLS
	unsigned long	dwFileTimeMS
	unsigned long	dwdwFileTimeLS

CPServerInfo

dwVersionMS CPServer Version (high bytes)
dwVersionLS CPServer Version (low bytes)
dwFileTimeMS Write time for CPServer (high bytes)
dwdwFileTimeLS Write time for CPServer (low bytes)

2.7.17 TString_array_t

Table 26: Data Structure – TString_array_t

Type Name	Type	Elements
TString_array_t	unsigned	nCount
	string	pData[ ]

TString_array_t

nCount array variable for size of array
pData[ ] array variable for array data

2.7.18 TCPSysMonECCInfo

Table 27: TCPSysMonECCInfo

Type Name	Type	Elements
TCPSysMonECCInfo	Valid	Boolean
	Long	SingleBitErrors
	Long	MultiBitErrors
	unsigned long	BlockAddr

TCPSysMonECCInfo

Valid	1 if the data in the structure is valid (0 if invalid)
SingleBitErrors	total number of single-bit errors since last boot.
MultiBitErrors	total number of multi-bit errors since last boot.
BlockAddr	address of the 4K block in which the last error occurred.

2.7.19 TCPSysMonValue

Table 28: TCPSysMonValue

Type Name	Type	Elements
TCPSysMonValue	Long	code
	long	value
	long	threshold
	int	status

TCPSysMonValue

code	the code of the monitored value
value	the value itself
threshold	threshold value that caused the alarm
status	status of the value

TCPSysMonValue codes

The **TCPSysMonValue** parameter code is a 32-bit value. The low order 24 bits contain the code of the parameter within the hardware module; the high order 8 bits contain the code of the hardware module (the location of the monitored parameter).

Table 29: TCPSysMonValue Codes

TCPSysMonValue - 32 bit Value																															
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01
Hardware Module Code								Parameter Code																							
128	64	32	16	08	04	02	01	8388608	4194304	2097152	1048576	524288	262144	131072	65536	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	08	04	02	01
07	06	25	04	03	02	01	00	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
80	40	20	10	8	4	2	1	10000000	8000000	4000000	2000000	1000000	800000	400000	200000	100000	80000	40000	20000	10000	8000	4000	2000	1000	800	400	200	100	80	40	20	10	8	4	2	1

Table Legend

Title
Bit position
Data Word Sections
Positional Value base 10
Power of 2
Hex value

Table 30: Hardware Module Codes

Hardware Module Codes
0 – Motherboard
1 – Main Chassis
2 – Expansion Chassis 1
3 – Expansion Chassis 2

Example

To illustrate the use of the Module and Parameter Codes, consider an installation with two chassis (a main chassis and an expansion chassis) and each chassis has a set of fans. To show values for each fan, the code for Fan 1 in the Main Chassis is 8192 + 0x01000000 = 0x01002000 [16785408] and the code for Fan 1 in Expansion Chassis 1 is 2048 + 0x02000000 = 0x02000800 [33562624]. All returned values are in decimal. This example shows the status of all three chassis fans at 0 RPM. (Status values other than zero indicate an alarm condition.)

The **status** field shows the **current status** of the monitored value:

0 – normal

1 – above high threshold. (Refer to **threshold** field for threshold value)

2 – below low threshold. (Refer to **threshold** field for threshold value)

3 – invalid measurement

+SysMonEx QueryAllValues

```
=00000000
{ 16 { 1 30 30 0 }        { 2 29 29 0 }            { 3 5921 5921 0 }
{ 4 6026 6026 0 }         { 5 1456 1456 0 }        { 6 1456 1456 0 }
{ 16 3248 3248 0 }        { 160 3312 3312 0 }      { 32 4731 4731 0 }
{ 48 11917 11917 0 }      { 64 -11410 -11410 0 }   { 176 1792 1792 0 }
{ 256 32 32 0 }           { 16779264 0 1000 2 }    { 16781312 0 1000 2 }
{ 16783360 0 1000 2 } }
```

2.7.20 SysMonValueIndex_array_t

SysMonValueIndex_array_t – an array of integers, specifying the value code.

This is the array used for entering parameters for the QueryValues method.

{ n codea codeb } { 2 1 3 } returns CPU-1 temperature and fan speed.

Where n is the number of elements (list of code values to return) and codex is the code of the parameter desired. The figure below is an axample of the array.

Table 31: The SysMonValueIndex_array_t Example

Element	SysMonValueIndex_array_t
0	2, 4, 17
1	3, 4, 17, 9
2	1, 17

2.7.21 TCPSysMonValue_array_t

TCPSysMonValue_array_t – an array of TCPSysMonValue Structures.

TCPSysMonValue

index the code of the monitored value
value the monitored value itself
threshold threshold value that caused the alarm
status status of the value

```
=00000000
{ 16 { 1 30 30 0 }     { 2 29 29 0 }            { 3 5921 5921 0 }
{ 4 6026 6026 0 }      { 5 1456 1456 0 }        { 6 1456 1456 0 }
{ 16 3248 3248 0 }     { 160 3312 3312 0 }      { 32 4731 4731 0 }
{ 48 11917 11917 0 } { 64 -11410 -11410 0 }  { 176 1792 1792 0 }
{ 256 32 32 0 }        { 16779264 0 1000 2 }    { 16781312 0 1000 2 }
{ 16783360 0 1000 2 } }
```

The above example has been formatted for ease of reading.

The first number (16) is the number of parameters returned in the array

=00000000 { 16 { 1 30 30 0 }...

1 = CPU1 temperature (code)
30 = 30 degrees C (value)
30 = 30 degrees C (threshold) – unless there is a fault, the threshold will always equal the value
0 = normal (status)

The parameter code is a 32-bit number. The low 24 bits contain the code of the parameter within the hardware module, the high 8 bits contain the code of the hardware module (the location of the monitored parameter).

Table 32: Hardware Module Codes

Codes	Hardware module
0	Motherboard
1	Main Chassis
2	Expansion Chassis 1
3	Expansion Chassis 2

2.7.22 TCPEventLogRecord

Table 33: TCPSysMonValue

Type Name	Type	Elements
TCPEventLogRecord	unsigned long	nEventID
	unsigned long	CPEventLogRecType
	unsigned long	nRecNumber
	unsigned long	nTime

nEventID the identifier of the event

{ header } “source” “text “
=00000000 { 1074003997 0 20 1047388718 }

nEvent ID nType
The header -nEventID = the identifier of the event
The nEventID field is the unique identifier of the event.

The Event ID is a 32-bit wide integer that encodes the identifier of the event. The format is:

```
 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+---+-+-+---------------------+-----------------------------+
|Sev| | |    Facility         |           Code              |
+---+-+-+---------------------+-----------------------------+
```

The most-significant 16 bits are OS dependant and should be ignored by the client. The actual alarm code is contained in the least-significant 16-bit word. Masking with 0x0000FFFF will get rid of the irrelevant data.

CPEventLogRecType_t nType = type of the event (severity)
The **nType** field shows the type (severity) of the event:
0 – Information message
1 – Error event
2 – Warning event

nRecNumber = the record number in the event log
The **nRecNumber** field shows the position of the record in the system event log
nTime = encoded time of the event
The **nTime** field has the encoded time of the event. It is the number of seconds elapsed since midnight (00:00:00) January 1, 1970.

2.7.23 CPWndFrameInfo

Table 34: Data Structure - CPWndFrameInfo

Type Name	Type	Elements
CPWndFrameInfo	unsigned	Set
	short	nFrameWidth
	DWORD	rgbFrameColor
	Boolean	bShowTitle
	Boolean	bShowUserData

CPWndFrameInfo

Set * bit field – low order 3 bits (0000 0111)
nFrameWidth frame with – 0 to 100 (default = 4)
rgbFrameColor frame color – RGB(R, G, B)
R/G/B - 0 to 255
bShowTitle indicates show/not show title (default = true)
bShowUserData indicates show/not show user data (default = false)

*Set argument is used to determine which parameters are to be changed (are valid).

Table 35: Window Frame Info Bit Positions

Set Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	0	SU	ST	FC	FW

2.7.24 CPWndTitleInfo

Table 36: Data Structure - CPWndTitleInfo

Type Name	Type	Elements
CPWndTitleInfo	unsigned	Set
	DWORD	rgbTextColor
	CPWndTitleBarPos	nBarPos
	short	nMinBarHeight
	CPWndTitleHorizJust	nHorizJust
	CPWndTitleVertJust	nVertJust

CPWndTitleInfo

Set * bit field – low order 5 bits (0001 1111)

rgbTextColor title text color – RGB(R, G, B)
R/G/B - 0 to 255

nBarPos 0 if title bar on top of the window, 1 if title bar at bottom (default = 0)

nMinBarHeight minimum height of the title bar – 0 to 100

nHorizJust 0 if text at the left of the title bar
1 if text at the middle of the title bar
2 if text at the right of the title bar
(default=0)

nVertJust 0 if text on the top of the title bar
1 if text at the middle of the title bar
2 if text at the bottom of the title bar
(default=0)

*Set argument is used to determine which parameters are to be changed (are valid).

Table 37: Window Title Info Bit Positions

Set Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	VJ	HJ	MH	BP	TC

2.7.25 CPWndTitleFontInfo

Table 38: Data Structure - CPWndTitleFontInfo

Type Name	Type	Elements
_CPWndTitleFontInfo	unsigned short unsigned wchar_t *	Set nTextSize nTextFlags pszFontName

_CPWndTitleFontInfo

Set * bit field – low order 3 bits (0000 0111)

nTextSize text size – 8 to 72

nTextFlags on input is a pointer to a string; on output is a pointer to a ps_string object (not used)

pazFontName font name

*Set argument is used to determine which parameters are to be changed (are valid).

Table 39: Window Title Font Info Bit Positions

Set Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	0	0	FN	TF	TS

2.7.26 ScreenTestPattern

Table 40: Data Structure - ScreenTestPattern

Type Name	Type	Elements
ScreenTestPattern	unsigned	Set
	short	nSingleScreen
	short	nBarType
	short	nGridCircle
	short	nFgColorRed
	short	nFgColorGreen
	short	nFgColorBlue
	short	nBgColorRed
	short	nBgColorGreen
	short	nBgColorBlue
	short	nSmoothGradient
	int	nLineSpacing

ScreenTestPattern

Set * bit field – 11 bits (0000 0111 1111 1111)

nSingleScreen
1 – single screen
0 – virtual screen

nBarType
0 – solid color
1 – vertical bars
2 – horizontal bars

nGridCircle
1 – grid drawn with circles
0 – grid drawn without circles

nFgColorRed
1 – red is used in foreground color
0 – red is not used in foreground color

nFgColorGreen
1 – green is used in foreground color
0 – green is not used in foreground color

nFgColorBlue
1 – blue is used in foreground color
0 – blue is not used in foreground color

nBgColorRed
1 – red is used in background color
0 – red is not used in background color

nBgColorGreen
1 – green is used in background color
0 – green is not used in background color

nBgColorBlue
1 – blue is used in background color
0 – blue is not used in background color

nSmoothGradient
1 – color pattern will be showed in smooth gradient;
0 – color pattern will be showed as bars

LineSpacing pixels between lines in grid pattern - 1 to 200

*Set argument is used to determine which parameters are to be changed (are valid).

Table 41: ScreenTestPattern Bit Positions

Set Bit Positions							
128	64	32	16	8	4	2	1
BG	BR	FB	FG	FR	GC	BT	SS
…	…	8192	4096	2048	1024	512	256
0	0	0	0	0	LS	SG	BB

2.7.27 MetadataRecord_t

Table 42: Data Structure – MetadataRecord_t

Type Name	Type	Elements
MetadataRecord_t	OLECHAR * OLECHAR *	propName propValue

MetadataRecord_t

propName Name of metadata
propValue Value of metadata

2.8 Objects and Methods

This section lists the ControlPoint Objects and their Methods. The following table is a listing of the Objects available within ControlPoint:

Table 43: Objects

Objects
AppCtrl
ConfigSys
CPShareSys
CPWebSys
Debug
EventLog
EventLogNotify
GalWinSys
GenieSubSystem
LiveVideoSys
Notify
PictureViewerSys
PixelNetMgr
RGBSys
ScreenUtil
SysMonEx
SysMonNotifyEx
UserMan
VidStreamSys
IPStreamSys
Window
WinServer

These Objects will be listed with their respective Methods in the following subsections. Also refer to alphabetical listing of Objects and Methods in **“Alphabetic Command Listing of Objects” on page 181** and **“Alphabetic Command Listing of Methods” on page 203**.

Note All in/out notations used in the following methods are from the server's point of view. **In to** the server, **out from** the server.

All in responses are data returned by **Get** commands, and are actual copies (copy/paste) from the appropriate command.

2.8.1 AppCtrl

Table 44: AppCtrl Object

Methods	
Exec	([in, string] CmdLine)

Exec executes the specified command-line on the CP Server. Note that all paths are server-relative. Program path is the path on the CP Server to the program you wish to run.

+AppCtrl Exec "program path"

2.8.2 ConfigSys

Table 45: ConfigSys Object

Methods	
GetServerInfo	([out] CPPlatformInfo, [out,string] versionInfo)
ListCfgGroup	([in] group_code, [out,string] objNames)

GetServerInfo returns information about the hardware platform and the version of the ControlPoint software installed on the CP Server.

GetServerInfo ([out] CPPlatformInfo, [out,string] versionInfo)
+ConfigSys GetServerInfo
=00000000 { PlatformCode ModelVersion ModelRevision OEMCode SerialNumber } "ControlPoint Version"
=00000000 { 19 0 1 0 4005 } "2.9.6414.296"
Returns information about the hardware platform and the version of the ControlPoint software installed on the server.

ListCfgGroup returns the object names in a configuration group. The configuration group types are:

1 – Named Inputs
2 – Application Windows
3 – Layouts
4 - CPShare
5 - HotKeys
6 - Schedule
7 - Video Stream Source
8 - Fonts
9 - Images
11 - Batch (The number 10 is not used)
13 - IPStream Source (The number 12 is not used)

The result is an array of strings.

+ConfigSys ListCfgGroup Type
=00000000 { 1 "my rgb" "rgb" "video" }
Listing of three named input objects.

2.8.2.1 Data Structure

CPPlatform Info contains platform code, model and revision version, OEM code, serial number, and ControlPoint Version.

The Platform Codes are:

18 - FC 1000
19 - FC 4000
20 - FC 8000
21 - FC 1100
22 - VizionPlus II
23 - FC4500

+ConfigSys GetServerInfo
=00000000 { Model, Model Version, Model Revision, OEM Code, Serial#, “ControlPoint Version” }
=00000000 { 19 0 1 0 4005 } "2.9.6414.296"

2.8.3 CPShareSys

Table 46: CPShareSys Object

Methods	
NewWindow	([out] WinId_t pwid)
NewWindowWithId	([in] WinId_t wid)
SetConnection	([in] WinId_t wid, [in] BSTR connection)
GetConnection	([in] WinId_t wid, [out,retval] BSTR * connection)
GetCredentials	([in] WinId_t wid, [out] VARIANT * server, [out] VARIANT * password)

The **CPShareSys** object and its method are used to set up CPShare windows.

Usage

A CPShare object must be previously manually created in the **Object Browser** within the CPClient.

NewWindow creates a new CPShare window and returns the window ID for that window. Server-assigned ID are > 10,000.

+CPShareSys NewWindow returns new window ID
=00000000 { 1234 }

NewWindowWithId creates a newCPShare window with a user-specified window ID. Returns user-assigned ID. User-assigned IDs are > 0 and < 10,000.

+CPShareSys NewWindowWithId { 123 }
=00000000 { 123 }

SetConnection sets up a connection object into a CPShare window. The connection object is referenced by its name and must be pre-defined with the Object Browser within the ControlPoint Client.

+CPShareSys SetConnection { 1 } "computer name" Sets a connection object "computer name" in CPShare window id = 1
=00000000

GetConnection returns the name of the connection object that the specific CPShare window is using.

+CPShareSys GetConnection { 1 } Queries CPShare window id = 1 for its currently selected connection object name
=00000000 “computer name”

GetCredentials returns the server host address and the password used to authenticate the CPShare connection.

+CPShareSys GetCredentials { 1 } Queries the connection authentication info used by CPShare window id = 1 to connect
=00000000 “computer name/IP address” “password”

Caution The **CPShareObject** must be created manually with the Object Browser in ControlPoint client.

2.8.4 CPWebSys

Table 47: CPWebSys Object

Methods	
NewWindow	([out] WinId_t * pwid)
NewWindowWithId	([in] WinId_t wid)
SetURL	([in] WinId_t wid, [in,string] BSTR srcURLAddress)
GetURL	([in] WinId_t wid, [out,string] BSTR * srcURLAddress)

The **CPWebSys** object and its method are used to set up CPWeb windows.
Usage

NewWindow creates a new CPWeb window and returns the window ID for that window. Returns the new window ID.

+CPWebSys NewWindow
=00010123

Note	You must use **“SetState ([in, out] TWindowState);” on page 199** to be able to view the Window. The default setting for this Window sets the size at 0,0 which renders the Window invisible.

NewWindowWithId creates a new CPWeb window with a user-specified window ID. Returns the user-assigned ID.

+CPWebSys NewWindowWithId { 123 }
=00000123

SetURL sets the URL address for a CPWeb window. The window ID must be specified. The URL address is a string.

+CPWebSys SetURL { 10007 } “www.jupiter.com”
=00000000

GetURL returns the current URL address of a CPWeb window. The window ID must be specified. Returns the URL of the current web page in the specified window.

+CPWebSys GetURL { 10023 }
=“www.jupiter.com”

2.8.5 Debug

Table 48: Debug Object

Methods	
GetLevel	([out] unsigned)
SetLevel	([in] unsigned NewLevel, [out unsigned OldLevel)
GetOutputs	([out] DWORD)
SetOutputs	([in] DWORD, new, [out] DWORD old)
GetFileName	([out] string)
CloseFile	( )
OpenFile	([in, string])
FlushToFile	([in, string])

The **Debug** object and its methods are used to help you debug your programs by allowing you to set up a logging file and the level of information logged.

2.8.6 EventLog

Table 49: EventLog Object

Methods	
SetPosition	([in] unsigned flags), ([in] unsigned long recNum)
GetRecord	([out] TCPEventLogRecord pRec), ([out, string] Source), ([out, string] EventText)
RegisterNotifyTarget	
UnregisterNotifyTarget	

EventLog is used to first set up the position of the pointer in the event log and then get the record. See mask bits that follow for setting position flags, and to turn on and off the change notification to the log.

Usage

+EventLog SetPosition flag recNum
+EventLog SetPosition 2 0

Sets the current position in the event log. The **recNumber** is the record number to position the record pointer, the **flag** bits determine the seek mode:

Bit 0 (mask 1) – go to the oldest record in the event log
Bit 1 (mask 2) – go to the latest (newest) event
Bit 2 (mask 4) – go to the record specified in **recNum**
Bit 3 (mask 8) – after positioning, the reading will be done in forward motion
Bit 4 (mask 16) – after positioning, the reading will be done in backward motion

Note **recNum** will only be valid for Bit2 (mask 4) but the position must be filled when sending the method. Failure to fill all fields will cause an error.

In order to retrieve a record you must first set the position in the record, then get the record.

2.8.7 EventLogNotify

Table 50: EventLogNotify Object

Methods	
NewEvent	([in] TCPEventLogRecord rec), [in, string] SourceName), [in, string] EventText)

You must implement this object to process event log notifications from the server.

Usage

+EventLogNotify NewEvent

The server calls this method on the client when a new event is generated. The arguments are the record header, the name of the source that generated the event and the text of the event message.

:**EventLogNotify { 1074003997 0 20 1047388718 }**
"GalSysMon" "Chassis Fan-3 speed back to normal. Current speed is 2463 RPM"

2.8.8 GalWinSys

Table 51: GalWinSys Object

Methods	
GetKind	([out] SubSystemKind_t)
IsOfKind	([in] WinId_t)
QueryAllWindows	([out] TWindowState_array_t)
NewWindow	([out] WinId_t)
NewWindowWithId	([in] WinId_t)
Start	([in] WinId_t)
Stop	([in] WinId_t)
Freeze	([in] WinId_t)
SetCrop	([in] WinId_t wid, [in] struct CPRect * pRect)
SetOrigin	([in] WinId_t wid, [in] long x, [in] long y)
GetCrop	([in] WinId_t wid, [out] struct CPRect * pRect)
GetImgBalance	([in] WinId_t, [out] ImgBalance)
SetImgBalance	([in] WinId_t, [in, out] ImgBalance)
GetInputSize	([in] WinId_t, [out] CPSize)
ApplyDefaults	([in] WinId_t)
QueryAllInputsCS	([out])
SelectInput	([in] WinId_t wid, inputName)
GetInput	([in] WinId_t wid, [out, inputName])

The **GalWinSys** object and its methods can operate on all window types (refer to **SubSystemKind**) and are inherited by **LiveVideoSys** and RGBSys. While GalWinSys can operate on all window types, **LiveVideoSys**

and **RGBSys** can only operate on their respective (and proper) window types.

Usage

+GalWinSys GetKind { WinId }
returns the SubSystemKind this Object can operate on. When used with LiveVideoSys or RGBSys returns SubSystemKind for respective Object.
=00000000 { 3 }

+GalWinSys IsOfKind { WinId }
returns either 0 or 1 (1= success but false) for question 'Can this Object operate on this window?' When used with LiveVideoSys or RGBSys returns 0 or 1 (true or false) for type of window queried.
=00000000 { 1 }

+GalWinSys QueryAllWindows
returns { nCount TwindowState pData[ ] } Id Kind nState nStateChange x y w h ZAfter
=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } }
{ { 102 } 3 1 7183 152 21 320 240 { 102 } }

+GalWinSys NewWindow
returns new window ID
= { 00010123 }

+GalWinSys NewWindowWithId { 123 }
Creates new window with specified ID number.
= { 00000123 }

+GalWinSys Start { 123 }
Starts capturing
=00000000

+GalWinSys Stop { 123 }
Stops capturing
=00000000

+GalWinSys Freeze { 123 }
Freezes the current frame
=00000000

+GalWinSys GetCrop { 123 }
Returns nX nY nW nH
=00000000 { 27 0 100 0 0 100 }

+GalWinSys SetOrigin { 123 } x y
Sets origin of cropped image to effect panning.
=00000000

+GalWinSys SetCrop { 123 } {nX nY nW nH }
Crop works in relation to the normal image size. nX and nY specify the pixels removed from the left and top. nW and nH specify the size of the cropped image. A 640x480 window set at 10 10 600 400 will have 10 pixels removed from the top, 10 pixels removed from the left, 30 and 70 removed from the right and bottom. ImageWidth - nX - (right pixels) = nW or 640 – 10 – 30 = 600.
=00000000

+GalWinSys GetImageBalance { 123 }
Returns { Set Brightness Contrast Gamma Hue Saturation } values of the specified window.
=00000000 { 123 } { 27 0 100 0 0 100 }

+GalWinSys SetImgBalance { 123 } { 27 0 100 0 0 100 }
Sets {Set Brightness Contrast Gamma Hue Saturation } values of the specified window. Uses **Set** to determine valid data values.
=00000000

+GalWinSys GetInputSize { 123 } Returns cx cy, of the specified window size.
=00000000 cx cy

+GalWinSys ApplyDefaults { 123 }
Sets the default values of the specified window for brightness (0), contrast (100), hue (0), saturation (0). Also, the cropping values are reset.
=00000000

+GalWinSys QueryAllInputsCS
Returns a list of named Input Names.

+GalWinSys SelectInput { WinId } “InputName”
Uses the **InputName** of a Named Input as a template for the properties of the specified window.

+GalWinSys GetInput { 123 }
Returns the name of a Named Input associated with the specified window.

2.8.9 GenieSubSystem

Table 52: GenieSubSystem Methods

Methods	
QuerySources	([out] /*StringArray_t* / unsigned char ** pInputs)
SetSource	([in] WinId_t wid, [in,string] wchar_t * inputName)
GetSource	([in] WinId_t wid, [out,string] wchar_t * inputName)
GetParameters	([in] WinId_t wid, [in] CatalystLinkParamId_array_t ids, [out] CatalystLinkParam_array_t * params)
SetParameters	([in] WinId_t wid, [in] CatalystLinkParam_array_t params)
DetectTiming	([in] WinId_t wid,)
SetRefreshClass	([in] WinId_t wid, [in] CPRGBRefreshClass_t rgbcls)
GetRefreshClass	([in] WinId_t wid, [out] CPRGBRefreshClass_t * rgbcls)
GetPixNetInfo	([in] WinId_t wid, [out] CatalystLinkWndInfo_t * info)

The GenieSubSystem object and its method are used to set up PixelNet windows. OutputNode Name is equivalent to Window Name. For all ger and query commands, you can use the last three bytes of the MAC address as a string (“1A 3B 2C”) or the Node Name. Where no Node Name has been configured for a node these commands will return the last three bytes of the MAC address as a string.

Usage

QuerySources lists all of the PixelNet input sources on-line
+GenieSubSystem QuerySources
=00000000 “source1” “source2” ... Returns a string list of the source names.

SetSource sets as source based on the Window ID, windowcharacteristics, and input name.
+GenieSubSystem SetSource Camera1 {30002 960 120 720 450}
=00000000

GetSource returns the source information based on the Window ID, windowcharacteristics, and input name.
+GenieSubSystem GetSource Camera1 30002
=00000000 Camera1 {30002 960 120 720 450}

GetParameters returns the display parameters for an input node associated with the specified window. Refer to **Table 53 on page 82** for a list of CatalystLink Parameter IDs. The following example returns the **Active Width (10)** and **Active Height (11)**.

+GenieSubSystem GetParameters ([in] WinId_t wid, [in] CatalystLinkParamId_array_t ids,
[out] CatalystLinkParam_array_t * params)

+GenieSubSystem GetParameters 30002 { 2 10 11 }
= 00000000 30002 { 2 { 10 1024 } { 11 768 } }
Returns the number of elements, then a list of parameter IDs and their values. Use **{ 0 }** to list all parameters.

SetParameters sets the display parameters for an input node associated with the specified window. Refer to **Table 53 on page 82** for a list of CatalystLink Parameter IDs.

+GenieSubSystem SetParameters ([in] WinId_t wid,
[in] CatalystLinkParam_array_t params)

+GenieSubSystem SetParameters { 30002 } { 2 10 11 }
The above command sets the Active Width and Active Height of the specified window. Refer to **Table 53 on page 82** for a list of CatalystLink Parameter IDs.

DetectTiming returns the current status of the auto detect flag for the specified window.
+GenieSubSystem DetectTiming { 123 }
=00000000 1
Returns auto detect setting (1/0) of specified window

Refresh Class sets the Frames Per Second (FPS) refresh rate for the window.
FPS Classification:
0 = 60 FPS
1 = 30 FPS
2 = 15 FPS
3 = 2 FPS

+SetRefreshClass ([in] WinId_t wid, [in] CPRGBRefreshClass_t rgbcls)

+SetRefreshClass 30002 {0 }
=00000000

Returns maximum refresh rate of specified window.
+GetRefreshClass ([in] WinId_t wid, [out] CPRGBRefreshClass_t * rgbcls)
=00000000 30002 {0 }

GetPixNetInfo returns Window ID and window information.
GetPixNetInfo ([in] WinId_t wid, [out] CatalystLinkWndInfo_t * info)
=00000000 30002 {0 0 1024 768 }

Table 53: Display Parameter IDs

CatalystLink Parameters	ID
idValid_Signal	1
idPreferedInterface	2
idAutodetect_Enabled	3
idActive_Interface	4
idImage_Width	5
idImage_Height	6
idPhase	7
idHTotal	8
idVTotal	9
idHActive	10
idVActive	11
idHBackPorch	12
idVBackPorch	13
idSync_Type	14
idHSync_Polarity	15
idVSync_Polarity	16
idHSync_Rate	17
idVSync_Rate	18
idBrightness	19
idBrightnessR	20
idBrightnessG	21
idBrightnessB	22
idContrast	23
idContrastR	24
idContrastG	25
idContrastB	26
idHue	27
idSaturation	28
idHigh_Definition	32
idHSyncWidth	34
idVSyncWidth	35
idAutoPhase	36
idAutoDE	37
idInterlaced	38
idBlackLevel	39
idBlinkLED	40

2.8.10 LiveVideoSys : GalWinSys

Table 54: LiveVideoSys : GalWinSys Object

Methods	
SetChannel	([out] WinId_t, [in] short)
GetChannel	([in] WinId_t, [in] short)
SetVideoSource	([in] WinId_t, [in] CPLiveVideoSource)
GetVideoSource	([in] WinId_t, [out] CPLiveVideoSource)
GetChannelRange	([out] short firstChannel, [out] short lastChannel)

The **LiveVideo System** object and its methods are used to set and get video parameters. These objects inherit **GalWinSys** and its Methods. All **GalwinSys** Methods can be used with **LiveVidowSys**.

Usage

+LiveVideoSys SetChannel { 123 } 1 Sets window 123 to channel 1

+LiveVideoSys GetChannel { 123 } Returns current Channel

+LiveVideoSys SetVideoSource { 123 } Set Format Type

+LiveVideoSys SetVideoSource { 123 } { 3 2 0 } Set Format and Type (Set=3) to PAL (2) Composite (0)

+LiveVideoSys GetVideoSource { 123 } Returns Set Format Type
=00000000 { 3 2 0 }

+GetChannelRange
=00000000 1 32 Returns the first and the last channel number for the specified subsystem

Note When using the SetChannel Method with either the LiveVideSys or RGBSys Objects, you define the type of window (either RGB or LiveVideo) by the type of input it receives.

When first created with the **NewWindow** command, the window will be hidden. The window will have size (0,0) and position (0,0).

2.8.11 Notify

Table 55: Notify Object

Methods	
WindowState	([in] TWindowState_array_t)
ScreenConfigChanged	([in] CPScreenConfig)

The **Notify Target** object and its methods are used to return **Notify** messages of changes made to windows or the configuration.

Usage

+Notify WindowsState { nCount TWindowState pData[ ] }
Id Kind nState nStateChange x y w h ZAfter

+Notify ScreenConfigChanged { TotalWidth TotalHeight SingleScreenWidth SingleScreenHeight }

Execute the following command if you want the remote device to receive notifications whenever there is a change to ControlPoint windows.

+WinServer RegisterNotifyTarget
=00000000

After this command is sent, ControlPoint server will send asynchronous notifications to the device, indicating changes of window status, etc.

Notifications are sent in the following format:

:Notify WindowsState { 1 { { 10002 } 2 1 5120 254 244 684 507 { -1 } }}

The serial device does not respond to the notifications messages. They are just being broadcasted by ControlPoint server to the interested parties.

2.8.12 PictureViewerSys

Table 56: PictureViewerSys Object

Methods	
NewWindow	([out] WinId_t * pwid)
NewWindowWithId	([in] WinId_t wid)
ShowPicture	([in] WinId_t wid, [in,string] BSTR srcFilePath)
GetFileName	([in] WinId_t wid, [out,retval] BSTR * srcFileName)
SetTextMode	([in] WinId_t wid, [in] int textmode)
GetTextMode	([in] WinId_t wid, [out] int *textmode)
SetText	([in] WinId_t wid, [in,string] wchar_t * text)
GetText	([in] WinId_t wid, [out,string] wchar_t ** text)

The **PictureViewerSys** object and its method are used to set up PictureViewer windows.

Usage

NewWindow creates a new PictureViewer window and returns the window ID for that window.

+PictureViewerSys NewWindow (returns new window ID)
=00010123

NewWindowWithId creates a new PictureViewer window with a user-specified window ID.

+PictureViewerSys NewWindowWithId { 123 }
=00000123

ShowPicture displays an image from the specified file path on the server. The window ID must be specified. The file path is a string

which does not contain a backslash. Use forward slash or double backslash, instead.

+PictureViewerSys ShowPicture { 10007 } “C:/MyPictures/APicture.bmp”

GetFileName returns the file path for the current picture on a PictureViewer window. The window ID must be specified.

+PictureViewerSys GetFileName { 10023 }

SetTextMode sets the PictureViewer text mode to on, off, or on with text scroll capability. The window ID must be specified.

+PictureViewerSys SetTextMode {10004} 1

where 0 = text mode off; 1 = text mode on; 2 = text mode on with text scroll

=00000000

GetTextMode returns the PictureViewer window’s (using WinId) text mode status.

+PictureViewerSys GetTextMode {10004}

where 0 = text mode off; 1 = text mode on; 2 = text mode on with text scroll

=00000000 1

SetText sets the entered text on the PictureViewer window. The window ID must be specified.

+PictureViewerSys SetText {10004} “Front”

=00000000

GetText returns the entered text on the PictureViewer window (using WinId).

+PictureViewerSys GetText {10004}

=00000000 “Front”

2.8.13 PixelNetManager

Table 57: PixelNetManager Methods

Methods	
QueryDevices	([out] /*CPPixNetDeviceInfo_array_t */ unsigned char ** pDevices)
GetDeviceMetadata	([in,string] wchar_t * devName, [out] /*Dictionary_t*/ unsigned char ** metadata)
SetDeviceMetadata	([in,string] wchar_t * devName, [in] /*Dictionary_t*/ unsigned char * metadata)
QueryRunningStatus	([in,string] wchar_t * devName, [out] int * pStatus)
QuerySysMonInfo	([in,string] wchar_t * devName, [out] DeviceSysMonInfo_t *SysMonInfo)
ResetDevice	([in,string] wchar_t * devName)
GetParameters	([in,string] wchar_t * devName, [in] CatalystLinkParamId_array_t ids, [out] CatalystLinkParam_array_t * params)
SetParameters	([in,string] wchar_t * devName, [in] CatalystLinkParam_array_t params)
DetectTiming	([in,string] wchar_t * devName)

The PixelNetManager object and its methods are used to manage PixelNet windows on the display wall. PixelNet Manager is a utility to manage PixelNet nodes. OutputNode Name is equivalent to Window Name. For all get and query commands, you can use the last three bytes of the MAC address as a string (“1A 3B 2C”) or the Node Name. Where no Node Name has been configured for a node these commands will return the last three bytes of the MAC address as a string.

Usage

QueryDevices returns the names of all the PixelNet devices attached to the system.

+ PixelNetManager QueryDevices ([out] /*CPPixNetDeviceInfo_array_t*/ unsigned char ** pDevices) = 00000000 “Node1” “Node2” “Node3”...

GetDeviceMetadata returns the metadata of a PixelNet input device. The device ID must be specified. Refer to **“MetadataRecord_t” on page 67** for the structure of MetadataRecord_t.

+ PixelNetManager GetDeviceMetadata ([in,string] wchar_t * devName, [out] /*Dictionary_t*/ unsigned char ** metadata)

+PixelNetManager GetDeviceMetadata { 27 }
=00000000 “NodeName” “userdata1” “userdate2”…
Returns a list of metadata associated with specified node, first item is node name.

SetDeviceMetadata sets the metadata to a PixelNet input device. The device ID must be specified.

+ PixelNetManager SetDeviceMetadata ([in,string] wchar_t * devName, [in] /*Dictionary_t*/ unsigned char * metadata)

+ PixelNetManager SetDeviceMetadata { 27 } “NodeName” “userdata1” “userdate2”…
=00000000

Writes a list of metadata to the specified node, first item is node name. UserData examples might be location, type of node, input source, etc.

QueryRunningStatus ([in,string] wchar_t * devName, [out] int * pStatus)
+PixelNetManager QueryRunningStatus “OutputNodeName” “associated windowname”
=00000000 1

QueryRunningStatus returns the status of output node—whether it has a window associated with it (i.e. is there a window on this display?)
Response 1 = Yes, a windowassociated with the output node is active;
Response 0 = No, there is no active window associated with the output node.

QueryRunningStatus returns the status of parameters used in system monitoring.

QuerySysMonInfo ([in,string] wchar_t * devName, [out] DeviceSysMonInfo_t *SysMonInfo);
+PixelNetManager QuerySysMonInfo “fans”
=00000000

In the following example, the returned values indicate the status of three chassis fans at 0 RPM.
{ 3 { 1 30 30 0 } { 2 29 29 0 } { 3 5921 5921 0 }

The first number (3) is the number of parameters returned in the array

=00000000 { 3 { 1 30 30 0 }...

1 = CPU1 temperature (code)
30 = 30 degrees C (value)
30 = 30 degrees C (threshold) – unless there is a fault, the threshold will always equal the value

0 = normal (status)

ResetDevice resets the input device (node). The device ID must be specified. By resetting the node, none of the current node settings will be affected. However, reset causes the node to stop processing the existing layout.
+ PixelNetManager ResetDevice ([in,string] wchar_t * devName)
+ PixelNetManager ResetDevice “NodeName”
=00000000

GetParameters returns the display parameters for an input node. Refer to **Table 53 on page 82** for a list of CatalystLink Parameter IDs.

+PixelNetManagerGetParameters ([in,string] wchar_t * devName, [in] CatalystLinkParamId_array_t ids, [out] CatalystLinkParam_array_t * params)

+PixelNetManager GetParameters “NodeName”
= 00000000 { 2 { 10 1024 } { 11 768 } }
The above command returns the HActive (active horizontal pixel value) and VActive (active vertical pixel value) values of the node.

SetParameters sets the display parameters for an input node.
+PixelNetManager SetParameters ([in,string] wchar_t * devName, [in] CatalystLinkParam_array_t params)

+PixelNetManager SetParameters “NodeName” { 2 { 10 1024 } { 11 768 } }
= 00000000
Sets the HActive (active horizontal pixel value) and VActive (active vertical pixel value) values of the node.

DetectTiming reports whether the auto-detect function is enabled on a node. The auto-detect function, when enabled, forces a lookup of VESA timing whenever the HActive, VActive, or VSync Rate properties change.

DetectTiming ([in,string] wchar_t * devName)
+PixelNetManager DetectTiming "NodeName"
=0000000 1
Returns the auto detect setting (1/0) of a specified node. 1 = auto-detect is enabled; 0 = auto-detect is disabled.

2.8.14 RGBSys: GalWinSys

Table 58: RGBSys : GalWinSys Object

Methods	
SetChannel	([out] WinId_t, [in] short nChannel)
GetChannel	([in] WinId_t, [out] short * pnChannel)
GetChannelRange	([out] short * FirstCh, [out] short * LastCh)
SetTiming	([in] WinId_t, [in] CPRGBTiming * pTiming)
GetTiming	([in] WinId_t, [out] CPRGBTiming) * pTiming)
DetectTiming	([in] WinId_t, [out] CPRGBTiming) * pTiming)
SetAutoDetectTiming	([in] WinId_t wid, [in] Boolean bEnable)
GetAutoDetectTiming	([in] WinId_t wid, [out] Boolean * bEnabled)
SetRCServer	([in] WinId_t wid, [in, string] wchar_t * serverName)
GetRCServer	([in] WinId_t wid, [out, string] wchar_t ** serverName)
SetDualLink	([in] WinId_t wid, [in] Boolean bEnable)
GetDualLink	([in] WinId_t wid, [out] Boolean * bEnabled)
SetComponent	([in] WinId_t wid, [in] Boolean bEnable)
GetComponent	([in] WinId_t wid, [out] Boolean * bEnabled)

The **RGB System** object and its methods are used to set and get RGB parameters. These objects inherit **GalWinSys** and its Methods. All **GalWinSys** Methods can be used with **RGBSys**.

Usage

+RGBSys SetChannel { 102 } 1
sets window 123 to RGB channel 1

+RGBSys GetChannel { 102 }
returns channel number
+RGBSys GetTiming { 102 }
Returns **{ bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg } =00000000 { 1 640 480 144 480 525 35 3 60 0 0 0 }**

+RGBSys SetTiming { WinId } { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }

+RGBSys DetectTiming { WinId }
Forces resample of input signal - Returns **{ bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg } =00000000 { 1 640 480 144 480 525 35 3 60 0 0 0 }**

+RGBSys SetAutoDetectTiming { WinId } 1
enable/disable (1/0) auto-detection of RGB timing

+RGBSys GetAutoDetectTiming { WinId }
Returns above setting (1/0)

Note The **+RGBSys DetectTiming { window_id }** command can be used to make ControlPoint detect the VGA signals faster by forcing it from the script instead of waiting for auto-detect to execute.
Call **DetectTiming** for every VGA window. Either dynamically query the Window IDs or use user-defined IDs in layouts so that Window IDs are familiar.

+RGBSys GetChannelRange
=000000000 1 32
Returns the first and the last channel numbers for the sub-system.
+RGBSys SetRCServer { WinId } NewRCServer
Sets the remoter server computer name or IP Address.

+RGBSys GetRCServer { WinId }
Returns above setting
=00000000 { NewRCServer }

+RGBSys SetDualLink { WinId } 1
Sets either Dual or Single Link (1 for Dual Link and 0 for Single Link)

+RGBSys GetDualLink { WinId }
Returns above setting
=00000000 { 1 }

+RGBSys SetComponent {WinId} 1
Sets either Component or RGB (1 for SetComponent and 0 for RGB)

+RGBSys GetComponent { WinId }
Returns above setting
=00000000 { 1 }

2.8.15 ScreenUtil

Table 59: ScreenUtil Object

Methods	
ShowPattern	([in] BSTR pattern_name)
SetPatternProp	([in,string] wchar_t * prop_name, [in] struct ScreenTestPattern * stp)
GetPatternProp	([in,string] wchar_t * prop_name, [out] struct ScreenTestPattern * stp)

The ScreenUtil object and its methods are used for screen tests.

Usage

The Screen Utility is use to setup seven different test patterns five of which have several options allowed. All patterns except the Screen ID can be displayed either across the whole wall (VirtualScreen) or on individual displays (Single Display). Zero or one must be use as a place holder for options not used. See Usage below.

There are two steps required for using the Screen Utility.
First you must open the respective utility page (Show Pattern)
Second you can then setup and make the changes you wish to display (Set Pattern Properties).

Use the commands below to select the function you want.

Show Pattern

“stdpat“ - display the Standard test pattern
“screenid“ - display the screen ID
“phasepat“ - display the Phase pattern
“bitmap“ - display a bitmap image
“colorpat“ - display the Color Bar pattern
“colorpat“ - display the Grid pattern
“” - the null function terminates the screen test

+ScreenUtil SetPatternProp “setpatternprop”
{ Set SingleScreen BarType GridCircle FgColorRed FgColorGreen FgColorBlue BgColorRed BgColorGreen BgColorBlue SmoothGradient LineSpacing }
+Object Method “function“

+ScreenUtility ShowPattern “colorpat”
Displays the color pattern page.

2.8.15.1 Show Pattern Properties

Once you have opened the function, you can make the changes to view the pattern you wish.

Table 60: Show Pattern Properties

Bit	Function	Description
1	SingleScreen	0 = Single Screen 1 = Full Screen (VirtualScreen)
2	BarType	0 = solid Color 1 = Vertical Bars 2 = Horizontal Bars
4	GridCircle	0 = Grid drawn without circles 1 = Grid drawn with circles
8	FgColorRed	0 = foreground color red off 1= foreground color red on
16	FGColorGreen	0 = foreground color green off 1= foreground color green on
32	FgColorBlue	0 = foreground color blue off 1= foreground color blue on
64	BgColorRed	0 = background color red off 1= background color red on
128	BGColorGreen	0 = background color green off 1= background color green on
256	BgColorBlue	0 = background color blue off 1= background color blue on
512	SmoothGradient	0 = Color pattern displayed as bars 1 = Color pattern displayed as smooth gradient
1024	LineSpacing	An integer between 1 and 200

Table 61: Show Pattern Properties Bit Positions

BIT POSITIONS													
…	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	LS	SG	BB	BG	BR	FB	FG	FR	GC	BT	SS

Standard Pattern

+ScreenUtility ShowPattern “stdpat”

Displays the standard color pattern.

No options for this function

Screen ID

+ScreenUtility ShowPattern “screenid”

Displays the output channel for each display device.

No options for this function

Phase Pattern

+ScreenUtility ShowPattern “phasepat”

Displays the phase pattern – every other pixel on/off.

No options for this function

Bitmap

+ScreenUtility ShowPattern “bitmap”

Displays the specified. image

No options for this function

Color Pattern

+ScreenUtility ShowPattern “colorpat”

Displays the color pattern page.

+ScreenUtility SetPatternProp “SetPatternProp” { set [options] }

Color Pattern displays a selected color as solid or 10 bars as gradient from color to black, or a smooth gradient from color to black.

+ScreenUtility SetPatternProp “SetPatternProp” { Set [options] }
{ 0x008 1 0 1 0 1 0 0 0 0 0 0 0 }
Set = 1, single screen = on (0) --Bar type = 1 -- vertical (2), Foreground = green (16) -- Set = 19 = 0x000013

Grid Pattern

+ScreenUtility ShowPattern “gridpat”

Displays a grid pattern on the screen.

+ScreenUtility SetPatternProp “SetPatternProp” { Set [options] }

Grid Pattern displays a pattern of horizontal and vertical lines.

+ScreenUtility ShowSetPatternProp “SetPatternProp”
{ 0x067F 1 0 1 1 1 1 0 0 1 1 20 } Single Screen on (1) – solid color off (0) – circle on (4) - Foreground = red + blue + green = white (8+16+32) --

background = blue (0+0+256) – gradient = bars (0) –
line spacing on = 20
Set = 1+0+4+8+16+32+64+0+0+0+512+1024=1661 = 0x067E

Exit Screen Utility

+ScreenUtility ShowPattern “”

This ‘null’ Function turns the Screen Utility off.

2.8.16 SysMonEx

Table 62: SysMonEx Object

Methods	
QueryAllValues	([out] TCPSysMonValue_array_t * values)
QueryValues	([in] CPSysMonValueCode_array_t list, [out] TCPSysMonValue_array_t * values)
QueryECCInfo	([out] TCPSysMonECCInfo * ecc)
RegisterNotifyTarget	( )
UnregisterNotifyTarget	( )

The **SysMonEx** object and its methods are used to retrieve the parameters used in system monitoring.

Usage

+SysMonEx QueryAllValues

Returns all system monitoring information. This method uses the **TCPSysMonValue_array_t** data structure.

Example:

To illustrate the use of the Module and Parameter Codes as shown in the tables, consider an installation with two chassis (a main chassis and an expansion chassis) and each chassis has a set of fans. To show values for each fan, the code for Fan 1 in the Main Chassis is 8192 + 0x01000000 = 0x01002000 [16785408] and the code for Fan 1 in Expansion Chassis 1 is 2048 + 0x02000000 = 0x02000800 [33562624]. All returned values are in decimal. This example shows the status of all three chassis fans at 0 RPM. (Status values other than zero indicate an alarm condition.) The example has been formatted for ease of reading.

```
=00000000
{ 16 { 1 30 30 0 }      { 2 29 29 0 }          { 3 5921 5921 0 }
{ 4 6026 6026 0 }       { 5 1456 1456 0 }      { 6 1456 1456 0 }
{ 16 3248 3248 0 }      { 160 3312 3312 0 }    { 32 4731 4731 0 }
{ 48 11917 11917 0 }    { 64 -11410 -11410 0 } { 176 1792 1792 0 }
{ 256 32 32 0 }         { 16779264 0 1000 2 }  { 16781312 0 1000 2 }
{ 16783360 0 1000 2 } }
```

The first number (16) is the number of parameters returned in the array

=00000000 { 16 { 1 30 30 0 }...

1 = CPU1 temperature (code)
30 = 30 degrees C (value)
30 = 30 degrees C (threshold) – unless there is a fault, the threshold will always equal the value
0 = normal (status)

Usage

+SysMonEx QueryValues { n par1 par2 ... }

Returns specified system monitoring information. This method uses the **TCPSysMonValue_array_t** data structure.

+SysMonEx QueryValues { 2 1 48 }

Returns 2 parameters (CPU1 Temp and +12V)

=00000000 { 2 30 11917 }

Usage

+SysMonEx QueryECCInfo

Returns ECC error information. This method uses the **TCPSystemECCInfo** data structure.

+SysMonEx QueryECCInfo

=00000000 { 1 0 0 0 }

TCPSysMonECCInfo Structure

Valid – 1 if the data in the structure is valid, 0 if the data is invalid.
SingleBitErrors – total number of single-bit errors since last boot.

MultiBitErrors – total number of multi-bit errors since last boot.
BlockAddr – address of the 4k block in which the last error occurred.

Usage

+SysMonEx RegisterNotifyTarget

Registers the client to receive notifications when the system status changes. Refer to Object, **"SysMonNotifyEx" on page 99**. Use this method to turn on the notify messages.

+SysMonEx UnregisterNotifyTarget

Unregisters the client to receive system-monitoring notifications. Use this method to turn off the notify messages.

TCPSysMonValue Structure

Code – the code of the monitored parameter
Value – the current reading of the parameter
Threshold – threshold value that caused the alarm
Status – status of the parameter

Status

The **status** field shows the **current status** of the monitored value:
0 – normal
1 – above high threshold. (See **threshold** field for threshold value)
2 – below low threshold. (See **threshold** field for threshold value)
3 – invalid measurement

As an example, SysMon can provide system monitoring for the specific values found in the Fusion Catalyst 4000:

Table 63: TCPSysMonValue Units

Parameter	Units
CPU Temp.	Degrees C
CPU2 Temp.	Degrees C
CPU1 Fan Speed	RPM
CPU2 Fan Speed	RPM
CPU1 Core Voltage	millivolts
CPU2 Core Voltage	millivolts
+3.3 Volts	millivolts
+3.3 Volts Standby	millivolts

Table 63: TCPSysMonValue Units

+5.0 Volts	millivolts
+12 Volts	millivolts
-12 Volts	millivolts
+1.8 Volts	millivolts
Motherboard Temp	Degrees C
Chassis Fan 1	RPM
Chassis Fan 2	RPM
Chassis Fan 3	RPM

2.8.17 SysMonNotifyEx

Table 64: SysMonNotifyEx Object

Methods	
ValuesChanged	([in] TCPSysMonValue_array_t)
ECCError	([in] TCPSysMonECCInfo)

ControlPoint clients that wish to receive system-monitoring notifications from the server need to implement the **SysMonNotifyEx** object.

Note It is important to note that this Object and its Methods are normally ONLY called by the server. Once you have registered notification this object will be called whenever a change is made in SysMon.

Usage

+SysMonNotifyEx ValuesChanged

The server calls this method on the client to supply information about values that changed. The supplied parameter is an array (**TCPSysMonValue_array_t**) of the values that have changed since the last notification.

:SysMonNotifyEx ValuesChanged { 3 { 5 1696 1696 0 } { 128 -5178 -5178 0 } { 48 12464 12464 0 } }

+SysMonNotifyEx ECCError
Called when an ECC error occurs. This method returns the System EDD Data.
:SysMonNotifyEx ECCError { 1 0 0 0 }

TCPSysMonECCInfo structure

Valid – 1 if the data in the structure is valid, 0 if the data is invalid.
SingleBitErrors – total number of single-bit errors since last boot.
MultiBitErrors – total number of multi-bit errors since last boot.
BlockAddr – address of the 4k block in which the last error occurred.

+EventLog GetRecord
Reads one record from the event log. Returns the record header information, the name of the source that generated the event and the text of the event. The position is automatically moved to the next record depending on the seek mode (forwards or backwards) specified in flag Bit 3 or Bit 4.

{ header } “source” “text”
Where header is described below, source is the program element that caused the error and text is the text of the event log message

=00000000 { 1074003997 0 20 1047388718 } "GalSysMon" "Chassis Fan-3 speed back to normal. Current speed is 2463 RPM"

=00000001 { 0 0 0 0 } “” “”
This is an error condition – indicating end of file (EOF).

header – { nEventID CPEventLogRecType_t nRecNumber nTime }
nEventID = the identifier of the event
The **nEventID** field is the unique identifier of the event.

CPEventLogRecType_t nType = type of the event (severity)
The **nType** field shows the type (severity) of the event:
0 – Information message
1 – Error event

2 – Warning event

nRecNumber = the record number in the event log
The **nRecNumber** field shows the position of the record in the system event log.

nTime = encoded time of the event
The **nTime** field has the encoded time of the event. It is the number of seconds elapsed since midnight (00:00:00) January 1, 1970.

+EventLog RegisterNotifyTarget
Registers the client to receive event notifications from the server. Refer to Object, **"EventLogNotify" on page 75**.

+EventLog UnregisterNotifyTarget
Unregisters the client to receive notifications.

2.8.18 UserMan

Table 65: User Management Object

Methods	
AddUser	([in string] UserName, [in string] AuthToken, [in] short level)
DeleteUser	([in] string)
EnumUsersCS	([out] string)
SetUserInfo	([in string] UserName, [in string] AuthToken, [in] short level)
GetUserInfo	([in string] UserName, [out string] AuthToken, [out] short level)
ChangePassword	([in string], OldToken, [in string] NewToken)

The **User Management** object and its methods are used to set and get user names and passwords as well as change passwords and user level.

Usage

+UserMan AddUser “username” “password” level
adds a user of user name and password with user level see **User Levels** below

+UserMan DeleteUser “username” deletes user

+UserMan EnumUsersCS lists all users
=00000000 "admin,localuser,test"

+UserMan SetUserInfo “username” “password” level
sets password and level for username

+USerMan GetUserInfo “username”
=00000000 "test" 1
returns password and level for username

+UserMan ChangePassword “oldpassword” “newpassword”
must be logged in as user to change password – only the user can change his own password.

User Levels
Level 0, administrator has full access
Level 1, user cannot administer users and cannot shut down the server
Level -1, special case (-1 means that the account is disabled)

2.8.19 VidStreamSys

Table 66: VidStreamSys Methods

Methods	
NewWindow	([out] WinId_t pwid)
NewWindowWithId	([in] WinId_t wid)
SetSource	([in] WinId_t wid, [in,string] wchar_t * srcName)
GetSource	([in] long wid, [out,string] wchar_t ** srcName)
GetDecoder	([in] WinId_t wid, [out] short * decoder)
GetNumDecoders	([out] short * decoders)
Start	([in] WinId_t wid)
Stop	([in] WinId_t wid)
SetCrop	([in] WinId_t wid, [in] struct CPRect * pRect)
SetOrigin	([in] WinId_t wid, [in] long x, [in] long y)
GetCrop	([in] WinId_t wid, [out] struct CPRect * pRect)
GetInputSize	([in] WinId_t wid, [out] struct CPSize * pSize)
GetSVSVersion	([in] WinId_t wid, [out] short * svsver)

The VidStreamSys object and its methods are used to set up VideoStream windows from SVS-8 units.

Usage

NewWindow creates a new VideoStream window with a server assigned ID.

VidStreamSys NewWindow
=00010123
Returns server assigned ID
NewWindowWithId creates a new VideoStream window with a user-specified ID

VidStreamSys NewWindowWithId
=00000123
Returns user assigned ID

SetSource sets the source for a VideoStream window. The window ID must be specified. The source object must be predefined. Refer to the ControlPoint User Manual about creating a VideoStream source object.

+VidStreamSys SetSource { 123 } “SourceObjectName”
=00000000
Sets the streaming source object for the specified window.

SetSource returns the object name of the current video source on a VideoStream window. The window ID must be specified.

+VidStreamSys GetSource { 123 }

GetSource Sets the object name of the current source on a VideoStream window. The window ID must be specified.

+VidStreamSys GetSource { 123 }
=00000000 “SourceObjectName”
Returns the object name of the current source

GetDecoder returns streaming video decoder channel number. The window ID must be specified.

VidStreamSys GetDecoder
=00000000
Returns decoder channel number for the specified window

GetNumDecoders returns the number of streaming video decoders attached to the system.

VidStreamSys GetNumDecoders
=00000000 8
Returns number of decoders available.

Start starts displaying the image from a video source in a VideoStream window.
The window ID must be specified.
+VidStreamSys Start { 123 }

Stop stops displaying the video image in a VideoStream window. The window ID must be specified.
+VidStreamSys Stop { 123 }

SetCrop sets the cropping parameters to an VideoStream window. Cropping works by relation to the normal image size. X and Y specify the pixels removed from the left and the top. W and H specify the size of the cropped image. A 640x480 window set at 10 10 600 400 will have 10 pixels remove from the top, 10 pixels removed from the left, 30 pixels from the right, and 70 removed from the right. ImageWidth – X – (right pixels) = W or 640 – 10 – 30 = 600.
+VidStreamSys SetCropPrm { 123 } { X Y W H }
+VidStreamSys SetCropPrm { 123 } { 10 10 600 400 }

GetCrop returns the cropping parameters from an VideoStream window.
+VidStreamSys GetCropPrm { 123 }
=00000000 { 27 0 100 0 0 100)
Returns nX nY nW nH

SetOrigin sets the origin of a cropped image to effect panning.
+VidStreamSys SetOrigin { 123 } X Y

GetInputSize returns the size of a specified Streaming Video window.
VidStreamSys GetInputSize { 123 }
=00000000 { 123 } 640 480
Returns cx cy

GetSVSVersion returns the version number of the SVS server.
+VidStreamSys GetSVSVersion { 123 }
=00000000 "1.9.4.348"
Returns current installed version of the SVS Server software.

2.8.20 IPStreamSys

Table 67: IPStreamSys Methods

Methods	
NewWindow	([out] WinId_t pwid)
NewWindowWithId	([in] WinId_t wid)
SetSource	([in] WinId_t wid, [in,string] wchar_t * srcName)
GetSource	([in] long wid, [out,string] wchar_t ** srcName)
GetDecoder	([in] WinId_t wid, [out] short * decoder)
GetNumDecoders	([out] short * decoders)

The IPStreamSys object and its methods are used to set up IPStream windows from Quad HD boards.

Usage

NewWindow creates a new IPStream window with a server assigned ID.

+IPStreamSys NewWindow
=00010123
Returns server assigned ID

NewWindowWithId creates a new IPStream window with a user-specified ID
+IPStreamSys NewWindowWithId
=00000123
Returns user assigned ID

SetSource sets the video for an IPStream window. The window ID must be specified. The source object must be predefined. Refer to the “ControlPoint Software Manual” about creating a IPStream source object.

+IPStreamSys SetSource { 123 } “SourceObjectName”
=00000000
Sets the streaming video object for the specified window.

SetSource returns the object name of the current video source on a IPStream window. The window ID must be specified.

+IPStreamSys GetSource { 123 }

GetSource Sets the object name of the current source on an IPStream window. The window ID must be specified.

+IPStreamSys GetSource { 123 }
=00000000 “SourceObjectName”
Returns the object name of the current source

GetDecoder returns streaming video decoder channel number, Link Status, and IP Address. The window ID must be specified.

+IPStreamSys GetDecoder { 123 }
=00000000 { 3 { "Decoder Channel" "1" } { "Link Status" "1" } { "IP Address" "10.4.1.63" } }
Returns Decoder Channel Number, Link Status, and IP Address for the specified window.

Note Ignore the “80040503 (Invalid RMC method name)” Error Message that appears when using this command.

GetNumDecoders returns the number of streaming video decoders attached to the system.
+IPStreamSys GetNumDecoders
=00000000 8
Returns number of decoders available.

2.8.21 Window

Table 68: Window Object

Methods	
GetState	([in] WinId_t, [out] TWindowState)
SetState	([in, out] TWindowState)
GetTitle	([in] WinId_t, [out, string] title)
SetTitle	([in] WinId_t, [in, string] title)
GetFrameInfo	([in] WinId_t wid, [out] struct CPWndFrameInfo * fi)
SetFrameInfo	([in] WinId_t wid, [in] struct CPWndFrameInfo * fi)
GetTitleInfo	([in] WinId_t, [out] struct CPWndTitleInfo * ti)
SetTitleInfo	([in] WinId_t, [in] struct CPWndTitleInfo * ti)
GetTitleFontInfo	([in] WinId_t wid, [out] struct CPWndTitleFontInfo * tfi)
SetTitleFontInfo	([in] WinId_t wid, [in] struct CPWndTitleFontInfo * tfi)
GrabImage	([in] WinId_t, [out,string] wchar_t **)

The **Window** object with its methods is used to set and get the state, title, frame information, title information, and title font information of a window.

Usage

+Window GetState { 123 }
Returns (Id Kind nState Set x y w h ZAfter)
=00000000 { { 123 } 2 1 7183 100 100 320 200 { 1 } }
SetState indicates valid data fields. 7183 indicates that all are valid. Refer to **“Flags” on page 46**.

+Window SetState { { WinID_t } Kind nState Set x y w h { ZAfter } }
Note that ZAfter is a WinId and puts the command window behind ZAfter. There are 2 special cases of ZAfter: -1 = on top with focus, -2 = send to back.

+Window SetState { { 123 } 2 1 7183 100 100 320 200 { 1 } }

+Window GetTitle { 123 }

Returns window title.

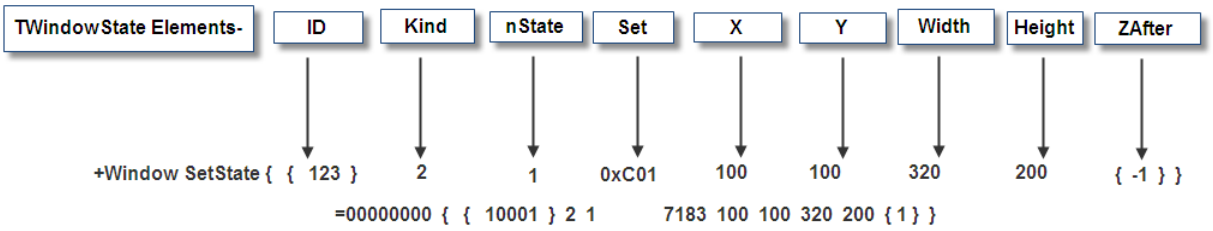
=00000000 { 123 } "RGB Window"

+Window SetTitle { 123 } “titlename”

Sets title “titlename” of the specified window.

=00000000

The **SetState** command is required to make a window visible. When first created with the **NewWindow** command, the window will be hidden. The window will have size (0,0) (height 0, width 0) and position (0,0).



When using the **SetState** command you must set the valid information fields in **Set** to make any subsequent fields valid. To make a window visible you must not only state that is visible in **nState**, you must make that field valid in the **Set** field.

nStateChange shows 0xC01. If we look in the tables, we see that the 0xC is a combination of 8 and 4 which validates the size and position field, while the 0x01 validates the **Visible** field in the **nState** field.

Table 69: Flags

nStateChange (Set)

Table 69: Flags

Get-Set Flags	* wsVisible +	0x0001
	* wsMinimized	0x0002
	* wsMaximized	0x0004
	* wsFramed	0x0008
	* wsLockAspect	0x0010
	* wsAlwaysOnTop	0x0020
	*wsCreated	0x0100
	*wsDestroyed	0x0200
	* wsPosition	0x0400
	* wsSize	0x0800
	* wsZOrder	0x1000
	wsTitle	0x2000
	wsKind	0x4000
	wsChannel	0x00010000
	wsBalance	0x00020000
	wsFormat	0x00040000
	wsCrop	0x00080000
	wsInput	0x00200000

* used in nStateChange ** Read Only

Table 70: nState Bits

nState bits	
wsVisible	0x0001
wsMinimized	0x0002
wsMaximized	0x0004
wsFramed	0x0008
wsLockAspect	0x0010
wsAlwaysOnTop	0x0020

Note Window SetState cannot be used to set the Kind type of a window.

FrameInfo

SetFrameInfo sets the frame information for a window. This information includes the width of the frame, the color of the frame and whether to show the title text or not.

When working the frame, you must first use the **Window SetState** command to turn the frame on or off.

To turn the frame on use the command below.
+Window SetState { { WinID } 2 8 0x0008 0 0 0 0 { 0 } }

To turn the frame off use the command below.

+Window SetState { { WinID } 2 0 0x0008 0 0 0 0 { 0 } }

Where WinID is the ID for the window you want to frame.

The Frame Info method under the Window object allows you to create, set color and size of a frame around a selected window. All ControlPoint windows can be framed. Zero or one must be use as a place holder for options not used. See Usage below.

+Object Method { WinId } { Set [options] }

+Window SetFrameInfo { WinId }
{ Set FrameWidth FrameColor ShowTitle ShowUserData }

Frame width (FW) in pixels – 0 to 100
Frame color (FC) = RGB
Red = 1 to 255
Blue = 1 to 255
Green = 1 to 255
White is all color on – (256) x (256) x (256) = 0xFFFFFF or 16,777,215
Black is all color off – 000 = 0x000000.
Example – green only – 00 FF 00 = 65,280 = 0x00FF00
Show title = 1 = display title – 0 = title off

+Window SetFrameInfo { 123 } { 7 10 00FF00 1 0 }
Green only - Set = 7 width, color and title all valid.

Table 71: FrameInfo Bit Positions

FrameInfo Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	0	SU	ST	FC	FW

+Window SetFrameInfo { 123 } { 7 10 16777219 1 } (Sets the window frame to width = 10, color = white, show title text = Yes)
+Window SetFrameInfo { 123 } { 2 0 255 0 0 } (Sets the window frame to color = red, ignoring other parameters)

GetFrameInfo returns the frame information from a window.
+Window GetFrameInfo { 123 }
=00000000 { 123 } { 7 10 16777219 1 0 }

Note Color sequence for the Window commands = Blue, Green, Red sequence rather than the standard RGB (Red, Green, Blue).

TitleInfo

The **SetTitleInfo** method under the Window object allows you to, set text color, text position, title position, minimum title bar height, title horizontal justification, and title vertical justification. Zero or one must be used as a place holder for options not used. See Usage below.

+Object Method { WinId } { Set [options] }
+Window SetTitleInfo { WinId }
{ Set TextColor BarPos MinBarHeight HorizJust VertJust }

Text color (TC) = R G B color – see **FrameInfo** (previous section)
Bar Position (BP)= 0 on top of frame – 1 on bottom of frame
Bar height (BH) = 0 to 100 pixels
Horiz. Just (HJ) = 0 = left – 1 = middle – 2 = right
Vert. Just (VJ) = 0 = top – 1 = middle – 2 = bottom

+Window SetTitleInfo { 123 } { 31 00FF00 0 20 1 1 }
Green only - Set = 31 TextColor BarPos MinBarHeight HorizJust VertJust all valid. Text color = green, on top, 20 pixels high, vertical and horizontal center.

Table 72: SetTitleInfo Bit Positions

SetTitleInfo Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	VJ	HJ	MH	BP	TC

+Window SetTitleInfo { wid } { Set TextColor BarPos MinBarHeight HorizJust VertJust }

+Window SetTitleInfo { 123 } { 1 0 0 0 0 0 } Sets the text color to black, and the other parameters are ignored

+Window SetTitleInfo { 123 } { 10 0 1 0 1 0 } Sets the title bar at the bottom of the window, the text is horizontal center justified, and the other parameters are ignored

GetTitleInfo returns the title information from a window.
+Window GetTitleInfo { 123 }
=00000000 { 123 } { 10 0 1 0 1 0 }

TitleFontInfo

The Title font Info method under the Window object allows you to set font size and font name Zero or one must be use as a place holder for options not used. See **Usage** below.

+Object Method { WinId } { Set [options] }

+Window SeTitleFontInfo { WinId }
{ Set TextSize TextFlags FontName }

Text size (TS) = 8 to 72 pixels
Text flag (TF) = 0 – not used
Font name (FN) = name of font wanted – must be a valid font name

+Window SetTitleFontInfo { 123 } { 3 12 0 Arial }
Set = 5 font name and font size valid.

Table 73: SetTitleFontInfo Bit Positions

SetTitleFontInfo Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	0	0	FN	TF	TS

SetTitleFontInfo sets the title font information for a window. This information includes the text size, text flags and the font name.

+Window SetTitleFontInfo { wid }
{ Set TextSize Flags FontName }

+Window SetTitleFontInfo { 123 } { 5 14 0 “Arial” } sets the text size = 14, font name = “Arial”, and ignores the flags

GetTitleFontInfo returns the title font information from a window.
+Window GetTitleFontInfo { 123 }
=00000000 { 123 } { 5 14 0 “Arial” }

GrabImage ([in] WinId_t, [out,string] wchar_t **)
+Window GrabImage { 123 }
=00000000 { 123 }
“C:\ProgramData\ControlPoint\ServerDataFiles\Images\XX_XX.bmp”
Returns a file path on the server for the image file. You can use the file path to query the image data from the server and save it to the local hard drive.

2.8.22 WinServer

Table 74: WinServer Object

Methods	
DeleteWindow	([in] WinId_t)
QueryAllWindows	([out] TWindowState_array_t)
QueryWindows	([in] WinId_t_array_t, [out] TwindowState_array_t)
FindWindow	([in,string] window_descriptor, [out] WinId_t)
InvokeAppWindow	([in,string] appWinName, [out] WinId_t)
GetAppWinInfo	([in] WinId_t winid, [out,string] window_descriptor, [out,string] cmdline, [out_string] workDir)
RegisterNotifyTarget	( )
UnregisterNotifyTarget	( )
GetServerInfo	([out] CPServerInfo)
GetScreenConfig	([out] CPScreenConfig)
Quit	( )
QueryAllLayoutsCS	([out, string])
QueryLastSetLayout	([out, string])
SetLayout	([in, string])
SaveLayout	([in, string], [in] WinId_t_array_t)
DeleteLayout	([in, string])

The **Window Server** object and its methods are used for server related items.

Usage

+WinServer DeleteWindow { WinId } (Deletes specified window)

+WinServer QueryAllWindows (Returns TwindowState_array_t)
This command returns the array of window states. The first digits are the windows in the array.
=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } } { { 102 } 3 1 7183 152 21 320 240 { 102 } }

+WinServer QueryWindows { 3 { 10001 } { 10002 } { 123 } }
Return TwindowState_array_t for each specified window ID.

+WinServer FindWindow “elara*”
=00000000 { 11033 }
Searches for a window on the screen that matches the specified window descriptor. Returns the window ID of the window. Returns a zero ID if no window is found.

+Winserver InvokeAppWindow “vnc_elara”
=00000000 { 11033 }
Invokes an application window object with the specified application object name. Returns the window ID of the application window.

+WinServer GetAppWinInfo { 123 }
=00000000 "ELARA|VNCVIEWER|VNCVIEWER"
"\"D:\\Program Files\\RealVNC\\vncviewer.exe\" /viewonly /config elara-5900.vnc "D:\\Program Files\\RealVNC\\"
Returns application information for the specified window. Returns the window descriptor, the command-line for the executable, and the working directory for the application.

+WinServer GetServerInfo
=00000000 { PlatformCode ModelVersion ModelRevision OEMCode SerialNumber } “Version”
=00000000 { 16 0 0 0 1057 } "1.9.4.348"
Returns{ dwVersionMS dwVersionLS dwFileTimeMS dwFileTimeLS }

+WinServer RegisterNotifyTarget turns on notify messages

+WinServer UnregisterNotifyTarget turns off notify messages

+WinServer GetServerInfo
Returns{ dwVersionMS dwVersionLS dwFileTimeMS dwFileTimeLS }

+WinServer GetScreenConfig
Returns { TotalWidth TotalHeight SingleScreenWidth SingleScreenHeight }

+WinServer Quit quits (exits) ControlPoint Server

+WinServer QueryAllLayoutsCS Returns all layout names
=00000000 “Layout1” “Layout2”...

+WinServer QueryLastSetLayout Returns the last loaded layout
=00000000 “SJ Layout”

+WinServer SetLayout “layoutname” Applies named layout

+WinServer SaveLayout “layoutname” { 0 } Saves named layout
{ WinId_t_array_t } (nCount WinId_t pData[ ])

+WinServer DeleteLayout “layoutname” Deletes specified layout

Chapter 3—Programming

3. Programming

This chapter offers many suggestions, examples, and explanations, as well as programming examples and program considerations. This section includes most of the commands listed in the previous chapter. It is not meant to be all-inclusive; it does not give examples of every command in the protocol. This section provides a general working example of using the ControlPoint Protocol and its instruction set (usage for each command is given below that command in the previous chapter).

3.1 Understanding Command Structures

This section discusses the use of some of the more complex command structures with several examples.

3.1.1 Understanding Set Structure

Several commands use the structure:

Object Method { Set Elem1 Elem2 ... }

Set validates subsequent elements in the command string. This same command **structure** is used in a more complex manner in the **SetState** Method shown following. The **SetImgBalance** Method will be used as an example for explaining this structure.

The **SetImgBalance** method sets the **Image Balance** of the specified window type. The **SetImgBalance** method syntax and an example command are shown below.

+LiveVideoSys SetImgBalance { WinId } { Set Bri Con Gam Hue Sat }
+LiveVideoSys SetImgBalance { 101 } { 27 0 100 0 0 100 }

The Set Element is used to validate all other elements in the subsequent list.

Table 75: Set Bit Positions

Set Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	Sat	Hue	Gam	Con	Bri

In this example **Set** is equal to 27. Here, 27 is the sum of 16, 8, 2, and 1 (1+2+8+16=21). These values equate to the bit positions (powers of 2) that are associated with the image balancing properties: **Hue**, **Saturation**, **Contrast,** and **Brightness**.

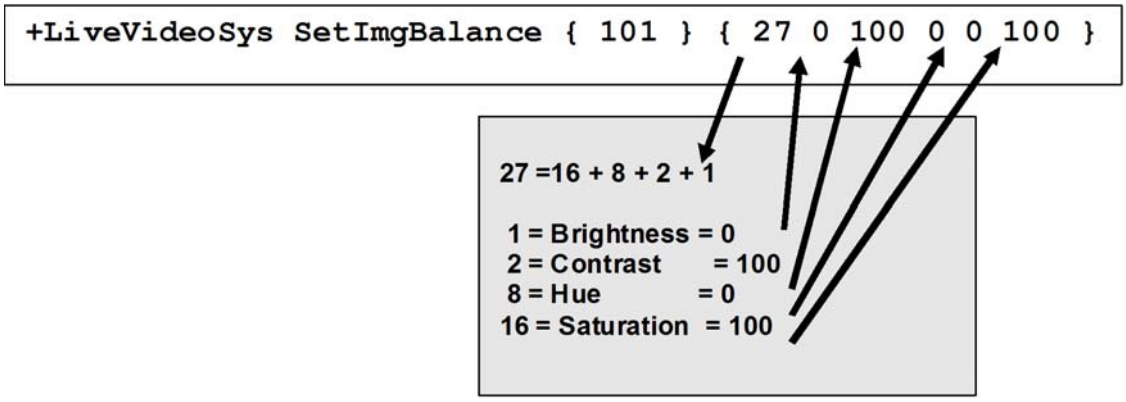


Figure 13 - SetImgBalance

Since **Gamma** is not used, **Set** with a value of 27, validates all four remaining elements in the list. The values shown are the default values for the parameters.

If you wanted only to set contrast and brightness (2 and 1) the **Set** element should be 3 as shown in the following program line which sets contrast to 7 and brightness to 93. All other values in the element list will be ignored except those specified by **Set**.

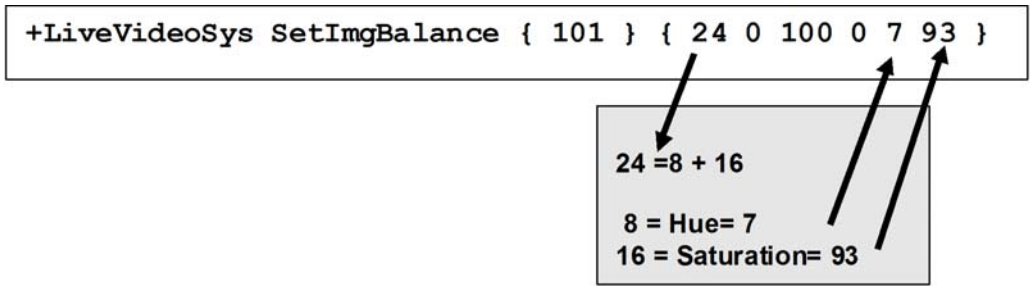


Figure 14 - Hue and Saturation

Hue will be set to a value of 7 and Saturation to 93.

3.1.2 Understanding and Using Flags

The following table is a list of the available Flags. These Flags are used with the data elements **nState** and **nStateChange (Set),** which are part of the **TWindowState** data structure. The flags in the table are **OR**ed together to form the **nState** or **nStateChange (Set)** element, and are used with the **GetState, ImgBalance, CPLiveVideosource**, **SetState,** and other commands.

Table 76: Flags

Flags		
Get - Set Flags	wsVisible	0x0001
	wsMinimized	0x0002
	wsMaximized	0x0004
	wsFramed	0x0008
	wsLockAspect	0x0010
	wsAlwaysOnTop	0x0020
	wsPosition	0x0400
	wsSize	0x0800
	wsZOrder	0x1000
	wsTitle	0x2000
Notify Flags (read only)	wsKind	0x4000
	wsChannel	0x00010000
	wsBalance	0x00020000
	wsFormat	0x00040000
	wsCrop	0x00080000

Several of these flags may be combined in one argument, some examples:

0x0005 = wsVisible, and wsMinimized
0x000b = wsVisible, wsMinimized, and wsFramed
0x0c03 = wsPosition, wsSize, and wsVisible, and wsMinimized

As another example, for **nStateChange (Set)** with returned value of 7181 = 0x1C0D, the bit positions for the following flags are set: **WsVisible**, **wsMinimized**, **wsFramed**, **wsPosition**, **wsSize**, and **wsZOrder**.

Note that if both **wsMinimized** and **wsMaximized** are clear (0) a window is restored.

3.1.3 Validity Fields

The following table shows the flags listed in the table on the previous page in their bit position order by hex digit.

Table 77: Flag Hex Positional Values

0x10000=16		0x1000=8		0x100=4		0x10=2		0x1=1	
80000	wsCrop	**8000**	na	**800**	wsSize	**80**	na	**8**	wsFramed
40000	wsFormat	**4000**	wsKind	**400**	wsPosition	**40**	na	**4**	wsMaximized
20000	wsBalance	**2000**	wsTitle	**200**	wsDestroyed	**20**	wsAlwaysOnTop	**2**	wsMinimized
10000	wsChannel	**1000**	wsZOrder	**100**	wsCreated	**10**	wsLockAspect	**1**	wsVisible

The maximum values for setting valid change fields for each column are 3, E, 1, D. The hex value 0x3E1D equates to 15901 decimal. Note that some variables are mutually exclusive (i.e., minimum/maximum and created/ destroyed), so maximum values cannot be *F* for a given column. Read Only variables are not included here, as they cannot be changed.

Table 78: nState Bits

nState bits	
wsVisible	0x0001
wsMinimized	0x0002
wsMaximized	0x0004
wsFramed	0x0008
wsLockAspect	0x0010
wsAlwaysOnTop	0x0020

nState bits tell ControlPoint how to change the state of a window, while ***nStateChange* (Set)** bits tell ControlPoint what fields are valid and how you want to change the window.

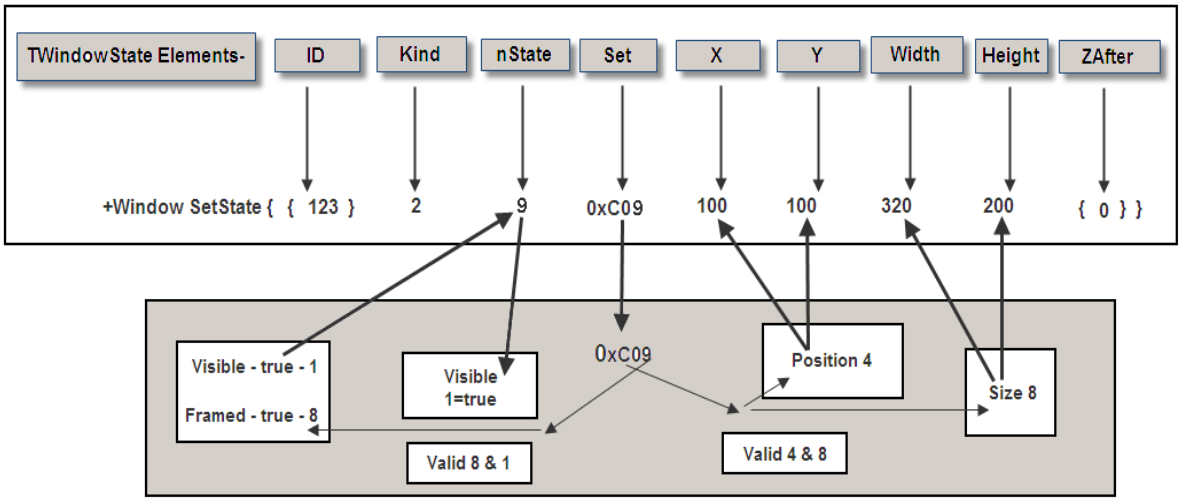


Figure 15 - nState

3.2 Window Titles

By default, the server gives a blank title to every window created. The user can change this title. The title is displayed only when the window's frame and title are made visible. The title is also displayed in the ControlPoint Client window icons. Users should give meaningful names to all windows.

3.3 Window IDs

Two methods can be used to create windows.

1. You can let the ControlPoint Server assign an ID number automatically. Use the **NewWindow** Method.

2. You can assign the Window ID yourself. Use the ***NewWindowWithId*** Method.

When the window is created the first time, a **WinId** and a title (blank) are given to that window by the server. When you save the window in a layout, the title is saved in that layout. **WinId**s are not saved when they are given by the server and will be unique for each window during the server session.
When you set a layout, the window is re-created and the server gives it the next available **WinId** for the session. Because the server creates a new

window, it has to give it a new **WinId**. The title remains the same, as it was saved in the layout.

3.4 Server Assigned Window IDs

Windows that are created with the **NewWindow** Method are assigned a Window ID that is greater than 10,000 (e.g. **+LiveVideoSys NewWindow**). This number is subject to change on different executions of the server. This can happen on a reboot or if you stop and then restart the ControlPoint Server.

Server assigned IDs were designed for creating windows manually with the ControlPoint Clients. If you are programming a system for use with a serial device (i.e. touch panel) or external control program you will need to keep track of window ID's so you can reference specific individual windows and control them as needed. Working with programmed windows will require that you employ **User-assigned IDs** to create your windows.

Note	Window IDs created by the server are not guaranteed to have the same ID number. To guarantee the validity of window ID numbers, you must employ User-Assigned window IDs in your programming.

3.5 Objects

The following are the available ControlPoint objects:

1. Applications
2. Named Inputs
3. CPShare
4. IPStream
5. VideoStreams
6. Image Cache *
7. PixelNet Input

* Image Cache is not a fully qualified object as it is not drag and drop at this time

Only Named input and the Application Object will be discussed here as the others require only minimal creation as an Object.

3.5.1 Named Inputs

3.5.1.1 Using Named Inputs

Named Inputs can be used to control a specific window. The term Named Input comes from the function of controlling the properties, or parameters, of a specific (named) input source such as **Server27** or **JoesOffice**.

Named Inputs can be used only with LiveVideo, DVI Capture, and DVI-KM windows. Named Inputs can be used to set parameters from any of the window property pages seen when a window is selected. The property pages may be one or more of the following: Source, Image Quality, Cropping, and RGB Timing.

A Named Input

1. Requires previous setup (refer to the following procedure).
2. Is a template for the window properties that it creates

   a – Window's properties defines that window
   b – Properties from Named Input are placed into the window properties
   c – The window is now defined by the properties set in Named Input
3. Allows property control of specific sources (Named Input)

Use a Named Input:

1. To set specific properties for a specific (Named) Video or RGB input source – usually due to needed adjustments.

Note	Size and Position are not included in the **Named Input** properties; these are either part of the Layout or already created when using the **Named Input** manually in an existing window.

2. To make specific adjustments for a group of sources (generic Named Input using channel zero)
3. To use window properties that you don't want to use a layout for
4. To set window properties from a touch panel (may require two outputs – one to the Fusion System and one to the matrix switch).
5. To open RGB windows very quickly without AutoDetect analyzing the signal which can take several seconds – Named Input is 'immediate'.
6. To use serial output to control an external device (i.e. matrix switch) without need for touch panel or execute another such controlling application.

Note	Not meant for general application execution—use menu item or **Application Object**.

A window is defined by its properties. Any change in the properties like channel or brightness, changes the appearance of the window contents. The Named Input acts as a template for the window's properties. The contents of the Named Input are used to fill the content of the window's Properties, thus defining the window.

In order to use a Named Input with the protocol, you must first create a Named Input object. Just like using a layout, you must first create that layout before you can call it through a protocol command.

If Channel 0 is selected in the Named Input Channel setting, the channel set in the window's properties will be used instead of the usual channel setting selected in the Named Input. This allows for specific source settings that are not dependent upon a specific RGB or LiveVideo input channel. For

example: you can put a given RGB source into different RGB windows set at different channels, or you can set the same adjustments for several different sources with similar properties.

Named Input Procedure (RGB input example)

1. Open RGB window and fill with desired source
2. Open window **Properties**
3. Make adjustments as necessary for this window's source:
   a. RGB Timing for a specifically difficult source
   b. Special cropping for a series of same systems
   c. The object to be called from external control
   d. Control of an external switch
   e. Any window properties can be used
4. Open ControlPoint Client (either locally or remotely)
5. Open the **Object Browser**
6. Select **Named Inputs**
7. Select the created/adjusted window mimic icon in the CPClient GUI
8. Right click in the **Object Browser Input** window
9. Select **New** – you will see properties similar to your Window Properties
10. Name the window appropriate to the source
11. Take care to not get these properties confused with window properties
12. It will automatically save when you close the INPUT property windows.

3.5.2 ControlPoint Protocol For Named Inputs

There are three new functions for the **GalWinSys** Object:

+GalWinSys QueryAllInputsCS

Returns a string with comma-delimited input names.

+GalWinSys SelectInput { 123 } "InputName"

Selects an input into window number 123.

+GalWinSys GetInput { 123 }

Returns the currently selected input for window number 123.

There is also a new WindowsState flag associated:

wsInput = 0x00200000

This flag is used in window state notifications to show that a new input is selected for the window.

3.5.3 Applications

Application Objects:

1. Require prior setup and running of the application.
2. Allow you to open and close applications with layouts.
3. Allow you to drag and drop objects on the desktop to open applications from the Object Browser.
4. Can be invoked manually from the Object Browser
5. Allow you to keep applications alive between layout changes.
6. CP 'owns' applications registered as objects.
   Will close with *Close All Windows* and with layout change.
   Must be run as object, not 'ad-hoc' to be owned.
7. Applications run by Scheduler must be registered as objects.
8. Can be used as a short cut to run a program – easier than menu item

3.5.3.1 Timeout and the Descriptor

1. Descriptor must match an open window within 'Timeout' seconds
2. If no Descriptor is matched, application may open but not be 'owned'. Will open in most cases.
3. Error displayed for unmatched Descriptor
4. Maximum Timeout setting = 30 seconds
5. Adjust Timeout for slow loading aps (Acrobat reader?)
6. Caution – Layout loading stops until 'Timeout' times out.
   Windows loaded after application will wait until Timeout finishes – can slow down Layout loading.

Before an Application Object can be controlled with the protocol it must first be created on the wall controller.

3.5.3.2 Application Object Procedure

1. Run the application on Wall Controller.
2. Run CPClient (locally or remote).
3. Open **Object Browser** and select **Application** tab.
4. Select the application's icon in the **Wall Mimic** - NOT the actual application window.
5. Right click in **Applications** and select **New**
6. See object properties open - derived from Application info.
7. Closing with **OK** saves object.

3.5.3.3 ControlPoint Protocol for the Application Objects

+Winserver InvokeAppWindow "vnc_elara"
=00000000 { 11033 }
Invokes an application window object with the specified application object name. Returns the window ID of the application window.

+WinServer GetAppWinInfo { 123 }
=00000000 "ELARA|VNCVIEWER|VNCVIEWER"
"\"D:\\Program Files\\RealVNC\\vncviewer.exe\" /viewonly /config elara-5900.vnc "D:\\Program Files\\RealVNC\\"
Returns application information for the specified window. Returns the window descriptor, the command-line for the executable, and the working directory for the application.

3.6 User-Assigned Window IDs

Windows that are created with the **NewWindowWithId** method will have ID numbers between 0 and 10,000 (exclusive). This ID number will always be the same and will be saved with the layout, when saved.

If you are programming for a serial device or external control program, it is important that you employ User-Assigned ID's when creating your windows (e.g. **+LiveVideoSys NewWindowWithId { 123 }**) so that you will always have the same window ID associated with a given window. A

constant and unique window ID is required when addressing windows for control by an external program or a serial device such as a touch panel.

Note In order to preserve window ID's when programming and using Layouts, it is important that you use the **NewWindowWithId** command when creating your windows. The User-Assigned Window ID will be saved as part of your layout.
Example: **+LiveVideoSys NewWindowWithId { 123 }**

3.7 Program Examples

This section shows you several coding examples for using the ControlPoint Protocol. All examples assume you are already logged into the server. For clarity, response lines are shown only for commands requesting information (**Get** commands).

3.7.1 Creating Windows

There are four basic operations necessary to open a window with LiveVideo or RGB content on the Display Wall.

Step One – New Window

First, you must create the window; this is done with the **NewWindow** or **NewWindowWithId** command. For our purposes here, we will use the **NewWindowWithId** command so we know what window is being used. Windows may be created with the **RGBSys** and **LiveVideoSys** Objects, or may be created generically with the **GalWinSys** Objects.

+LiveVideoSys NewWindowWithId { 101 }

Step Two – Set Type

Second, you must define the content for this window by selecting a Subsystem and channel. This is done with the **LiveVideoSys** or **RGBSys** Objects and the **SetChannel** Command after creating a window as above.

+LiveVideoSys SetChannel { 101 } 2

Step Three - Start

Third, you must start the capture process with the **Start** command. In **all** cases, the window type must be specified **before** it is Started. A window's type can be specified with either the proper object (e.g. **RGBSys**) when

creating the window, or with the **SetChannel** method. It is the action of giving the window a specified content that makes it a specific Subsystem type when using the SetChannel method.

```
+GalWinSys Start { 101 }
```

Step Four – Set State

Fourth, you must make the window visible, size it, and place it somewhere on your display wall. The four parameters required to do this can be set with the **SetState** command. All windows are created with 0 width, 0 height, at position (0,0), and are hidden (invisible), so at a minimum these four parameters **must** be set with this command in order to view the window.

```
+Window SetState { { 101 } 2 1 7183 100 100 320 240 { -1 } }
```

The listing below shows the creation of an RGB window using the generic form with the **GalWinSys** Object.

1. **GalWinSys NewWindowWithId**
2. **RGBSys SetChannel**
3. **GalWinSys Start**
4. **Window SetState**

Alternately, windows can be created by using the specific subtype **Object**.

```
1. RGBSys NewWindowWithId
2. RGBSys SetChannel
3. RGBSys Start
4. Window SetState

+LiveVideoSys NewWindowWithId { 101 }
+LiveVideoSys SetChannel { 101 } 2
+GalWinSys Start { 101 }
+Window SetState { { 101 } 2 1 7183 100 100 320 240 { -1 } }
```

Creating the window with the **RGBSys** or **LiveVideoSys** Objects sets the window type and allows you to use the **Start** command immediately after if desired.

Care must be taken when creating windows, to set the subtype **before** you **Start** the window. When generic windows are created and open using **GalWinSys**, a **SetChannel** command must be sent before the **Start** command. The **SetChannel** command sets the subtype so the **Start** command knows what to start. If a subtype is used to create the window (i.e. **RGBSys** or **LiveVideoSys**), then you may send the **Start** command immediately and subsequently set a channel and then size and place the window in whichever order you wish. To avoid errors, keep track of all window channels. RGB Capture Channels must be unique.

Note RGB windows must have unique channel numbers. If you have your inputs connected directly you were only allowed one window per input. With the Fusion Catalyst series you get 4 windows per board (i.e. 2 windows per channel).

Some window considerations:

- LiveVideo windows cannot overlap or be stacked with each other but can overlap or be staked with any other window.
- RGB windows can overlap and be stacked with any window.

Note The window **SubSystemKind** must be established before issuing the **Start** command. This procedure must be followedinordertoavoid**WindowTypeMismatch**errors.

Windows are created with 0 height, 0 width, hidden, and at position (0,0). These four parameters must be set (using the **SetState** command), in order to open and view an RGB or LiveVideo window.

Video windows default to channel 1 when created. When creating a window, it is important to use the **SetChannel** command to select the proper input to be viewed.

3.7.2 Simple Open and Configure Windows

This example opens and places two windows – one RGB and one LiveVideo window – on the Display Wall. Comments are placed below code lines. This example also configures the two windows. LiveVideo will be configured for PAL and S-Video input.

+LiveVideoSys NewWindowWithId { 101 }
creates a new LiveVideo window with Id # 101

+RGBSys NewWindowWithId { 102 }
creates a new RGB window with Id # 102

+LiveVideoSys SetChannel { 101 } 3
set LiveVideo channel for window 101 to channel 3

+LiveVideoSys SetVideoSource { 101 } { 3 2 1 }
set Format and Type – PAL, S-Video

+RGBSys SetChannel { 102 } 2
set RGB channel 1 for window 102 to channel 2

+Window SetState { { 101 } 2 1 7183 100 100 320 240 { -1 } }
make Video visible, place at (100, 100), size at 320x240 and place on top

+Window SetState { { 102 } 3 9 7183 0 0 320 240 { -1 } }
make RGB visible, framed, place at (0, 0), size at 320x240 and place on top

Windows must be made visible, sized and positioned. Windows when first created have size (0,0), position (0,0), and are invisible (hidden).

The following four lines of code are the minimal needed to create and place a video window.

+LiveVideoSys NewWindowWithId { 101 }
+LiveVideoSys SetChannel { 101 } 2
+GalWinSys Start { 101 }
+Window SetState { { 101 } 2 1 7183 100 100 320 240 { -1 } }

Note	Care must be taken to set the subtype either with the specific Object and Method or by using the **SetChannel** command, before starting the window.

3.7.3 Create and Configure Windows

Alternately, you can create windows with GalWinSys and configure them for type later by setting the input channel as shown below. This example omits the **SetState** command to show only the window creations.

+GalWinSys NewWindowWithId { 101 }
opens a new window with Id # 101

+GalWinSys NewWindowWithId { 102 }
opens a new window with Id # 102

+LiveVideoSys SetChannel { 101 } 2
set **LiveVideo** channel for window 101 to channel 2

+RGBSys SetChannel { 102 } 2
set **RGB** channel 1 for window 102 to channel 2

+GalWinSys Start { 101 }
start the capture process for window 101

+GalWinSys Start { 102 }
start the capture process for window 102

3.7.4 Change Window Type

You can change the type of a window from RGB to LiveVideo or LiveVideo to RGB by simply defining the channel using the Sub-System object in the command. This may be useful, if you wish to open a single window through which you wish to view many different inputs.

Change RGB Window to LiveVideo Window

+LiveVideoSys SetChannel { 101 } 1
set **LiveVideo** channel for window 101 to channel 1 – makes whatever window 101 was a LiveVideo window on channel 1 –
+GalWinSys Start { 101 }
After changing a window's sub-type, you must use the ***Start*** command.

Change LiveVideo Window back to RGB Window

+RGBSys SetChannel { 101 } 1
set **RGB** channel for window 101 to channel 1
+GalWinSys Start { 101 }

When changing the window subtype, you must use the **Start** command after the change.

3.7.5 Create, Configure, then Move Windows

This example opens and places two windows – one RGB and one LiveVideo window. Comments are placed below code lines. This example will configure the two windows and then move them. The first section of code is the same as listed in a previous example.

+LiveVideoSys NewWindowWithId { 101 }
opens a new LiveVideo window with Id # 101

+RGBSys NewWindowWithId { 102 }
opens a new RGB window with Id # 102

+LiveVideoSys SetChannel { 101 } 1
set LiveVideo channel for window 101 to channel 1

+LiveVideoSys SetVideoSource { 101 } { 3 2 1 }
set = Format and Type – PAL, S-Video

+RGBSys SetChannel { 102 } 1
set RGB channel 1 for window 102 to channel 1

+Window SetState { { 101 } 2 1 7183 100 100 320 240 { -1 } }
make Video visible, place at (100, 100), size at 320x240 and place on top

+Window SetState { { 102 } 3 9 7183 10 10 320 240 { -1 } }
make RGB visible, framed, place at (10, 1 0), size at 320x240 and place on top

+GalWinSys Start { 101 }
start the capture process for window 101

+GalWinSys Start { 102 }
start the capture process for window 102

We have now opened, configured, and placed an RGB and a LiveVideo window on screen one. Referring to the following figure, we will now move the RGB window to screen 3 (offset by 50 pixels both x and y), and the LiveVideo window to screen 6. We will move the video window and then resize it to fill the screen, and we will put a frame around it.

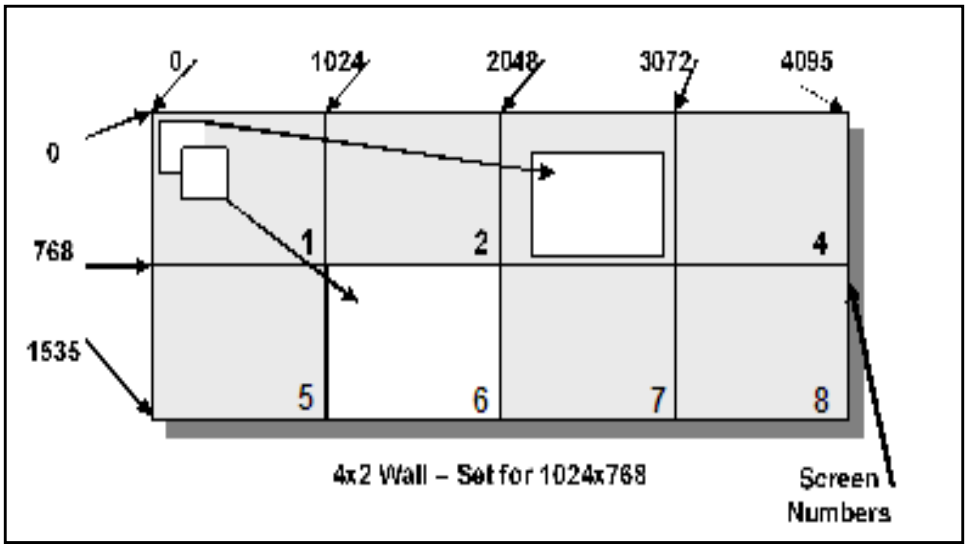


Figure 14 – Example Coordinates for Moving Windows

+Window SetState { { 101 } 2 1 7183 2095 50 320 240 { -1 } }
make Video visible, place at (100, 100), size at 320x240 and place on top

+Window SetState { { 102 } 3 9 7183 1024 768 1024 768 { -1 } }
make RGB visible, framed, place at (1024, 768), size at 1024x768 and place it on top

3.7.6 Getting & Setting Parameters

Let's look at some of the ways to look at and set different parameters. Return codes are omitted for **Set** commands assuming a successful execution.

3.7.6.1 LiveVideo Parameters

This section will examine ways of checking and setting LiveVideo parameters.

3.7.6.2 Get/Set Channel

The **GetChannel** method is used to get the channel and input type of the specified window.

+LiveVideoSys GetChannel { 101 }
returns the channel that is set for LiveVideo window 101
returns:
=00000000 1 (successful command channel 1)

+LiveVideoSys SetChannel { 101 } 3
sets the channel for LiveVideo window 101 to 3

Note It is important to note that when using the **SetChannel** Method with either the **LiveVideSys** or **RGBSys** Objects, you define the type of window (either RGB or LiveVideo) by the type of input it receives. **SetChannel** can be used to change the sub-type of a window.

3.7.6.3 Get/Set Video Source

The **GetVideoSource** method is used to get the format and input type of the specified window.

+LiveVideoSys GetVideoSource { 101 }
returns the video source settings
returns:
=00000000 { 3 1 0 }
(success - Set=3 for both Format and Type – 1 for NTSC, 0 for Composite)

+LiveVideoSys SetVideoSource { 101 } { 3 1 0 }
Set=3 for both Format and Type – 1 for NTSC, 0 for Composite
Refer to "**Table 6 on page 47**" and "**Table 7 on page 48**".

3.7.6.4 Get/Set Crop

The **GetCrop** Method is used to get the cropping parameters of the specified window.

+GalWinSys GetCrop { 101 }
asks for cropping settings
returns:
=00000000 { 0 0 640 480 } (success - {nX nY nW nH }

+GalWinSys SetCrop { 101 } { 10 10 600 400 }
sets Crop by relation to the normal image size. nX and nY specify the pixels removed from the top and bottom. nW and nH specify the size of the cropped image. A 640x480 window set at 10 10 600 400 will have 10 pixels removed from the top, 10 pixels removed from the left, 30 and 70 removed from the right and bottom.

ImageWidth - nX - (right pixels) = nW
or 640 – 10 – 30 = 600
ImageHeight - nY - (bottom pixels) = nH
or 480 – 10 – 70 = 400

Note	Remember that values for X and W (Left and Right) must be specified as even pixels, whereas, Y and H (Top and Bottom) can accommodate both even and odd values.

Example
The following scenario will demonstrate how even and odd values work with cropping values.

A DVI window is set to 1600 x 1200 with the following cropping values—**X=20 Y=20 W=20 H=20** through the commands:

+GalWinSys GetCrop { 10018 }
Returns:
=00000000 { 20 20 1560 1160 }

Next, cropping is set by 1 extra pixel for X and Y:

+GalWinSys SetCrop { 10018 } { 21 21 1559 1159 }
Returns:
=00000000

The odd values are accepted by the acknowledged return of =00000000, however, the DVI Capture **Window Properties** (in the GUI) now report the following values:

X=22 Y=21 W=18 H=20

Notice that the **X**-value has been adjusted to **22** and the **W** to **18** as only even numbers are allowed for X and W, while the **Y**-value has accepted the odd value of **21**.

Only when another **Get** command is issued again does the corrected X-value (by rounding of to the next even number) show up:

+GalWinSys GetCrop { 10018 }
Returns:
=00000000 { 22 21 1560 1159 }

3.7.6.5 Get Kind

The **GetKind** Method answers the question “What window **types** can this object work on?”

+LiveVideoSys GetKind

asks for SubSystemKind **this** Object can operate on.

Returns:

=00000000 2

Success - reports 2 (LiveVideo) Since we have inquired under LiveVideoSys, it tells us that LiveVideoSys can operate on LiveVideo windows.

+GalWinSys GetKind

Returns:

=00000000 1 (1 = all ControlPoint Windows)

3.7.6.6 QueryAllWindows

The **QueryAllWindows** method will report the status of all windows. Several objects can use this method.

+WinServer QueryAllWindows (Returns all windows)
+GalWinSys QueryAllWindows (Returns all windows)
+RGBSys QueryAllWindows (Returns only RGB windows)
+LiveVideoSys QueryAllWindows (Returns only Video windows)

+LiveVideoSys QueryAllWindows

returns settings on all windows – Because we have asked under **LiveVideoSys**, **QueryAllWindows** returns only LiveVideo window settings.

returns:

=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } } { { 102 } 2 1 7183 152 21 320 240 { 102 } }

Success - The first number is the count of windows returned followed by each window settings). For **Set**, returned value of 7183 = 0x1C0F. Note bit positions for the following flags are set. **wsVisible**, **wsMinimized**, **wsMaximized**, **wsFramed**, **wsPosition**, **wsSize**, and **wsZOrder**. This indicates valid data in these fields.

nState flags together with **x, y, w, h** and **ZOrder** represent the current state of the window. These fields carry data if marked as valid in **Set**. For example, if you want to see if a window is minimized, you first have to make sure that **wsMinimized** in **Set** is set to 1, which means that the bit in **nState** carries info about the minimized state of the window. Then check the bit **wsMinimized** in **nState** to see if the window is minimized or not. Do not get confused because the name of the flag is the same for **Set** and **nState** - **Set** verifies that the corresponding bit in **nState** is valid and means something.

3.7.6.7 Get Input Size

The **GetInputSize** method reports the current image size of the specified window.

+LiveVideoSys GetInputSize { 101 }

returns the size of the input to this window.

returns:

=00000000 { 640 480 } (success – input is NTSC so is 640x480)

3.7.6.8 Get/Set Image Balance

The **Get/SetImgBalance** Method returns/sets the image balance of the specified window. (Refer to **"Get/Set Image Balance" on page 145** under **RGB Parameters**).

+LiveVideoSys GetImgBalance { 101 }

returns image balance settings for LiveVideo window

returns:

=00000000 { 27 0 100 0 0 100 }

Success - the first number is the Set bits and tells you which fields are valid.

Table 79: Get/Set Image Balance

Set Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	Sat	Hue	Gam	Con	Bri

27 = 16 + 8 + 2 + 1 or Saturation, Hue, Contrast, and Brightness. Gamma (4) is not yet implemented and will be 0.

The following values equate to the default settings:

+LiveVideoSys SetImgBalance { WinId } { Set Bri Con Gam Hue Sat }

+LiveVideoSys SetImgBalance { 101 } { 27 0 100 0 0 100 }

sets image balance parameters for LiveVideo window

Table 80: Default Image Balance Values

Parameter	Min.	Default	Max.
Brightness	-100%	0	+100%
Contrast	0%	100	200%
Hue	-180°	0°	+180°
Saturation	0%	100	200%

All values in the above table can be set at one time with the following commands:

+LiveVideoSys ApplyDefaults { 123 }

or

+GalWinSys ApplyDef

3.7.6.9 Get/Set Video Source

The **Get/SetVideoSource** Method allows you to get/set the format of the video signal (NTSC, PAL, etc.) and the signal type (Composite or S-Video) for the specified window.

+LiveVideoSys GetVideoSource { 101 }

returns video source information for the specified LiveVideo window

returns:

=00000000 { 3 1 0 }

Success - the first number is the Set bits and tells you which fields are valid.

Table 81: Get/Set Video

Set Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	0	0	0	Type	Format

3 = 2 + 1 or Type and Format. The following values equate to the current settings:

+LiveVideoSys SetVideoSource { WinId } { Set Format Type }

+LiveVideoSys SetVideoSource { 101 } { 3 2 1 }
sets video source to S-video and PAL parameters for LiveVideo window

Table 82: LiveVideoFormat

LiveVideoFormat	
LIVE_VIDEO_NTSC	0
LIVE_VIDEO_NTSCJ	1
LIVE_VIDEO_PAL	2
LIVE_VIDEO_PALM	3
LIVE_VIDEO_PALN	4
LIVE_VIDEO_SECAM	5
LIVE_VIDEO_PALNc	6
LIVE_VIDEO_NTSC443	7
LIVE_VIDEO_PAL60	8

Table 83: Type and Format Values

LiveVideoType	
LIVE_VIDEO_COMPOSITE	0
LIVE_VIDEO_SVIDEO	1

3.7.7 RGB Parameters

This section will examine ways of checking and setting RGB parameters. Many parameters have already been shown in the LiveVideo examples above; this section shows only commands not yet exemplified here.

3.7.7.1 Get/Set Timing

The **Get/SetTiming** Method returns/sets the timing parameters for the specified RGB window.

+RGBSys GetTiming { 123 }
returns the timing parameters for the spedified RGB window.

Returns:
{bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }

=00000000 { 1 640 800 144 480 525 35 3 60 0 0 0 }

+RGBSys SetTiming { 123 } { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }
Sets the timing parameters for the specified RGB window. Note, you should perform a **Set Timing** immediately after creating an RGB window; otherwise, the execution will include the wait for the auto-detect to finish.

+RGBSys SetTiming { 123 } { 1 640 800 144 480 525 35 3 60 0 0 0 }

Set Bit Positions										
1024	512	256	128	64	32	16	8	4	2	1
VsyncNeg	HsyncNeg	SyncType	Vfreq	Phase	Voff	Vtot	Hheigh	Hoff	Htot	Width

Note **bValid** is a Boolian function (1/0).

3.7.7.2 Get/Set Image Balance

The **Get/SetImgBalance** Method returns/sets only the contrast and brightness of the specified RGB window.

+RGBSys GetImgBalance { 102 }
asks for image balance settings

returns:

=00000000 { 3 0 100 0 0 100 }

Success - the first number is the Set bits and tells you which fields are valid.

Table 84: Get/Set Image Balance

Set Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	Sat	Hue	Gam	Con	Bri

3 = 2 + 1 or Contrast, and Brightness. Saturation, Hue, and Gamma do not apply to RGB and will be 0. The following values equate to the current settings.

+RGBSys SetImgBalance { 102 } { 3 0 100 0 0 0 }

sets image balance parameters for RGB window

3.7.7.3 Get InputSize

The **GetInputSize** Method reports the current size of the specified RGB window.

+RGBSys GetInputSize { 102 }

returns input size for this RGB window.

returns:

=00000000 { 640 480 }

Values returned of (0 0) indicate an invalid or hidden RGB window.

3.7.7.4 Detect Timing

The **DetectTiming** Method causes a timing calculation of the inout signal for the specified RGB window. **DetectTiming** will take between 300 milliseconds and 2 seconds, depending on the input format.

+RGBSys DetectTiming { 102 }

Forces resample of input signal.

returns:

{ bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }

=00000000 { 1 640 480 144 480 525 35 3 60 0 0 0 }

Note The **+RGBSys DetectTiming { window_id }** command can be used to make ControlPoint detect the VGA signals faster by forcing it from the script instead of waiting for auto-detect to execute.
Call **DetectTiming** for every VGA window. Either dynamically query the Window IDs or use user-defined IDs in your layouts so that the Window IDs are known to you.

3.7.8 Get/Set for Any Window

This section will examine ways of checking and setting parameters for any window.

3.7.8.1 Get/Set Title

The **GetTitle** Method returns or set the current title of the specified window.

+Window GetTitle { 102 }
returns title settings for this RGB window.

returns:
=00000000 “RGB"

+Window SetTitle { 102 } “titlename”
sets the title as named for this RGB window.

3.7.8.2 Get/Set Lock Aspect Ratio

Use the **Get/SetState** method to get or set the aspect ratio lock for the specified window. See explanation on following pages.

3.7.8.3 Get/Set Window State

The Object **Window**, **Get/SetState** Method is used to determine or set the state of a window. From this single command line, you can set or determine the following parameters in a given window.

- Visibility (visible or hidden)
- Minimized (to an icon on the task bar)
- Maximized (full screen)
- Framed (has a window frame or bare image)

- Lock Aspect Ratio (locked/unlocked)
- Position (Position of window on the desk top (x, y))
- Size (size of the window image (without frame) (x, y))
- ZOrder (place a window on top of, or under another window)

The **SetState** Method uses several flags and can be somewhat complicated. The **Set** flags determine which state fields will be valid either for a **Get** or a **Set**. Non-valid data fields may have data but may not be correct if not specified as valid in the **Set** field.

The nState field is used to specify the following window states:

- Visibility (visible or hidden)
- Minimized (to an icon on the task bar)
- Maximized (full screen)
- Framed (has a window frame or bare image)
- Lock Aspect Ration (locked/unlocked)

Note The **Set** flags determine which state fields will be valid. Data fields returned with **Get** may not be valid if associated nStateChange bits are not set.

Get a Window's State

The **GetState** command is shown below for window number 123 with the response following.

```
+Window GetState { 123 }
=00000000 { { 123 } 2 9 7183 137 272 320 240 { -1 } }
```

Now, lets take the response apart:

0 = success

123 = window Id

2 = the window type – video

9 = Framed and visible *

Visible = 1 = 1

Minimized= 2 = 0

Maximized = 4 = 0

Framed= 8 = 8

7183 or 0x1C0F = all fields valid – see following page
Position of window is at (x=137, y=272)
Size of the window is (w=320 by h=240)
ZAfter is a window ID number to put this window behind
-1 = the window is on top and has focus - -2 = send to back.
*a window cannot be both minimized and maximized; the maximum value can only be B or D.

Refer to **"Understanding and Using Flags" on page 122**.

Set a Window's State

The **SetState** command is shown below for window 123. Note the labeling of the specific fields.

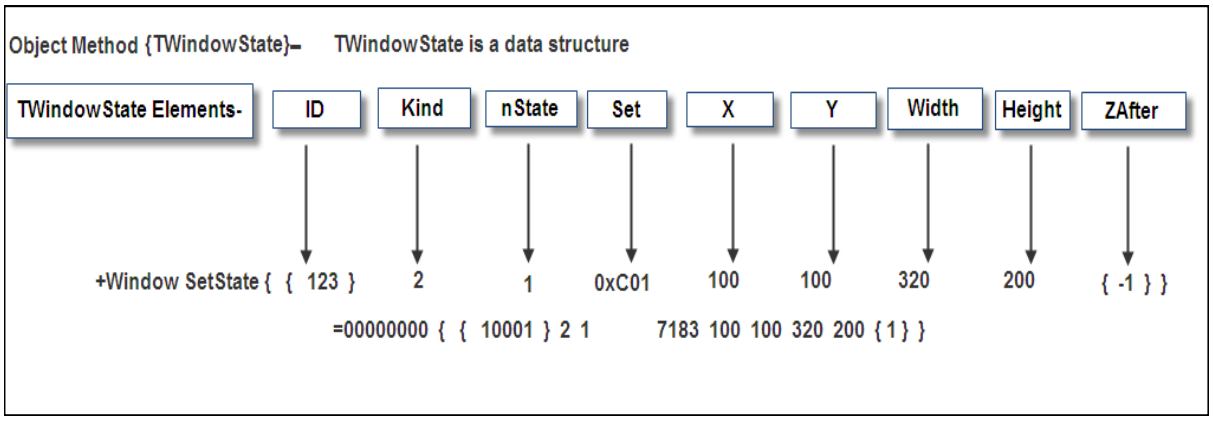


Figure 15 – Example of SetState

Set Values
1 = 1 = ZAfter
C = 8 + 4 = position and size
0 = 0
9= 8 +1 = Framed and visible

Note It is important to note that the **Set** field determines the validity of the data in all other data fields.

While the **nState** bits are used to change the respective parameters, the **Set** bits make them valid.

Table 85: nState Bits

nState Bits	
wsVisible	0x0001
wsMinimized	0x0002
wsMaximized	0x0004
wsFramed	0x0008
wsLockAspect	0x0010
wsAlwaysOnTop	0x0020

A window cannot be both minimized and maximized at the same time so the max. value will either be 9 + 2 or 9 + 4, 11 or 13 (1B or 1D).

Table 86: nStateChange Bits

nStateChange Bits	
wsVisible	0x0001
wsMinimized	0x0002
wsMaximized	0x0004
wsFramed	0x0008
wsLockAspect	0x0010
wsAlwaysOnTop	0x0020
wsZAfter	0x1000

0x1C1F is the maximum value:
1 in the first column indicates ZAfter
C in the second column indicates both size and position are valid.
1 in the third column indicates LockAspect (lock aspect ratio).
F for all fields in nState being valid, although the actual nState cannot be F.

An example:
If you only want to set a window on top, you would use the following command.

+Window SetState { { 123 } 2 0 0x1000 0 0 0 0 { -1 } }
=00000000 { { 123 } 2 9 7183 137 272 320 240 { -1 } }

123 for the window Id,
2 for a video window
0 not changing nState
1000 hex for ZOrder (0x1000=4096)
four 0's for place holders
-1 for On Top & has focus

Note that there is considerable data returned, and that the **Set** value is 7183 making all returned fields valid.

The **SetState** Method can be used to set the position, size, and visibly of a named window. It can also be used to minimize, maximize and add or remove the window frame to/from the window image.

3.7.8.4 System Monitoring QueryAllValues

Use **QueryAllValues** to retrieve all monitored system parameters. The **QueryAllValues** command returns 24 separate parameters.

+SysMonEx QueryAllValues

Returns all monitored values. This object uses the **TCPSysMonValue_array_t data structure**

{ =00000000 { n elements { e1 } { e2 } … }

{ success { n elements { code value threshold status } { e2 } …}

```
=00000000
{ 16 { 1 30 30 0 }          { 2 29 29 0 }            { 3 5921 5921 0 }
  { 4 6026 6026 0 }         { 5 1456 1456 0 }        { 6 1456 1456 0 }
  { 16 3248 3248 0 }        { 160 3312 3312 0 }      { 32 4731 4731 0 }
  { 48 11917 11917 0 }      { 64 -11410 -11410 0 }   { 176 1792 1792 0 }
  { 256 32 32 0 }           { 16779264 0 1000 2 }    { 16781312 0 1000 2 }
  { 16783360 0 1000 2 } }
```

The above example has been formatted for ease of reading.

TCPSysMonValue
code the code of the monitored value (see following page)
value the monitored value itself
threshold threshold value that caused the alarm. This field is valid ONLY if there is alarm condition
status status of the value (see following page)

Parameter codes

Table 87: Parameter Codes

Code	Parameter	hex		Code	Parameter	hex
1	CPU1 temperature	01		256	motherboard temp	100
2	CPU2 Temp	02		512	chassis temp	200
3	CPU1 Fan speed	03				
4	CPU2 Fan Speed	04		2048	Chassis fan 1 speed	800
5	CPU1 core voltage	05				
6	CPU2 core voltage	06		4096	chassis fan 2 speed	1000
7	CPU1 socket temp	07				
8	CPU2 socket temp	08		6144	chassis fan 3 speed	1800
				8192	chassis blower 1 speed	2000
16	3.3V	10		10240	chassis blower 2 speed	2800
32	+5V	20		12288	chassis blower 3 speed	3000
48	+12V	30				
64	-12V	40				
80	2.5V	50				
96	Battery	60				
112	1.5V	70				
128	- 5V	80				
144	5V standby	90				
160	3.3V standby	A0				
176	1.8V	B0				

The listed parameter codes cover all Fusion and Vizion models.

Example:
To illustrate the use of the Module and Parameter Codes as shown in the tables, consider an installation with two chassis (a main chassis and an expansion chassis). Each chassis has a set of fans. In order to show values for fans in two chassis, the code for fan 1 in the main chassis is 2048 + 0x01000000 = 0x01000800, and the code for fan 1 in the expansion chassis is 2048 + 0x02000000 = 0x02000800.

3.7.8.5 Get Record

GetRecord is used to retrieve a record from the event log. Before a record can be retrieved the position in the record must first be set.

Use the mask bits for setting position flags, before getting an event log record.

+EventLog SetPosition flag recNum
+EventLog SetPosition 2 0

This example sets the **current** position in the event log. The **recNumber** is the record number at which to position the record pointer, the **flags** bits determines the seek mode:

Bit 0 (mask 1) – go to the oldest record in the event log
Bit 1 (mask 2) – go to the latest (newest) event
Bit 2 (mask 4) – go to the record specified in **recNum**
Bit 3 (mask 8) – after positioning, the reading will be done in forward motion
Bit 4 (mask 16) – after positioning, the reading will be done in backward motion

Note **recNum** will only be valid for Bit2 (mask 4) but the position must be filled when sending the method. Failure to fill all fields will cause an error.

In order to retrieve a record you must first set the position in the record then get the record.

+EventLog SetPosition 2 0
+EventLog GetRecord

This command reads one record from the event log at the latest entered record. It returns the record header information, the name of the source that generated the event and the text of the event. The position is automatically moved to the next record depending on the seek mode (forwards or backwards) specified in flag Bit 3 or Bit 4.

=00000000 { header } “source” “text”

Where header is described below, source is the program element that caused the error, and text is the text of the event log message

=00000000 { 1074003997 0 20 1047388718 } "GalSysMon" "Chassis Fan-3 speed back to normal. Current speed is 2463 RPM"

=00000001 { 0 0 0 0 } “” “”

This is really an error condition – indicating end of file (EOF). You will get this return condition if, after retrieving the newest record you, you try to retrieve another record without first setting the pointer. At this point, the pointer has advanced past the last (newest) record. See also **Bit3** and **Bit4** for setting the direction for moving the pointer after reading a record.

header - { nEventID CPEventLogRecType_t nRecNumber nTime }
nEventID = the identifier of the event
The nEventID field is the unique identifier of the event.

CPEventLogRecType_t nType = type of the event (severity)
The nType field shows the type (severity) of the event:
0 – Information message
1 – Error event
2 – Warning event

nRecNumber = the record number in the event log
The nRecNumber field shows the position of the record in the system event log.

nTime = encoded time of the event
The **nTime** field has the encoded time of the event. It is the number of seconds elapsed since midnight (00:00:00) January 1, 1970.

3.7.9 Managing Users

This section shows examples of how to use the **User Management** commands.

3.7.9.1 Add User

The **AddUser** Method allows the administrator to add users to the authentication list.

+UserMan AddUser “username” “password” level

sets up a new user with user name, password and user level.

Note This is an **admin only** command.

3.7.9.2 Delete User

The **DeleteUser** Method allows the administrator to remove users from the authentication list.

+UserMan DeleteUser “username”

deletes user specified in username

Note This is an **admin only** command.

3.7.9.3 List Users

The **EnumUsersCS** Method lists all users in the authentication list.

+UserMan EnumUsersCS”

lists all users.

returns:

=00000000 “admin,localuser,test”

3.7.9.4 Get User Info

The **GetUserInfo** Method returns the password and level of the specified user.

+UserMan GetUserInfo “username”

returns password and level

returns:

=00000000 "password" 0

Note This is an **admin only** command.

3.7.9.5 Set User Info

The **SetUserInfo** Method allows the administrator to edit the user's password and level.

+UserMan SetUserInfo "username" "password" level

edits password, and level for username.

Note This is an **admin only** command.

3.7.9.6 Change Password

The **ChangePassword** Method allows the user to change his own password.

+UserMan ChangePassword "oldpassword" "newpassword"

changes a user's password as specified.

Note This is a **user only** command. Only the logged-in user can change his/her own password.

3.7.10 Changing a Window's Type

It is a simple matter to change an RGB window to a LiveVideo Window or the reverse. Each window is assigned an ID number that is used to control the window's content. The use of the ID number with the **SetChannel** command and the appropriate object, then, lets you specify the type of window and channel for the specified window. The example below shows changing an RGB window of ID number **10002** to a LiveVideo window using video input channel 5. The window, with ID number grater than 10,000 was a ControlPoint Server supplied ID number.

+LiveVideoSys SetChannel { 10002 } 5

where 10002 is previous RGB window number

+ LiveVideoSys Start { 10002 }

this is similar to the manual operation using the window properties dialog.

Note	The **Start** command must be used whenever the SubType of a window is changed.

3.7.11 Deleting a Window

The **DeleteWindow** Method deletes the specified window.

+WinServer DeleteWindow { 123 }

deletes window 123

Note	Only windows that have a system menu can be closed with ControlPoint DeleteWindow. This excludes menus, tool tip pop-ups, and other temporary pop-up windows.

- All windows that have title and/or close icon in the title-bar can be closed.

- If you try to execute DeleteWindow on a non-qualifying window, the server will return the following Error Code: 800705A9. For more information, refer to **“Error Codes” on page 175**.

3.7.12 Layouts

Use the **Layout** commands as outlined below when working with layouts.

Save Layout
+WinServer SaveLayout “layoutname” { 0 }

Set Layout
+WinServer SetLayout “layoutname”

Delete Layout
+WinServer DeleteLayout “layoutname”

Query Last Set Layout
+WinServer QueryLastSetLayout “Name of Last Layout”

Note	To avoid the situation where the hardware and ControlPoint are detecting VGA timing while the signal is being switched, close all ControlPoint windows first, switch the signal, then apply the new layout.

3.7.12.1 Procedure to Avoid Auto-detection of Timing while Switching VGA/DVI Signals

1. Issue the following command:

   +WinServer SetLayout “” – empty layout name closes all ControlPoint
   +windows
   Switch AMX DGX

2. Delay one or two frames time (depending on the switch)

   +WinServer SetLayout “your_layout_name”

Furthermore, the **+RGBSys DetectTiming { window_id }** command can be used to make ControlPoint detect the VGA signals faster by forcing it from the script instead of waiting for auto-detect to execute.

Call **DetectTiming** for every VGA window. Either dynamically query the Window IDs or use user-defined IDs in layouts so that the Window IDs are familiar.

3.7.13 Window Frame & Title

Use the **Window Frame & Title** commands as outlined below when customizing the window’s frame and title.

Set Frame Info
+Window SetFrameInfo { 123 } { 7 10 16777219 1 0 }
Get Frame Info
+Window GetFrameInfo { 123 }

FrameInfo

SetFrameInfo sets the frame information for a window. This information includes the width of the frame, the color of the frame and whether to show the title text or not.

Note When working with the frame, you must first use the **Window SetState** command to turn the frame on or off.

To turn the frame on use the command below.
+Window SetState { { WinID } 2 8 0x0008 0 0 0 0 { 0 } }

To turn the frame off use the command below.

+Window SetState { { WinID } 2 0 0x0008 0 0 0 0 { 0 } }

Where WinID is the ID for the window you want to frame.

The Frame Info method under the Window object allows you to create, set color and size of a frame around a selected window. All ControlPoint windows can be framed. Zero or one must be use as a place holder for options not used. See Usage below.

+Object Method { WinId } { Set [options] }

+Window SetFrameInfo { WinId }
{ Set FrameWidth FrameColor ShowTitle ShowUserData }

Frame width (FW) in pixels – 0 to 100
Frame color (FC) = RGB
Red = 1 to 255
Blue = 1 to 255
Green = 1 to 255
White is all color on – (256) x (256) x (256) = 0xFFFFFF or 16,777,215
Black is all color off – 000 = 0x000000.
Example – green only – 00 FF 00 = 65,280 = 0x00FF00
Show title = 1 = display title – 0 = title off

+Window SetFrameInfo { 123 } { 7 10 00FF00 1 0 }
Green only - Set = 7 width, color and title all valid.

Table 88: FrameInfo Bit Positions

FrameInfo Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	0	SU	ST	FC	FW

+Window SetFrameInfo { 123 } { 7 10 16777219 1 0 } (Sets the window frame to width = 10, color = white, show title text = Yes)
+Window SetFrameInfo { 123 } { 2 0 255 0 0 } (Sets the window frame to color = red, ignoring other parameters)

GetFrameInfo returns the frame information from a window.
+Window GetFrameInfo { 123 }
=00000000 { 123 } { 7 10 16777219 1 0 }

TitleInfo

SetTitleInfo sets the title information for a window. This information includes the text color, the position of the title bar (on top of the window or at bottom), the minimum height of the title bar, horizontal justified and vertical justified.

Set Title Info
+Window SetTitleInfo { 123 } { 1 16777219 0 0 0 0 }

Get Title Info
+Window GetTitleInfo { 123 }

The **Title Info** method under the Window object allows you to, set text color, text position, title position, minimum title bar height, title horizontal justification, and title vertical justification. Zero or one must be use as a place holder for options not used. See Usage below.

+Object Method { WinId } { Set [options] }

+Window SetTitleInfo { WinId }
{ Set TextColor BarPos MinBarHeight HorizJust VertJust }

Text color (TC) = R G B color – see **FrameInfo** (previous section)
Bar Position (BP)= 0 on top of frame – 1 on bottom of frame
Bar height (BH) = 0 to 100 pixels
Horiz. Just (HJ) = 0 = left – 1 = middle – 2 = right
Vert. Just (VJ) = 0 = top – 1 = middle – 2 = bottom

+Window SetTitleInfo { 123 } { 31 00FF00 0 20 1 1 }
Green only - Set = 31 TextColor BarPos MinBarHeight HorizJust VertJust all valid. Text color = green, on top, 20 pixels high, vertical and horizontal center.

Table 89: SetTitleInfo Bit Positions

SetTitleInfo Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	VJ	HJ	MH	BP	TC

+Window SetTitleInfo { wid } { Set TextColor BarPos MinBarHeight HorizJust VertJust }

+Window SetTitleInfo { 123 } { 1 0 0 0 0 0 } (Sets the text color to black, and the other parameters are ignored)

+Window SetTitleInfo { 123 } { 10 0 1 0 1 0 } (Sets the title bar at the bottom of the window, the text is horizontal center justified, and the other parameters are ignored)

TitleFontInfo

The Title font Info method under the Window object allows you to set font size and font name Zero or one must be use as a place holder for options not used. See Usage below.

+Object Method { WinId } { Set [options] }

+Window SetTitleFontInfo { WinId } { Set TextSize TextFlags FontName }

Text size (TS) = 8 to 72 pixels
Text flag (TF) = 0 – not used
Font name (FN) = name of font wanted – must be a valid font name

+Window SetTitleFontInfo { 123 } { 3 12 0 Arial }
Set = 5 font name and font size valid.

Table 90: SetTitleFontInfo Bit Positions

SetTitleFontInfo Bit Positions							
128	64	32	16	8	4	2	1
0	0	0	0	0	FN	TF	TS

SetTitleFontInfo sets the title font information for a window. This information includes the text size, text flags and the font name.

+Window SetTitleFontInfo { wid }

{ Set TextSize Flags FontName }

+Window SetTitleFontInfo { 123 } { 5 14 0 “Arial” } (sets the text size = 14, font name = “Arial”, and ignores the flags)

GetTitleFontInfo returns the title font information from a window.
+Window GetTitleFontInfo { 123 }
=00000000 { 123 } { 5 14 0 “Arial” }

GetTitleFontInfo returns the title information from a window.
+Window GetTitleFontInfo { 123 }
=00000000 { 123 } { 10 0 1 0 1 0 }

Set TitleFontInfo
+Window SetTitleFontInfo { 123 } { 5 14 0 “Arial” }

Get TitleFontInfo
+Window GetTitleFontInfo { 123 }

3.7.14 CPWeb Window

Use the **CPWebWindow** commands as outlined below when working with a web window to set or get a URL.

Set URL
+CPWebSys SetURL { WinId } “http://www.jupiter.com”
Sets the URL for the specified web window.

Get URL
+CPWebSys GetURL { WinId }
Returns the current page URL for the specified web window.

3.7.15 PictureViewer Window

Use the **PictureViewer** window command to show or get a picture window file path.

NewWindow creates a new PictureViewer window and returns the window ID for that window.
+PictureViewerSys NewWindow (returns new window ID)
=00010123

NewWindowWithId creates a new PictureViewer window with a user-specified window ID.

+PictureViewerSys NewWindowWithId { 123 }
=00000123

ShowPicture

+PictureViewerSys ShowPicture { WinId }
“C:/MyPictures/APic.bmp”

Displays an image in the specified picture window from the given path.

Get FileName

+PictureViewerSys GetFileName { WinId }

Returns the path for the displayed image in the specified window.

SetTextMode sets the PictureViewer text mode to on, off, or on with text scroll capability. The window ID must be specified.

+PictureViewerSys SetTextMode { WinId } 1

where 0 = text mode off; 1 = text mode on; 2 = text mode on with text scroll

=00000000

GetTextMode returns the PictureViewer window’s (using WinId) text mode status.

+PictureViewerSys GetTextMode { WinId }

where 0 = text mode off; 1 = text mode on; 2 = text mode on with text scroll

=00000000 1

SetText sets the entered text on the PictureViewer window. The window ID must be specified.

+PictureViewerSys SetText { WinId } “Front”
=00000000

GetText returns the entered text on the PictureViewer window (using WinId).

+PictureViewerSys GetText { WinId }
=00000000 “Front”

3.7.16 Screen Test Pattern

ScreenUtil

The ScreenUtil object and its methods are used for screen tests. The Screen Utility is used to setup seven different test patterns five of which have several options allowed. All patterns except the Screen ID can be displayed either across the whole wall (VirtualScreen) or on individual displays (Single Display). Zero or one must be use as a place holder for options not used. See Usage below.

There are two steps required for using the Screen Utility.
First, you must open the respective utility page (Show Pattern)
Second, you can setup and make the changes you wish to display (Set Pattern Properties).

Use the commands below to select the function you want.

Show Pattern

“stdpat“ - display the Standard test pattern
“screenid“ - display the screen ID
“phasepat“ - display the Phase pattern
“bitmap“ - display a bitmap image
“colorpat“ - display the Color Bar pattern
“colorpat“ - display the Grid pattern
“” - the null function terminates the screen test

+ScreenUtil SetPatternProp “setpatternprop”
{ Set SingleScreen BarType GridCircle FgColorRed FgColorGreen FgColorBlue BgColorRed BgColorGreen BgColorBlue SmoothGradient LineSpacing }

+Object Method “function“

+ScreenUtil ShowPattern “colorpat”
Displays the color pattern page.

Note There are two steps required for using the Screen Utility. First, you must open the respective utility page (Show Pattern). Second, you can setup and make the changes you wish to display (Set Pattern Properties).

Show Pattern Properties

Once you have opened the function, you can make the changes to view the pattern you wish.

Table 91: Show Pattern Properties

Bit	Function	Description
1	SingleScreen	0 = Single Screen 1 = Full Screen (VirtualScreen)
2	BarType	0 = solid Color 1 = Vertical Bars 2 = Horizontal Bars
4	GridCircle	0 = Grid drawn without circles 1 = Grid drawn with circles
8	FgColorRed	0 = foreground color red off 1= foreground color red on
16	FGColorGreen	0 = foreground color green off 1= foreground color green on
32	FgColorBlue	0 = foreground color blue off 1= foreground color blue on
64	BgColorRed	0 = background color red off 1= background color red on
128	BGColorGreen	0 = background color green off 1= background color green on
256	BgColorBlue	0 = background color blue off 1= background color blue on
512	SmoothGradient	0 = Color pattern displayed as bars 1 = Color pattern displayed as smooth gradient
1024	LineSpacing	An integer between 1 and 200

Table 92: Show Pattern Properties Bit Positions

BIT POSITIONS													
…	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	LS	SG	BB	BG	BR	FB	FG	FR	GC	BT	SS

Standard Pattern

+ScreenUtility ShowPattern “stdpat”

Displays the standard color pattern.
No options for this function

Screen ID

+ScreenUtility ShowPattern “screenid”

Displays the output channel for each display device.
No options for this function

Phase Pattern

+ScreenUtility ShowPattern “phasepat”

Displays the phase pattern – every other pixel on/off.
No options for this function

Bitmap

+ScreenUtility ShowPattern “bitmap”

Displays the specified. image
No options for this function

Color Pattern

+ScreenUtility ShowPattern “colorpat”

Displays the color pattern page.

+ScreenUtility SetPatternProp “SetPatternProp” { set [options] }

Color Pattern displays a selected color as solid or 10 bars as gradient from color to black, or a smooth gradient from color to black.

+ScreenUtility SetPatternProp “SetPatternProp” { Set [options] }

{ 0x008 1 0 1 0 1 0 0 0 0 0 0 0 } Set = 1, single screen = on (0)
Bar type = 1 -- vertical (2), Foreground = green (16)
Set = 19 = 0x000013

Grid Pattern

+ScreenUtility ShowPattern “gridpat”

Displays a grid pattern on the screen.

+ScreenUtility SetPatternProp “SetPatternProp” { Set [options] }

Grid Pattern displays a pattern of horizontal and vertical lines.

+ScreenUtility ShowSetPatternProp “SetPatternProp”

{ 0x067F 1 0 1 1 1 1 0 0 1 1 20 } Single Screen on (1) – solid color off (0) circle on (4) - foreground = red + blue + green = white (8+16+32) -- background = blue (0+0+256) – gradient = bars (0) – line spacing on = 20

Set = 1+0+4+8+16+32+64+0+0+0+512+1024=1661 = 0x067E

Exit Screen Utility

+ScreenUtility ShowPattern “”

This ‘null’ Function turns the Screen Utility off.

Table 93: ScreenUtility Functions and Options

	FUNCTION	standardpat	screenid	phasepat	colorpat	gridpat
Bit	**OPTIONS**					
1	SingleScreen (SS)	NA	NA	NA	1 = single screen 0 = Virtual Screen	1 = single screen 0 = VirtualScreen
2	BarType (BT)	NA	NA	NA	0 = solid color 1 = vertical lines 2 = horizontal line	0 = solid color 1 = vertical lines 2 = horizontal line
4	GridCircle (GC)	NA	NA	NA	1 = on 0 = off	1 = on 0 = off
	Foreground Color					
8	FgColorRed (FR)	NA	NA	NA	1 = on 0 = off	1 = on 0 = off
16	FGColorGreen (FG)	NA	NA	NA	1 = on 0 = off	1 = on 0 = off
32	FGColorBlue (FB)	NA	NA	NA	1 = on 0 = off	1 = on 0 = off
	Background Color					
64	BgColorRed (BR)	NA	NA	NA	NA	1 = on 0 = off
128	BGColorGreen (BG)	NA	NA	NA	NA	1 = on 0 = off
256	BgColorBlue (BG)	NA	NA	NA	NA	1 = on 0 = off
	Other Options					
512	SmoothGradient (SG)	NA	NA	NA	1 = on 0 = bars	1 = on 0 = bars
1024	LineSpacing (LS)	NA	NA	NA	NA	integer 1 to 200

Bitmap omitted in this table for clarity. See Usage above.
NA = not applicable – must have value as field position holder.

Table 94: ScreenUtility Bit Positions

BIT POSITIONS													
…	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	0	LS	SG	BB	BG	BR	FB	FG	FR	GC	BT	SS

3.8 Programming Considerations

This section discusses several considerations concerned with programming your external device and staying connected to the ControlPoint Server.

3.8.1 Getting Connected with the ControlPoint Server

You may use two different methods of connecting (described in detail previously in this manual) for the initial connection and authentication:

1. Connecting and authenticating from the command line when starting GalileoConnect on the Wall Controller

2. Connecting and authenticating from the touch panel or external device (GalileoConnect must be started on/from the Wall Controller itself).

These two methods are described in the following pages.

3.8.2 Connect and Login from the Command Line

When the command line connect method is used on the Wall Controller there is no way for ControlPoint to know that the external device is there or not. ControlPoint will wait for input and respond appropriately. Once connected, there is no way for the external device to know if it got disconnected from ControlPoint. The Auto Connect feature will try to re-connect.

There is startup help available for GalileoConnect. When GalileoConnect is started from the command line with a /? or /h switch, the dialog shows you the **default** parameters for GalileoConnect. These default parameters will connect the serial port to ControlPoint Server running on the **local machine** and talk to your COM1 port. GalileoConnect will perform authentication by using the default arguments – or the command line parameters must be entered (see following section).

3.8.3 Connect and Login from the Touch Panel

Some serial devices or touch panels allow for automatic connection and authentication. If the Wall Controller and the touch panel are powered on at the same time the touch panel will try to connect and authenticate while the Wall Controller is still booting the Windows operating system. This usually means that the connection fails. It is suggested that a connect loop be programmed continuously sending the connect string (**connect localhost 25456**) while waiting for an **OK** response. Once **OK** is received, you can then send your authentication strings, and continue with your first ControlPoint Protocol command string.

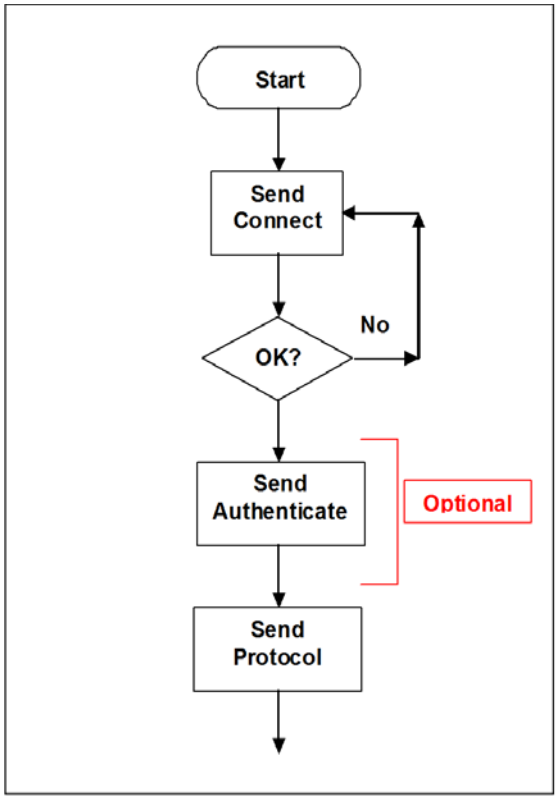


Figure 16 - Connect Loop Flow Chart

3.9 Command Line Default Values

The command line arguments used with **galcon.exe**, along with their default values are listed in the table below.

Table 95: Default GalCon Argument Values

Argument	Value
comport	COM1
baud	9600
server	localhost
port	25456
user	localuser
pass	(none)
auto	true
log	false

Using the galcon.exe Command

The default values listed in the table are pre-programmed into the system. The programmed default value will be used for any non-specified arguments when the **galcon.exe** command is executed. When the GalileoConnect Client is started with these arguments, each argument's default value will be changed to the last value used when executing the **galcon.exe** command. For example, the programmed default for the server is **localhost**. When you use the command-line **server** argument with the value **jupiter123**, the programmed value of **localhost** will be changed to **jupiter123.**

Using the Connect Command

The **Connect** command uses the default value for both **server** and **port** when executed without these values in the command. Furthermore, the values for **user** and **pass** will also be used for authentication if null strings are supplied for the next two lines expected after the **Connect** command.

Note Windows command line argument values will change the default values while serial device commands do not change them.

3.9.1 Detecting Disconnected Status

Data is only sent to the external device in asynchronous mode. That is, **responses** are sent only to **commands**. No data will be sent when not expected; therefore, there is no real-time notification of connection failure. When sending commands you should check for the success response **=00000000** and/or the **ER Socket Error**, which indicates the serial device is no longer connected to the ControlPoint Server.

See the following section and flow charts.

3.9.2 Staying connected with the ControlPoint Server

Once you have connected to the ControlPoint Server from your external device, there is no way for that device to know when the connection to the ControlPoint Server may have been broken.

Connections can be broken by several methods:

- Loss of power to the Wall Controller
- Rebooting or resetting the Wall Controller
- Quitting the ControlPoint Server
- Quitting the GalileoConnect Server
- Physical removal of the Ethernet connection on either system
- Physical disconnection of the serial cable from either system
- Failure of the Ethernet
- Sending improper commands to the ControlPoint Server (see **auto** below)

The **auto** command and argument for GalileoConnect (**auto=true**) allows for providing a method to reconnect to the server automatically should there be a disconnected status.

auto – enables/disables the auto-connect feature. If the value **true** is specified, GalileoConnect will automatically try to reconnect with the ControlPoint server if the connection breaks. An attempt is made about every five seconds. If the serial device can handle server connections programmatically, auto-connect can be disabled by specifying **false** for the argument.

Auto uses the default values stored from the default or last use of the command line variables for starting GalileoConnect, and will try to connect and authenticate when activated.

All commands sent to the Wall Controller from the serial device should be followed with a test for the success response code **=00000000** (success) and/or the **ER Socket Error**, which indicates a disconnected status. If you receive **ER** error response, you will need to send your connect and authentication strings again as described previously and then re-issue the original command that received the error status.

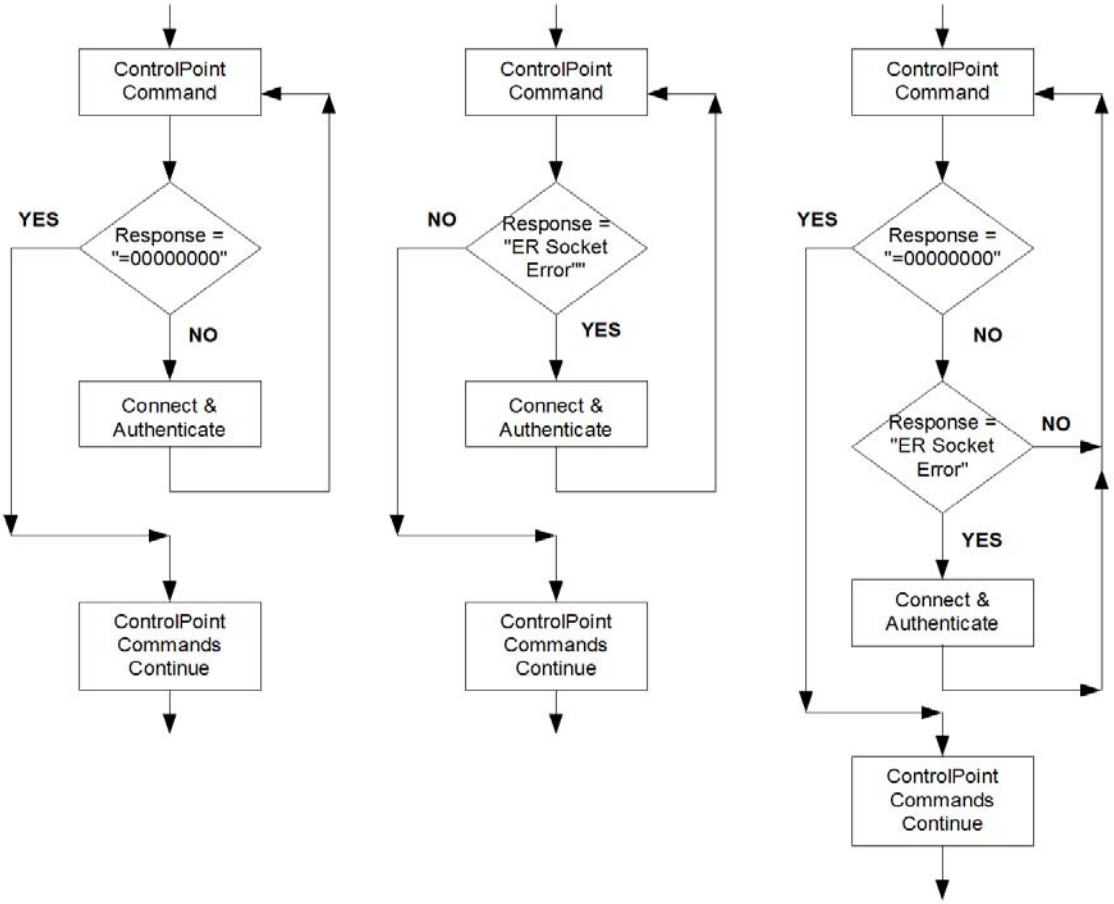


Figure 17 - Stay Connected Flow Charts

Caution You must test and wait for a return response from each command before sending the next command to the ControlPoint Server. Failure to do so can result in performance issues with commands and their respective actions and/or complete failure of subsequent commands to be recognized by the ControlPoint Server.

3.10 AMX Touch Panels

The following sections show several examples of AMX programming.

3.10.1 Serial Port Setup

The following is an example of setting up the serial port:

```
DEFINE_EVENT

DATA_EVENT[JWC]
{
ONLINE:
{
SEND_COMMAND JWC,'SET BAUD 9600,N,8,1 485 DISABLE'
SEND_COMMAND JWC,'HSOFF'    (* turn off hardware HS *)
SEND_COMMAND JWC,'XOFF'     (* turn off Software HS *)
```

3.10.2 Sending Commands

The following is an example of a program line to output a ControlPoint Protocol command. All text transmitted to the serial port must be in double quotes, text must be in single quotes.

```
    {
    SEND_STRING JWC,"'+WinServer SetLayout
  "ResetWindows"',13,10"
    }
```

Where **SEND_STRING** is the output command and **JWC** is the Jupiter Wall Controller (defined device). What follows is the actual command surrounded by double quotes. The actual protocol text is between single quotes. The numbers 13 and 10 represent the carriage return and line feed characters that must conclude every ControlPoint command line.

Each command must be on a single line and cannot be broken across multiple lines. The example below is shown on two lines but actually is only a single program line.

```
SEND_STRING JWC,"'+Window SetState { { 2 } 3 1 7168 10 10
199 150 { -1 } }',13,10"
```

3.10.3 Define Device

The following is an example of defining the device:

```
DEFINE_DEVICE

JWC  = 5001:2:0  (* RS232 Jupiter Wall Controller *)
            (* 9600, 8, N, 1 *)
```

Appendix A—Error Codes

A. Error Codes

This appendix lists the error codes in the ControlPoint Protocol. All error codes are expressed in their HEX notation (0x80040301).

A.1 Result Codes

This section lists the Result codes.

Table 96: Results Codes

Result Codes	Description
00000000	OK
00000001	Success, but false returned

A.2 Server Error Codes

The following tables list the error codes returned by their category

Table 97: Server Error Codes

ControlPoint Server Errors	Description
80040301	Invalid window ID
80040302	Not found
80040303	Window type mismatch
80040304	Invalid argument
80040305	Invalid archive version
80040306	Archive not found
80040307	Window ID already used
80040308	Invalid archive format
80070002	The specified file does not exist.
800705A9	Cannot show or remove the window in the way specified. This happens when a user attempts to close a system window that cannot be closed remotely (e.g., "Startup" pop-up menu or tooltip window). Only top-level windows with title frame and/or a close icon can be closed.

A.3 Protocol Parsing Error Codes

Table 98: Protocol Parsing Error Codes

Protocol Parsing Errors	Description
80040501	Not enough parameters supplied
80040502	Too may parameters supplied
80040503	Invalid RMC method name
80040504	Invalid RMC object name
80040505	Bad parameter format

A.4 Socket Connection Error Codes

Table 99: Connection and Server Error Codes

Socket Connection and Internal Server Execution Errors	Description
80040506	TCP/IP (socket) connection error
80040507	Server error
80040508	Unhandled exception
80040509	Division by zero exception
8004050A	Internal error
8004050B	Thread is already in an apartment
8004050C	Thread is not in an apartment
8004050D	Execution terminated
8004050E	Thread not initialized
8004050F	Object name is already registered
80040510	Error dispatching request
80040515	End of stream reached (internal error).
80040516	Bad protocol frame (internal error).

A.5 User Management Error Codes

Table 100: User Management Error Codes

User Management and Security	Description
80040511	Access is denied
80040512	Invalid user
80040513	User already exists
80040514	User Database error

A.6 RGB Related Error Codes

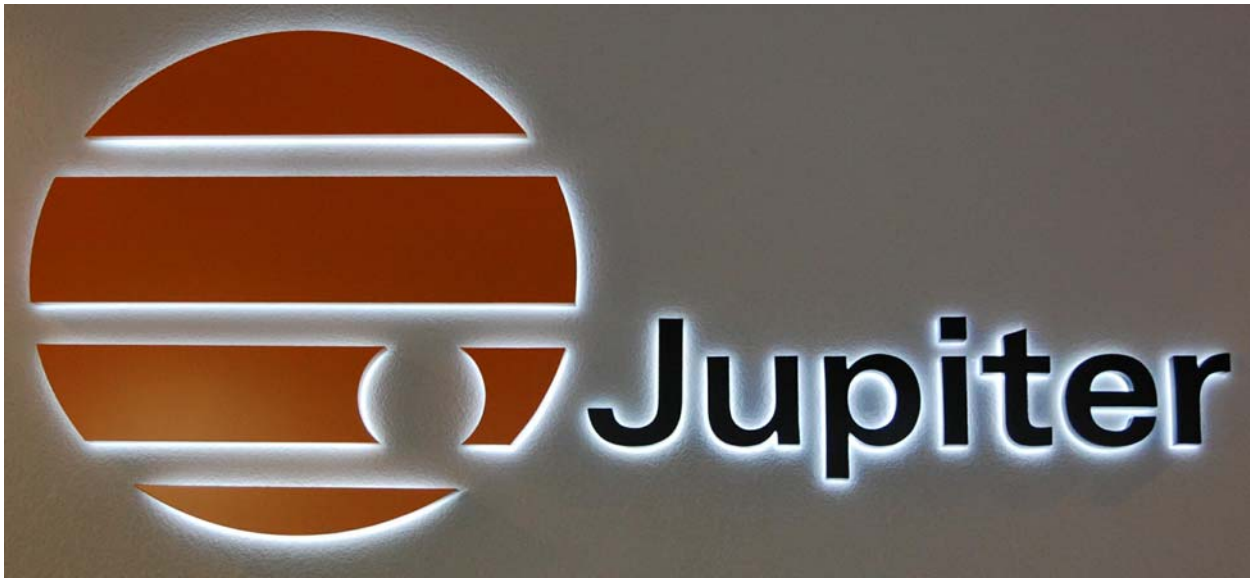
Table 101: RGB Related Error Codes

RGB Related Errors	Description
80040600	Unsupported Display Format, RGB
80040601	No timing has been selected, RGB
80040602	No input has been selected, RGB
80040603	No display information available, RGB
80040604	Engine is already running, RGB
80040605	There is no available engine
80040606	No device has been selected
80040607	The clip region cannot be handled by the device and is assumed empty
80040608	The input cannot be captured because of size, zoom or alignment limitations
80040609	No image balance has been set
8004060B	The ADC has not been initialized" (internal error).
80040700	Invalid window ID - Window not found
80040C02	Invalid Parameters

A.7 Video Related Error Codes

Table 102: Video Error Codes

Video Related Errors	Description
80040700	Invalid window ID - Window not found
80040701	Video is not enabled in this display mode, or the video input is not available in the current configuration
80040800	The DMA engine does not support the alignment


Jupiter

Appendix B—Alphabetic Command Listing of Objects

B. Alphabetic Command Listing of Objects

B.1 Objects

AppCtrl

Exec ([in, string] CmdLine)
+AppCtrl Exec “program path”
Executes the specified command-line on the server. Note that all paths are server-relative. Program path is the path on the server to the program you wish to run.

ConfigSys

GetServerInfo ([out] CPPlatformInfo, [out,string] versionInfo)
+ConfigSys GetServerInfo
=00000000 { PlatformCode ModelVersion ModelRevision OEMCode SerialNumber } "ControlPoint Version"
=00000000 { 19 0 1 0 4005 } "2.9.6414.296"
Returns information about the hardware platform and the version of the ControlPoint software installed on the server.

ListCfgGroup ([in] group_code, [out,string] objNames[]
+ConfigSys ListCfgGroup 1
=00000000 { 3 "1 RGB" "2 Rgb" "Video" }
Returns the object names in a configuration group. The configuration group codes are:
1 – Named Inputs
2 – Application Windows
3 – Layouts
4 - CPShare
5 - HotKeys
6 - Schedule
7 - Video Stream Source
8 - Fonts
9 - Images
11 - Batch (The number 10 is not used)
The result is an array of strings.

Objects

CPShareSys

NewWindow ([out] WinId_t pwid) **+CPShareSys NewWindow { 123 })** **=00010123** Returns server assigned ID
NewWindowWithId ([in] WinId_t wid) **+CPShareSys NewWindowWithId { 123 }** **=00000123** Returns user assigned ID
SetConnection ([in] WinId_t wid, [in,string] connection) **+CPShareSys SetConnection { 1 } “Elara”** **=00000000** returns the name of the connection object the CPShare window is using
GetConnection ([in] WinId_t wid, [out,string] connection) **+CPShareSys GetConnection { 1 }** **=00000000 “Elara”** (Queries CPShare window id = 1 for its currently selected connection name)
GetCredentials ([in] WinId_t wid, [out,string] server, [out, string] password) **+CPShareSys GetCredentials { 1 }** **=00000000 “Elara” “testpassword”** returns the server host address and the password used to authenticate the CPShare connection.

CPWebSys

NewWindow ([out] WinId_t pwid) **+CPWebSys NewWindow (returns new window ID)** **=00010123** Returns server assigned ID
NewWindowWithId ([in] WinId_t wid) **+CPWebSys NewWindowWithId { 123 }** **=00000123** Returns user assigned ID
SetURL ([in] WinId_t, [in,string]) **+CPWebSys SetURL { 10007 } "www.jupiter.com** **=00000000** Load a web page with the specified URL address.
GetURL ([in] WinId_t, [out,string]) **+CPWebSys GetURL { 123 }** **=00000000 "123 http://www.jupiter.com"** Return the URL address of the current web**.**

Debug

CloseFile ()
FlushToFile ([in, string])
GetFileName ([out, string])
GetLevel ([out] unsigned)
GetOutputs ([out] DWORD)
OpenFile ([in, string])
SetOutputs ([in] DWORD new, [out] DWORD old)
SetLevel ([in] unsigned NewLevel, [out] unsigned OldLevel)

Objects

EventLog

SetPosition ([in] unsigned flags), ([in] unsigned long recNum) **+EventLog SetPosition flag recNum** **+EventLog SetPosition 2 0** **=00000000** Sets the current position in the event log. The recNumber is the record number to position at, the flags bits determines the seek mode:
GetRecord ([out] TCPEventLogRecord pRec), **([out, string] Source), ([out, string] EventText)** **+EventLog GetRecord flag recNum** **+EventLog GetRecord** **=00000000** Reads one record from the event log. Returns the record header information, the name of the source that generated the event and the text of the event.
RegisterNotifyTarget ( ) **+EventLog RegisterNotifyTarget** **=00000000** Registers the client to receive event notifications from the server. Refer to **EventLogNotify** Object in the next table.
UnregisterNotifyTarget ( ) **+EventLog UnregisterNotifyTarget** **=00000000** Unregisters the client to receive notifications.

EventLogNotify

NewEvent ([in] TCPEventLogRecord rec), [in, string] SourceName), [in, string] EventText) **+EventLogNotify NewEvent** **=00000000 { 1074003997 0 20 1047388718 } "GalSysMon" "Chassis Fan-3 speed back to normal. Current speed is 2463 RPM"** The server calls this method on the client when a new event is generated. The arguments are the record header, the name of the source that generated the event and the text of the event message.

GalWinSys

ApplyDefaults ([in] WinId_t) **+GalWinSys ApplyDefaults { 123 }** **=00000000** Brightness, contract, hue, saturation and cropping are reset for specified window
GetCrop ([in] WinId_t, [out] CPRect) **+GalWinSys GetCrop { 123 }** **=00000000 { 27 0 100 0 0 100)** Returns nX nY nW nH
GetImgBalance ([in] WinId_t, [out] ImgBalance) **+GalWinSys GetImageBalance { 123 } {Set Brightness Contrast Gamma Hue Saturation }** **=00000000 { 123 } { 27 0 100 0 0 100)** returns {Set Brightness Contrast Gamma Hue Saturation }, Uses Set to determine valid data values
SetImgBalance ([in] WinId_t, [out] ImgBalance) **+GalWinSys SetImgBalance { 123 } { 27 0 100 0 0 100)** **=00000000 { 123 } { 27 0 100 0 0 100)** Sets {Set Brightness Contrast Gamma Hue Saturation }, Uses Set to determine valid data values
GetInputSize ([in] WinId_t, [out] CPSize) **+GalWinSys GetInputSize { 123 }** **=00000000 { 123 } 640 480** (Returns cx cy)
GetKind ([out] SubSystemKind_t) **+GalWinSys GetKind { }** **=00000000 { 123 } { 2 }** (returns the SubSystemKind this Object can operate on) When used with LiveVideoSys or RGBSys returns SubSystemKind for respective Object.
IsOfKind ([in] WinId_t) **+GalWinSys IsOfKind { 123 }** **=00000000 { 123 } 1** (returns either 0 or 1 (1= success but false) for question 'Can this Object operate on this window?'). When used with LiveVideoSys or RGBSys returns 0 or 1 (true or false) for type of window queried.

NewWindow ([out] WinId_t) **+GalWinSys NewWindow** **=00000000 { 00010123 }** (returns new window ID)
NewWindowWithId ([in] WinId_t) **+GalWinSys NewWindowWithId { 123 }** **=00000000 { 00000123 }** Creates new window with specified ID number.
QueryAllWindows ([out] TWindowState_array_t) **+GalWinSys QueryAllWindows** **=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } }** **{ { 102 } 3 1 7183 152 21 320 240 { 102 } }** returns { nCount TwindowState pData[ ] } (Id Kind nState nStateChange x y w h ZAfter
SetCrop ([in] WinId_t, [in] CPRect) **+GalWinSys SetCrop { 123 } {nX nY nW nH }** Crop works by relation to the normal image size. nX and nY specify the pixels removed from the top and bottom. nW and nH specify the size of the cropped image. A 640x480 window set at 10 10 600 400 will have 10 pixels removed from the top, 10 pixels removed from the left, 30 and 70 removed from the right and bottom. ImageWidth - nX - (right pixels) = nW or 640 – 10 – 30 = 600.
GetCrop ([in] WinId_t wid, [out] struct CPRect * pRect) **+GalWinSys GetCrop { 123 }** **=00000000 { 27 0 100 0 0 100)** (Returns nX nY nW nH)
SetOrigin ([in] WinId_t, [in] long x, [in] long y) **+GalWinSys SetOrigin { 123 } x y** **=00000000** Sets origin of cropped image to effect panning
SetImgBalance ([in] WinId_t, [in, out] ImgBalance) **+GalWinSys SetImageBalance { 123 }** **{Set Brightness Contrast Gamma Hue Saturation }** **=00000000** Sets {Set Brightness Contrast Gamma Hue Saturation }, Uses **Set** to determine valid data values
Start ([in] WinId_t) **+GalWinSys Start { 123 }** (Starts capturing)

Stop ([in] WinId_t) **+GalWinSys Stop { 123 }** (Stops capturing)
Freeze ([in] WinId_t) **+GalWiniSys Freeze { 123 }** (Freezes the frame in the specified window)
QueryAllInputsCS ([out, string]) **+GalWinSys QueryAllInputsCS** **=00000000 “input1” “input2” …** Returns a string input names.
SelectInput ([in] WinId_t wid, InputName) **+GalWinSys SelectInput { 123 } “InputName”** **=00000000** Selects an input into a window.
GetInput ([in] WinId_t wid, [out, inputName]) **+GalWinSys GetInput { 123 }** **=00000000 { 123) “input1”** Returns the currently selected input for a window.

LiveVideoSys

GetChannel ([in] WinId_t, [out] short) **+LiveVideoSys GetChannel { 123 }** **=00000000 { 123 } 3** Returns current Channel
GetVideoSource ([in] WinId_t, [out] CPLiveVideoSource) **+LiveVideoSys GetVideoSource { 123 }** **=00000000 { 123 } { 3 2 0 }** Returns Set Format Type
SetChannel ([in] WinId_t, [in] short); **+LiveVideoSys SetChannel { 123 } 1** Sets window 123 to channel 1
SetVideoSource ([in] WinId_t, [in] CPLiveVideoSource) **+LiveVideoSys SetVideoSource { 123 } { Set Format Type }** **+LiveVideoSys SetVideoSource { 123 } { 3 2 0 }** Set Format and Type (Set=3) to PAL (2) Composite (0)

Notify

ScreenConfigChanged ([in] CPScreenConfig) **+Notify ScreenConfigChanged { TotalWidth TotalHeight SingleScreenWidth SingleScreenHeight }** Refer to **Notify** listing**.**
WindowsState ([in] TWindowState_array_t) **+Notify WindowsState { nCount TWindowState pData[ ] } (Id Kind nState nStateChange x y w h ZAfter)** Refer to **Notify** listing**.**

PictureViewerSys

NewWindow ([out] WinId_t pwid) **+PictureViewerSys NewWindow** **=00010123** Returns server assigned ID
NewWindowWithId ([in] WinId_t wid) **+PictureViewerSys NewWindowWithId { 123}** **=00000123** Returns user assigned ID
ShowPicture ([in] WinId_t, [in,string]) **+PictureViewerSys ShowPicture { 123 } “C:/MyPictures/APic.bmp”** **=00000000** Displays an image with the specified file path.
GetFileName ([in] WinId_t, [out,string]) **+ PictureViewerSys GetFileName { 123 }** **=00000000 "123 C:/MyPictures/APic.bmp"** Return the file path of the current image.
SetTextMode ([in] WinId_t wid, [in] int textmode) **+PictureViewerSys SetTextMode { WinId } 1** **=00000000** Sets the PictureViewer text mode (where 0 = text mode off; 1 = text mode on; 2 = text mode on with text scroll). The window ID must be specified.
GetTextMode ([in] WinId_t wid, [out] int *textmode) **+PictureViewerSys GetTextMode { WinId }** **=00000000 1** Returns the PictureViewer window’s text mode status using WinId.
SetText ([in] WinId_t wid, [in,string] wchar_t * text) **+PictureViewerSys SetText { WinId } “Front”** **=00000000** Sets the entered text on the PictureViewer window. The window ID must be specified.
GetText ([in] WinId_t wid, [out,string] wchar_t ** text) **+PictureViewerSys GetText { WinId }** **=00000000 “Front”** Returns the entered text on the PictureViewer window (using WinId).

Objects

RGBSys

GetAutoDetectTiming ([in] WinId_t wid, [out] Boolean * bEnabled **+RGBSys GetAutoDetectTiming { 123 }** **=00000000 1** (Returns auto detect setting (1/0) of specified window
GetChannel ([in] WinId_t, [out] short) **+RGBSys GetChannel { 123 }** **=00000000 1** (returns channel number)
GetChannelRange ([out] short * FirstCh, [out] short * LastCh) **+RGBSys GetChannelRange** **=000000000 1 32** Returns the first and the last channel numbers for the sub-system
GetComponent ([in] WinId_t wid, [out] Boolean * bEnabled) **+RGBSys GetComponent { WinId }** **=000000000 { 1 }** Returns the classification of 1 for SetComponent and 0 for RGB.
GetDualLink ([in] WinId_t wid, [out] Boolean * bEnabled) **+RGBSys GetDualLink { WinId }** **=00000000 { 1 }** Returns the classification of 1 for Dual Link and 0 for Single Link.
GetRCServer ([in] WinId_t wid, [out, string] wchar_t ** serverName) **+RGBSys GetRCServer { WinId }** **=00000000 { NewRCServer }** Returns the name or IP address of the RCServer
GetTiming ([in] WinId_t, [out] CPRGBTiming) **+RGBSys GetTiming { 123 }** **=00000000 { 1 640 480 144 480 525 35 3 60 0 0 0 }** Returns { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg
DetectTiming ([in] WinId_t, [out] CPRGBTiming) **+RGBSys DetectTiming { 123 }** **=00000000 { 1 640 480 144 480 525 35 3 60 0 0 0 }** (Forces resample of input signal - Returns { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }

SetAutoDetectTiming ([in] WinId_t wid, [in] Boolean bEnable) **+RGBSys SetAutoDetectTiming { 123 } 1** **=00000000** Enable/disable (1/0) auto-detection of RGB timing
SetChannel ([in] WinId_t, [in] short) **+RGBSys SetChannel { 123 } 1** **=00000000 { 123 } 1** (sets window 123 to RGB channel 1)
SetComponent ([in] WinId_t wid, [in] Boolean bEnabled) **+RGBSys SetComponent { WinId } 1** **=000000000** Sets the classification of 1 for SetComponent and 0 for RGB
SetDualLink ([in] WinId_t wid, [in] Boolean bEnabled) **+RGBSys SetDualLink { WinId } 1** **=00000000** Sets the classification of 1 for Dual Link and 0 for Single Link.
SetRCServer ([in] WinId_t wid, [in, string] wchar_t * serverName) **+RGBSys SetRCServer { WinId } NewRCServer** **=00000000** Sets the name or IP address of the RCServer
SetTiming ([in] WinId_t, [in] CPRGBTiming) **+RGBSys SetTiming { 123 } { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }** **+RGBSys SetTiming { 123 } { 1 640 480 144 480 525 35 3 60 0 0 0 }** **=00000000** Sets the RGB Timing parameters

Objects

ScreenUtil

ShowPattern ([in,string])
+ScreenUtil ShowPattern "colorpat"

"stdpat" - display the Standard test pattern
"screenid"- display the screen ID
"phasepat" - display the Phase pattern
"bitmap" - display a bitmap image
"colorpat" - display the Color Bar pattern
"colorpat" - display the Grid pattern
"" - the null function terminates the screen test

Display a screen test pattern with the specified pattern name.

SetPatternProp ([in,string] prop_name, [out] struct ScreenTestPattern *)
+ScreenUtil SetPatternProp "setpatternprop" { 0x7ff 1 1 1 1 1 1 0 0 0 1 10 }

Set properties to the current test pattern with the specified values.

+ScreenUtil SetPatternProp "setpatternprop" { Set SingleScreen BarType GridCircle FgColorRed FgColorGreen FgColorBlue BgColorRed BgColorGreen BgColorBlue SmoothGradient LineSpacing }

+ScreenUtility SetPatternProp "SetPatternProp" { Set [options] } { 0x008 1 0 1 0 1 0 0 0 0 0 0 0 }

Set = 1, single screen = on (0) --
Bar type = 1 -- vertical (2), Foreground = green (16) --
Set = 19 = 0x000013

GetPatternProp ([in,string] prop_name, [out] struct ScreenTestPattern *)
+ ScreenUtil GetPatternProp "getpatternprop"
=00000000 { 0x7ff 1 1 1 1 1 1 0 0 0 1 10 }

Returns the properties of the current test pattern.

SysMonEx

QueryAllValues ([out] TCPSysMonValue_array_t) **+SysMonEx QueryAllValues** Returns all monitored values. This object uses the **TCPSysMonValue_array_t** data structure.
QueryValues ([in] CPSysMonValueIndex_array_t) ([out] TCPSysMonValue_array_t) **+SysMonEx QueryValues { 2 2 17 }** **=00000000 { 2 { 2 29 0 0 } { 17 11977 0 11977 } }** Returns specific values, listed by codes specified. QueryValues { n codea codeb }
QueryECCInfo [out] TCPSysMonECCInfo) **TCPSysMonECCInfo** **+SysMonEx QueryECCInfo** **=00000000 { 0 0 0 0 }** Returns the ECC memory status. This method uses the data structure
RegisterNotifyTarget ( ) **+SysMonEx RegisterNotifyTarget** Registers the client to receive notifications when the system status changes. Refer to Object, **“SysMonNotifyEx” on page 99**. Use this method to turn on the notify messages.
UnregisterNotifyTarget ( ) **+SysMonEx UnregisterNotifyTarget** Unregisters the client to receive system-monitoring notifications. Use this method to turn off the notify messages.

Objects

SysMonNotifyEx

ECCError [in] TCPSysMonValue_array_t) **+SysMonNotifyEx ECCError** **:SysMonNotifyEx ECCError { 0 1 0 0 }** Called when an ECC error occurs. This method returns the System EDD Data.
ValuesChanged ([in] TCPSysMonECCInfo) **+SysMonNotifyEx ValuesChanged** **:SysMonNotifyEx ValuesChanged { 3 { 5 1696 1696 0 } { 128 -5178 -5178 0 }** **{ 48 12464 12464 0 } }** The server calls this method on the client to supply information about values that changed. The supplied parameter is an array (**TCPSysMonValue_array_t**) of the values that have changed since the last notification.

UserMan

AddUser ([in, string] UserName, [in, string] AuthToken, [in] short Level) **+UserMan AddUser “username” “password” level** **+UserMan AddUser “sam” “spike123” 2** adds a user of user name and password with user level. Refer to **User Levels** below.
ChangePassword ([in, string] OldToken, [in, string] NewToken) **+UserMan ChangePassword “oldpassword” “newpassword”** must be logged in as user to change password
DeleteUser ([in, string]) **+UserMan DeleteUser “username”** deletes user
EnumUsersCS ([out, string]) **+UserMan EnumUsersCS** **=00000000 "admin,localuser,test"** (lists all users)
GetUserInfo ([in, string] Name, [out, string] AuthToken, [out] short Level) **+USerMan GetUserInfo “username”** **=00000000 "test" 1** returns password and level for username

SetUserInfo ([in, string] Name, [in, string] AuthToken, [in] short Level)
+UserMan SetUserInfo "username" "password" level
sets password and level for username

User Levels
Level 0 – administrator - full access
Level 1 – user cannot administer users and cannot shut down the server
Level –1 – is a special case; it means that the account is disabled.

VidStreamSys

NewWindow ([out] WinId_t pwid)
+VidStreamSys NewWindow
=00010123
Returns server assigned ID

NewWindowWithId ([in] WinId_t wid)
+VidStreamSys NewWindowWithId
=00000123
Returns user assigned ID

SetSource ([in] WinId_t wid, [in,string] wchar_t * srcName)
+VidStreamSys SetSource { 123 } "SourceObjectName"
=00000000
Sets the streaming video source object for the specified window.

GetSource ([in] long wid, [out,string] wchar_t ** srcName)
+VidStreamSys GetSource { 123 }
=00000000 "SourceObjectName"
Returns the object name of the current video source

GetDecoder ([in] WinId_t wid, [out] short * decoder)
+VidStreamSys GetDecoder { 123 }
=00000000
Returns decoder channel number for the specified window.

GetNumDecoders ([out] short * decoders)
+VidStreamSys GetNumDecoders
=00000000 8
Returns number of decoders available.

Start ([in] WinId_t wid) **+VidStreamSys Start { 123 }**
Stop ([in] WinId_t wid) **+VidStreamSys Stop { 123 }**
GetImgBalance ([in] WinId_t wid, [out] struct ImgBalance * pBal) **+VidStreamSys GetImgBalance** **=00000000** Returns the image balance parameters for the specified VideoStream window.
SetImgBalance ([in] WinId_t wid, [in,out] struct ImgBalance * pBal) **+VidStreamSys SetImgBalance { 123 } { Set Brightness Contrast Gamma Hue Saturation }** **+VidStreamSys SetImgBalance { 123 } { 19 0 100 0 0 100 }** **=00000000** Sets the image balance parameters for the specified VideoStream window.
SetCrop ([in] WinId_t wid, [in] struct CPRect * pRect) **+VidStreamSys SetCropPrm { 123 } { X Y W H }** **+VidStreamSys SetCropPrm { 123 } { 10 10 600 400 }** **=00000000** **SetCrop** sets the cropping parameters to a VideoStream window. Cropping works by relation to the normal image size. X and Y specify the pixels removed from the left and the top. W and H specify the size of the cropped image. A 640x480 window set at 10 10 600 400 will have 10 pixels remove from the top, 10 pixels removed from the left, 30 pixels from the right, and 70 removed from the right. ImageWidth – X – (right pixels) = W or 640 – 10 – 30
SetOrigin ([in] WinId_t wid, [in] long x, [in] long y) **+VidStreamSys SetOrigin { 123 } x y** **=00000000** Sets origin of cropped image to effect panning
GetCrop ([in] WinId_t wid, [out] struct CPRect * pRect) **+VidStreamSys GetCrop { 123 }** **=00000000 { 27 0 100 0 0 100)** Returns nX nY nW nH
GetInputSize ([in] WinId_t wid, [out] struct CPSize * pSize) **+VidStreamSys GetInputSize { 123 }** **=00000000 { 123 } 640 480** (Returns cx cy)
GetSVSVersion ([in] WinId_t wid, [out] short * svsver) **+VidStreamSys GetSVSVersion { 123 }** **=00000000 "1.9.4.348"** Returns current installed version of the SVS Server software.

IPStreamSys

NewWindow ([out] WinId_t pwid) **+IPStreamSys NewWindow** **=00010123** Returns server assigned ID
NewWindowWithId ([in] WinId_t wid) **+IPStreamSys NewWindowWithId** **=00000123** Returns user assigned ID
SetSource ([in] WinId_t wid, [in,string] wchar_t * srcName) **+IPStreamSys SetSource { 123 } "SourceObjectName"** **=00000000** Sets the streaming video source object for the specified window.
GetSource ([in] long wid, [out,string] wchar_t ** srcName) **+IPStreamSys GetSource { 123 }** **=00000000 "SourceObjectName"** Returns the object name of the current video source
GetDecoder ([in] WinId_t wid, [out] short * decoder) **+IPStreamSys GetDecoder { 123 }** **=00000000 { 3 { "Decoder Channel" "1" } { "Link Status" "1" } { "IP Address" "10.4.1.63" } }** Returns decoder Channel Number, Link Status, and IP Address for the specified window **Note** Ignore the "80040503 (Invalid RMC method name)" Error Message that appears when using this command.
GetNumDecoders ([out] short * decoders) **+IPStreamSys GetNumDecoders** **=00000000 8** Returns number of decoders available.
GetIPDVersion ([in] WinId_t wid, [out] short * svsver) **+IPStreamSys GetIPDVersion** **=00000000 6** Returns version of decoder firmware.

Objects

Window

GetState ([in] WinId_t, [out] TWindowState) **+Window GetState { 123 }** **=00000000 { { 123 } 2 1 7183 100 100 320 200 { 1 } }** Set indicates valid data fields. 7183 indicates that all are valid. Refer to **"Understanding and Using Flags" on page 122** and **Table 86 on page 150**. Returns (Id Kind nState Set x y w h ZAfter)
GetTitle ([in] WinId_t, [out, string] title) **+Window GetTitle { 102 }** **=00000000 "102 RGB"** Returns window title
GrabImage ([in] WinId_t, [out,string] wchar_t **) **+Window GrabImage { 123 }** **=00000000 { 123 }** **"C:\ProgramData\ControlPoint\ServerDataFiles\Images\XX_XX.bmp"** Returns a file path on the server for the image file. You can use the file path to query the image data from the server and save it to the local hard drive.
SetState ([in, out] TWindowState); **+Window SetState { { WinID_t } Kind nState Set x y w h { ZAfter } }** **+Window SetState { { 123 } 2 1 7183 100 100 320 200 { 1 } }** Refer to **page 111**.
SetTitle ([in] WinId_t, [in, string] title); **+Window SetTitle { 123 } "titlename"** sets title "titlename" to this window.
+Window SetTitleInfo { wid } { Set TextColor BarPos MinBarHeight HorizJust VertJust }+Window SetTitleInfo { 123 } { 31 00FF00 0 20 1 1 } Green only - Set = 31 TextColor BarPos MinBarHeight HorizJust VertJust all valid. Text color = green, on top, 20 pixels high, vertical and horizontal center. **+Window SetTitleInfo { 123 } { 1 0 0 0 0 0 }** Sets the text color to black, and the other parameters are ignored **+Window SetTitleInfo { 123 } { 10 0 1 0 1 0 }** Sets the title bar at the bottom of the window, the text is horizontal center justified, and the other parameters are ignore
GetTitleInfo returns the title information from a window. **+Window GetTitleInfo { 123 }** **=00000000 { 123 } { 10 0 1 0 1 0 }**

Object Method { WinId } { Set [options] }

+Window SetFrameInfo { WinId }
{ Set FrameWidth FrameColor ShowTitle Show UserData }

Frame width (FW) in pixels – 0 to 100
Frame color (FC) = RGB
Red = 1 to 255
Blue = 1 to 255
Green = 1 to 255
White is all color on – (256) x (256) x (256) = 0xFFFFFF or 16,777,215
Black is all color off – 000 = 0x000000.
Example – green only – 00 FF 00 = 65,280 = 0x00FF00
Show title = 1 = display title – 0 = title off

+Window SetFrameInfo { 123 } { 7 10 00FF00 1 0 }
Green only - Set = 7 width, color and title all valid.

GetFrameInfo returns the frame information from a window.
+Window GetFrameInfo { 123 }
=00000000 { 123 } { 7 10 16777219 1 0 }

+Window SetTitleFontInfo { wid }
{ Set TextSize Flags FontName }

+Window SetTitleFontInfo { 123 } { 5 14 0 "Arial" } Sets the text size = 14, font name = "Arial", and ignores the flags

---

GetTitleFontInfo returns the title font information from a window.
+Window GetTitleFontInfo { 123 }
=00000000 { 123 } { 5 14 0 "Arial" }

Objects

WinServer

DeleteLayout ([in, string]) **+WinServer DeleteLayout “layoutname”** Deletes named layout
DeleteWindow ([in] WinId_t) **+WinServer DeleteWindow { 123 }** Deletes specified window
FindWindow ([in,string] window_descriptor, [out] WinId_t) **+WinServer FindWindow "descriptor"** **=00000000 { 0010123 }** Searches for a window on the screen that matches the specified window descriptor. Returns the window ID of the window. Returns a zero ID if no window is found.
GetScreenConfig ([out] CPScreenConfig) **+WinServer GetScreenConfig** **=00000000 { 123 } { 1024 768 1024 768 }** Returns { TotalWidth TotalHeight SingleScreenWidth SingleScreenHeight }
GetServerInfo ([out] CPServerInfo) **+WinServer GetServerInfo** **=00000000 { PlatformCode ModelVersion ModelRevision OEMCode SerialNumber } “Version”** **=00000000 { 16 0 0 0 1057 } "1.9.4.348"** Returns{ dwVersionMS dwVersionLS dwFileTimeMS dwFileTimeLS }
GetAppWinInfo ([in] WinId_t winid, [out,string] window_descriptor, [out,string] cmdLine, [out,string] workDir) **+WinServer GetAppWinInfo { 0010123 }** **=00000000 "ELARA|VNCVIEWER|VNCVIEWER" "\"D:\\Program Files\\RealVNC\\vncviewer.exe\" /viewonly /config elara-5900.vnc" "D:\\Program Files\\RealVNC\\"** returns application window info for object with the specified name
InvokeAppWindow ([in,string] appWinName, [out] WinId_t) **+WinServer GetAppWinInfo { 123 } “ObjectName”** **=00000000 { 123 }** Invokes an application window object with the specified name. Returns the window ID of the application window.

QueryAllLayoutsCS ([out, string]) **+WinServer QueryAllLayoutsCS** (Returns all layout names) **=00000000 “Layout1” “Layout1” …** Returns list of layout names
QueryAllWindows ([out] TWindowState_array_t) **+WinServer QueryAllWindows** **=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } }** **{ { 102 } 3 1 7183 152 21 320 240 { 102 } }** Returns TwindowState_array_t
QueryLastSetLayout ([out, string]) **+WinServer QueryLastSetLayout** **=00000000 “Friday’s Layout”** The name of the last layout loaded will be retrieved by this command. However, use of this command gets complicated with many users. In a scenario where different users use the same wall with different permissions, a user trying to retrieve the name of a layout that was loaded 10 minutes ago may not retrieve it with this command if another user loaded a different layout in the last few minutes. Only the name of the last layout that was loaded will be retrieved by this command, regardless of who loaded the layouts and who issued the command.
QueryWindows ([in] WinId_t_array_t, [out] TWindowState_array_t) **+WinServer QueryWindows { 3 { 10001 } { 10002 } { 123 } }** **=00000000 { 102 } 3 1 7183 152 21 320 240 { 102 }** Returns TwindowState_array_t for each specified window ID.
Quit ( ); **+WinServer Quit { }** (quits (exits) ControlPoint Server)
RegisterNotifyTarget () **+WinServer RegisterNotifyTarget { }** (turns on notify messages)
SaveLayout ([in, string], [in] WinId_t_array_t) **+WinServer SaveLayout “layoutname” { 0 }** Saves named layout
SetLayout ([in, string]) **+WinServer SetLayout “layoutname”** Applies named layout
UnregisterNotifyTarget () **+WinServer UnregisterNotifyTarget { }** Turns off notify messages

Appendix C—Alphabetic Command Listing of Methods

C. Alphabetic Command Listing of Methods

AddUser ([in, string] UserName, [in, string] AuthToken, [in] short Level) **+UserMan AddUser "username" "password" level** **+UserMan AddUser "sam" "spike123" 2** adds a user of user name and password with user level, Refer to **User Levels** below.
ApplyDefaults ([in] WinId_t) **+GalWinSys ApplyDefaults { 123 }** **=00000000** Brightness, contract, hue, saturation and cropping are reset for specified window
ChangePassword ([in, string] OldToken, [in, string] NewToken) **+UserMan ChangePassword "oldpassword" "newpassword"** must be logged in as user to change password
CloseFile ()
DeleteLayout ([in, string]) **+WinServer DeleteLayout "layoutname"** Deletes named layout
DeleteUser ([in, string]) **+UserMan DeleteUser "username"** deletes user
DeleteWindow ([in] WinId_t) **+WinServer DeleteWindow { 123 }** Deletes specified window

DetectTiming ([in] WinId_t, [out] CPRGBTiming)
+RGBSys DetectTiming { 123 }
=00000000 { 1 640 480 144 480 525 35 3 60 0 0 0 }
(Forces resample of input signal - Returns { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }

Exec ([in, string] CmdLine)
+AppCtrl Exec "program path"
Executes the specified command-line on the server. Note that all paths are server-relative. Program path is the path on the server to the program you wish to run.

ECCError [in] TCPSysMonValue_array_t)
+SysMonNotifyEx ECCError
:SysMonNotifyEx ECCError { 0 1 0 0 }
Called when an ECC error occurs. This method returns the System EDD Data.

EnumUsersCS ([out, string])
+UserMan EnumUsersCS
=00000000 "admin,localuser,test"
(lists all users)

FindWindow ([in,string] window_descriptor, [out] WinId_t)
+WinServer FindWindow "descriptor"
=00000000 { 123 }
Searches for a window on the screen that matches the specified window descriptor. Returns the window ID of the window. Returns a zero ID if no window is found.

FlushToFile ([in, string])

Freeze ([in] WinId_t)
+GalWiniSys { 123 }
Freezes the frame in the specified window

+GalWinSys
+VidStreamSys

GetAppWinInfo ([in] WinId_t winid, [out,string] window_descriptor, [out,string] cmdLine, [out,string] workDir)
+WinServer GetAppWinInfo { 0010123 }
=00000000 "ELARA|VNCVIEWER|VNCVIEWER" "\"D:\\Program Files\\RealVNC\\vncviewer.exe\" /viewonly /config elara-5900.vnc" "D:\\Program Files\\RealVNC\\"
return application window info for object with the specified name

GetAutoDetectTiming ([in] WinId_t wid, [out] Boolean * bEnabled
+RGBSys GetAutoDetectTiming { 123 }
=00000000 1
Returns auto detect setting (1/0) of specified window

Methods

+RGBSys GetChannelRange **=000000000 1 32** Returns the first and the last channel numbers for the sub-system
GetConnection ([in] WinId_t wid, [out,string] connection) **+CPShareSys GetConnection { 1 }** **=00000000 “Elara”** (Queries CPShare window id = 1 for its currently selected connection name)
GetCredentials ([in] WinId_t wid, [out,string] server, [out, string] password) **+CPShareSys GetCredentials { 1 }** **=00000000 “Elara” “testpassword”** returns the server host address and the password used to authenticate the CPShare connection.
GetChannel ([in] WinId_t, [out] short) **+LiveVideoSys GetChannel { 123 }** **=00000000 { 123 } 3** Returns current LiveVideo Channel of specified window **+RGBSys GetChannel { 123 }** **=00000000 1** Returns current RGB channel of specified window
GetComponent ([in] WinId_t wid, [out] Boolean * bEnabled) **+RGBSys GetComponent { WinId }** **=000000000 { 1 }** Returns the classification of 1 for SetComponent and 0 for RGB.
GetCrop ([in] WinId_t, [out] CPRect) **+GalWinSys GetCrop { 123 }** **=00000000 { 27 0 100 0 0 100)** Returns nX nY nW nH **+GalwinSys** **+VideoStreamSys**
GetDecoder ([in] WinId_t wid, [out] short * decoder) **VidStreamSys GetDecoder** **=00000000 { 123 }** Returns de coder channel number
GetDualLink ([in] WinId_t wid, [out] Boolean * bEnabled) **+RGBSys GetDualLink { WinId }** **=00000000 { 1 }** Returns the classification of 1 for Dual Link and 0 for Single Link.

C—Alphabetic Command Listing of Methods

GetFileName ([in] WinId_t, [out,string]) **+PictureViewerSys GetFileName { 123 }** **=00000000 "123 C:/MyPictures/APic.bmp"** Return the file path of the current image.
GetImgBalance ([in] WinId_t, [out] ImgBalance) **+GalWinSys GetImageBalance { 123 } {Set Brightness Contrast Gamma Hue Saturation }** **=00000000 { 123 } { 27 0 100 0 0 100)** returns {Set Brightness Contrast Gamma Hue Saturation }, Uses Set to determine valid data values **+GalWinSys** **+VidStreamSys**
GetInput ([in] WinId_t wid, [out, inputName]) **+GalWinSys GetInput { 123 }** **=00000000 { 123) "input1"** Returns the currently selected input for a window.
GetInputSize ([in] WinId_t, [out] CPSize) **+GalWinSys GetInputSize { 123 }** **=00000000 { 123 } 640 480** (Returns cx cy) **+CPShare** **+CPWebSys** **+GalWinSys** **+PictureViewerSys** **+VidStreamSys**
GetKind ([out] SubSystemKind_t) **+GalWinSys GetKind { }** **=00000000 { 123 } { 2 }** (returns the SubSystemKind this Object can operate on) When used with LiveVideoSys or RGBSys returns SubSystemKind for respective Object.
GetLevel ([out] unsigned)
GetNumDecoders ([out] short * decoders) **VidStreamSys GetNumDecoders** **=00000000 8** Returns number of decoders available.
GetOutputs ([out] DWORD)

Methods

GetRCServer ([in] WinId_t wid, [out, string] wchar_t ** serverName) **+RGBSys GetRCServer { WinId }** **=00000000 { NewRCServer }** Returns the name or IP address of the RCServer
GetRefreshTime ([in] WinId_t, [out] long) **+RGBSys GetRefreshTime { 123 }** **=00000000 { 123 } 1** Returns current update rate)
GetScreenConfig ([out] CPScreenConfig) **+WinServer GetScreenConfig** **=00000000 { 123 } { 1024 768 1024 768 }** Returns { TotalWidth TotalHeight SingleScreenWidth SingleScreenHeight }
GetServerInfo ([out] CPPlatformInfo, [out,string] versionInfo) **+ConfigSys GetServerInfo** **=00000000 { PlatformCode ModelVersion ModelRevision OEMCode SerialNumber } “ControlPoint Version”** **=00000000 { 19 0 1 0 4005 } "2.9.6414.296"** Returns information about the hardware platform and the version of the ControlPoint software installed on the server.
GetSource ([in] long wid, [out,string] wchar_t ** srcName) **+VidStreamSys GetSource** **=00000000 { 123 } “SourceObjectName”** Returns the object name of the current video source
GetState ([in] WinId_t, [out] TWindowState) **+Window GetState { 123 }** **=00000000 { { 123 } 2 1 7183 100 100 320 200 { 1 } }** Set indicates valid data fields. 7183 indicates that all are valid. Refer to **“Understanding and Using Flags” on page 122** and **Table 86 on page 150**. Returns (Id Kind nState Set x y w h ZAfter)
GetSVSVersion ([in] WinId_t wid, [out] short * svsver) **+VidStreamSys GetSVSVersion { 123 }** **=00000000 "2.9.4.348"** Returns current installed version of the SVS Server software.
GetText ([in] WinId_t wid, [out,string] wchar_t ** text) **+PictureViewerSys GetText { WinId }** **=00000000 “Front”** Returns the entered text on the PictureViewer window (using WinId).

GetTextMode ([in] WinId_t wid, [out] int *textmode) **+PictureViewerSys GetTextMode { WinId }** **=00000000 1** Returns the PictureViewer window's text mode status using WinId.
GetTitle ([in] WinId_t, [out, string] title) **+Window GetTitle { 102 }** **=00000000 "102 RGB"** Returns window title
GetTitleInfo returns the title information from a window. **+Window GetTitleInfo { 123 }** **=00000000 { 123 } { 10 0 1 0 1 0 }**
GetTitleFontInfo returns the title font information from a window. **+Window GetTitleFontInfo { 123 }** **=00000000 { 123 } { 5 14 0 "Arial" }**
GetFrameInfo returns the frame information from a window. **+Window GetFrameInfo { 123 }** **=00000000 { 123 } { 7 10 16777219 1 0 }**
GetTiming ([in] WinId_t, [out] CPRGBTiming) **+RGBSys GetTiming { 123 }** **=00000000 { 1 640 480 144 480 525 35 3 60 0 0 0 }** Returns { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg
GetURL ([in] WinId_t, [out,string]) **+CPWebSys GetURL { 123 }** **=00000000 "123 http://www.jupiter.com"** Return the URL address of the current web**.**
GetUserInfo ([in, string] Name, [out, string] AuthToken, [out] short Level) **+UserMan GetUserInfo "username"** **=00000000 "test" 1** returns password and level for username
GrabImage ([in] WinId_t, [out,string] wchar_t **) **+Window GrabImage { 123 }** **=00000000 { 123 }** **"C:\ProgramData\ControlPoint\ServerDataFiles\Images\XX_XX.bmp"** Returns a file path on the server for the image file. You can use the file path to query the image data from the server and save it to the local hard drive.

Methods

InvokeAppWindow ([in,string] appWinName, [out] WinId_t)
+WinServer GetAppWinInfo { 123 } “ObjectName”
=00000000 { 123 }
Invokes an application window object with the specified name. Returns the window ID of the application window.

IsOfKind ([in] WinId_t)
+GalWinSys IsOfKind { 123 }
=00000000 { 123 } 1
(returns either 0 or 1 (1= success but false) for question ‘Can this Object operate on this window?’). When used with LiveVideoSys or RGBSys returns 0 or 1 (true or false) for type of window queried.

ListCfgGroup ([in] group_code, [out,string] objNames[]
+ConfigSys ListCfgGroup 1
=00000000 { 3 "1 RGB" "2 Rgb" "Video" }
Returns the object names in a configuration group. The configuration group codes are:
1 – Named Inputs
2 – Application Windows
3 – Layouts
4 - CPShare
5 - HotKeys
6 - Schedule
7 - Video Stream Source
8 - Fonts
9 - Images
11 - Batch (The number 10 is not used)
The result is an array of strings.

NewEvent ([in] TCPEventLogRecord rec), [in, string] SourceName), [in, string] EventText)
+EventLogNotify NewEvent
The server calls this method on the client when a new event is generated. The arguments are the record header, the name of the source that generated the event and the text of the event message.

=00000000 { 1074003997 0 20 1047388718 } "GalSysMon" "Chassis Fan-3 speed back to normal. Current speed is 2463 RPM"

C—Alphabetic Command Listing of Methods

NewWindow ([out] WinId_t pwid)
+CPShareSys NewWindow { 123 })
=00010123
Returns server assigned ID

+CPShare
+CPWebSys
+GalWinSys
+PictureViewerSys
+VidStreamSys

NewWindowWithId ([in] WinId_t wid)
+CPShareSys NewWindowWithId { 123 }
=00000123
Returns user assigned ID

+CPShare
+CPWebSys
+GalWinSys
+PictureViewerSys
+VidStreamSys

OpenFile ([in, string])

QueryAllWindows ([out] TWindowState_array_t)
+GalWinSys QueryAllWindows
=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } }
{ { 102 } 3 1 7183 152 21 320 240 { 102 } }
Returns { nCount TwindowState pData[ ] } (Id Kind nState nStateChange x y w h ZAfter

QueryAllWindows ([out] TWindowState_array_t)
+WinServer QueryAllWindows
=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } }
{ { 102 } 3 1 7183 152 21 320 240 { 102 } }
Returns TwindowState_array_t

QueryAllInputsCS ([out, string])
+GalWinSys QueryAllInputsCS
=00000000 “input1” “input2” …
Returns a string list of input names.

QueryAllLayoutsCS ([out, string])
+WinServer QueryAllLayoutsCS (Returns all layout names)
=00000000 “Layout1” “Layout1” …
Returns list of layout names

Methods

QueryAllValues ([out] TCPSysMonValue_array_t) **+SysMonEx QueryAllValues** Returns all monitored values. This object uses the **TCPSysMonValue_array_t** data structure.
QueryAllWindows ([out] TWindowState_array_t) **+GalWinSys QueryAllWindows** **=00000000 { 2 { { 101 } 2 9 7183 154 292 320 240 { 10003 } }** **{ { 102 } 3 1 7183 152 21 320 240 { 102 } }** Returns { nCount TwindowState pData[ ] } (Id Kind nState nStateChange x y w h ZAfter Refer to **"QueryAllWindows" on page 141.** **+GalWinSys** **+LiveVideoSys** **+RGBSys** **+WinServer**
QueryECCInfo [out] TCPSysMonECCInfo) **TCPSysMonECCInfo** **+SysMonEx QueryECCInfo** **=00000000 { 0 0 0 0 }** Returns the ECC memory status. This method uses the data structure
QueryLastSetLayout ([out, string]) **+WinServer QueryLastSetLayout** **=00000000 "Friday's Layout"** The name of the last layout loaded will be retrieved by this command. However, use of this command gets complicated with many users. In a scenario where different users use the same wall with different permissions, a user trying to retrieve the name of a layout that was loaded 10 minutes ago may not retrieve it with this command if another user loaded a different layout in the last few minutes. Only the name of the last layout that was loaded will be retrieved by this command, regardless of who loaded the layouts and who issued the command.
QueryValues ([in] CPSysMonValueIndex_array_t) ([out] **TCPSysMonValue_array_t)** **+SysMonEx QueryValues { 2 2 17 }** **=00000000 { 2 { 2 29 0 0 } { 17 11977 0 11977 } }** Returns specific values, listed by codes specified. QueryValues { n codea codeb }
QueryWindows ([in] WinId_t_array_t, [out] TWindowState_array_t) **+WinServer QueryWindows { 3 { 10001 } { 10002 } { 123 } }** **=00000000 { 102 } 3 1 7183 152 21 320 240 { 102 }** Returns TwindowState_array_t for each specified window ID.

Quit ( ); **+WinServer Quit { }** (quits (exits) ControlPoint Server)
RegisterNotifyTarget () **+WinServer RegisterNotifyTarget { }** (turns on notify messages) **RegisterNotifyTarget ( )** **+EventLog RegisterNotifyTarget** **=00000000** Registers the client to receive event notifications from the server. Refer to Object, **"EventLogNotify" on page 75**. **RegisterNotifyTarget ( )** **+SysMonEx RegisterNotifyTarget** Registers the client to receive notifications when the system status changes. Refer to Object **"SysMonNotifyEx" on page 99**. Use this method to turn on the notify messages.
SaveLayout ([in, string], [in] WinId_t_array_t) **+WinServer SaveLayout "layoutname" { 0 }** Saves named layout
ScreenConfigChanged ([in] CPScreenConfig) **+Notify ScreenConfigChanged { TotalWidth TotalHeight SingleScreenWidth SingleScreenHeight }** Refer to **Notify** listing
SetAutoDetectTiming ([in] WinId_t wid, [in] Boolean bEnable) **+RGBSys SetAutoDetectTiming { 123 } 1** **=00000000** (enable/disable (1/0) auto-detection of RGB timing)
SetChannel ([in] WinId_t, [in] short); **+LiveVideoSys SetChannel { 123 } 1** (Sets window 123 to channel 1) **SetChannel ([in] WinId_t, [in] short)** **+RGBSys SetChannel { 123 } 1** **=00000000 { 123 } 1** (sets window 123 to RGB channel 1)
SetComponent ([in] WinId_t wid, [in] Boolean bEnabled) **+RGBSys SetComponent { WinId } 1** **=000000000** Sets the classification of 1 for SetComponent and 0 for RGB

Methods

SetConnection ([in] WinId_t wid, [in,string] connection)
+CPShareSys SetConnection { 1 } “Elara”
=00000000
returns the name of the connection object the CPShare window is using

SetCrop ([in] WinId_t, [in] CPRect)
+GalWinSys SetCrop { 123 } {nX nY nW nH }
Crop works by relation to the normal image size. nX and nY specify the pixels removed from the top and bottom. nW and nH specify the size of the cropped image. A 640x480 window set at 10 10 600 400 will have 10 pixels removed from the top, 10 pixels removed from the left, 30 and 70 removed from the right and bottom. ImageWidth - nX - (right pixels) = nW or 640 – 10 – 30 = 600.

+GalwinSys
+VideoStreamSys

SetDualLink ([in] WinId_t wid, [in] Boolean bEnabled)
+RGBSys SetDualLink { WinId } 1
=00000000
Sets the classification of 1 for Dual Link and 0 for Single Link.

Object Method { WinId } { Set [options] }

+Window SetFrameInfo { WinId }
{ Set FrameWidth FrameColor ShowTitle ShowUserData }
Frame width (FW) in pixels – 0 to 100
Frame color (FC) = RGB
Red = 1 to 255
Blue = 1 to 255
Green = 1 to 255
White is all color on – (256) x (256) x (256) = 0xFFFFFF or 16,777,215
Black is all color off – 000 = 0x000000.
Example – green only – 00 FF 00 = 65,280 = 0x00FF00
Show title = 1 = display title – 0 = title off

+Window SetFrameInfo { 123 } { 7 10 00FF00 1 0 }
Green only - Set = 7 width, color and title all valid.

SetImgBalance ([in] WinId_t, [in, out] ImgBalance)
+GalWinSys SetImageBalance { 123 }
{Set Brightness Contrast Gamma Hue Saturation }
=00000000
Sets {Set Brightness Contrast Gamma Hue Saturation }, Uses **Set** to determine valid data values

+GalWinSys
+VidStreamSys

SetLayout ([in, string]) **+WinServer SetLayout "layoutname"** Applies named layout
SetLevel ([in] unsigned NewLevel, [out] unsigned OldLevel)
SetOrigin ([in] WinId_t, [in] long x, [in] long y) **+GalWinSys SetOrigin { 123 } x y** **=00000000** Sets origin of cropped image to effect panning
SetOutputs ([in] DWORD new, [out] DWORD old)
SetPatternProp ([in,string] prop_name, [out] struct ScreenTestPattern *) **+ScreenUtil SetPatternProp "setpatternprop" { 0x7ff 1 1 1 1 1 1 0 0 0 1 10 }** Set properties to the current test pattern with the specified values. **+ScreenUtil SetPatternProp "setpatternprop" { Set SingleScreen BarType GridCircle FgColorRed FgColorGreen FgColorBlue BgColorRed BgColorGreen BgColorBlue SmoothGradient LineSpacing }** **+ScreenUtility SetPatternProp "SetPatternProp" { Set [options] } { 0x008 1 0 1 0 1 0 0 0 0 0 0 0 }** Set = 1, single screen = on (0) -- Bar type = 1 -- vertical (2), Foreground = green (16) -- Set = 19 = 0x000013
SetPosition ([in] unsigned flags), ([in] unsigned long recNum) **+EventLog SetPosition flag recNum** **+EventLog SetPosition 2 0** Sets the current position in the event log. The recNumber is the record number to position at, the flags bits determines the seek mode.
SetRCServer ([in] WinId_t wid, [in, string] wchar_t * serverName) **+RGBSys SetRCServer { WinId } NewRCServer** **=00000000** Sets the name or IP address of the RCServer
SetSource ([in] WinId_t wid, [in,string] wchar_t * srcName) **+VidStreamSys SetSource { 123 } "SourceObjectName"** **=00000000** Sets the streaming video source object for the specified window.
SetState ([in, out] TWindowState); **+Window SetState { { WinID_t } Kind nState Set x y w h { ZAfter } }** **+Window SetState { { 123 } 2 1 7183 100 100 320 200 { 1 } }** See SetState

Methods

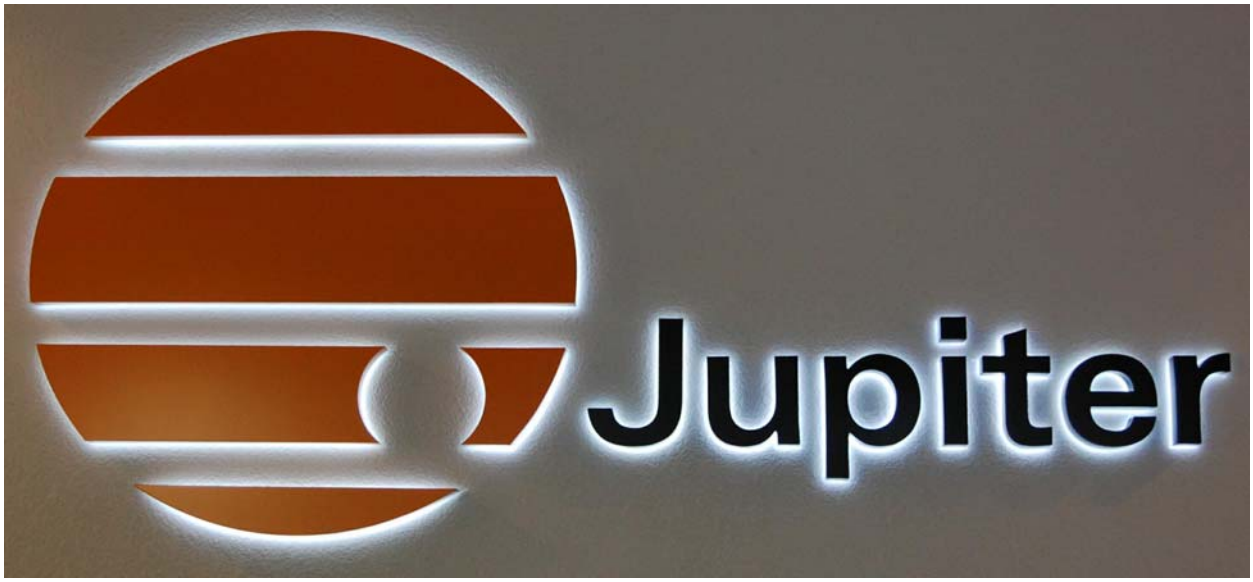
SetText ([in] WinId_t wid, [in,string] wchar_t * text) **+PictureViewerSys SetText { WinId } "Front"** **=00000000** Sets the entered text on the PictureViewer window. The window ID must be specified.
SetTextMode ([in] WinId_t wid, [in] int textmode) **+PictureViewerSys SetTextMode { WinId } 1** **=00000000** Sets the PictureViewer text mode (where 0 = text mode off; 1 = text mode on; 2 = text mode on with text scroll). The window ID must be specified.
SetTiming ([in] WinId_t, [in] CPRGBTiming) **+RGBSys SetTiming { 123 } { bValid nWidth nHTotal nHOffset nHeight nVTotal nVOffset nPhase nVFreq nSyncType bHSynNeg bVSyncNeg }** **+RGBSys SetTiming { 123 } { 1 640 480 144 480 525 35 3 60 0 0 0 }** **=00000000** Sets the RGB Timing parameters
SetTitle ([in] WinId_t, [in, string] title); **+Window SetTitle { 123 } "titlename"** sets title "titlename" to this window.
+Window SetTitleFontInfo { wid } **{ Set TextSize Flags FontName }** **+Window SetTitleFontInfo { 123 } { 5 14 0 "Arial" }** sets the text size = 14, font name = "Arial", and ignores the flags
+Window SetTitleInfo { wid } { Set TextColor BarPos MinBarHeight HorizJust VertJust }+Window SetTitleInfo { 123 } { 31 00FF00 0 20 1 1 } Green only - Set = 31 TextColor BarPos MinBarHeight HorizJust VertJust all valid. Text color = green, on top, 20 pixels high, vertical and horizontal center. **+Window SetTitleInfo { 123 } { 1 0 0 0 0 0 }** Sets the text color to black, and the other parameters are ignored **+Window SetTitleInfo { 123 } { 10 0 1 0 1 0 }** Sets the title bar at the bottom of the window, the text is horizontal center justified, and the other parameters are ignore
SetURL ([in] WinId_t, [in,string]) **+CPWebSys SetURL { 10007 } "www.jupiter.com** **=00000000** Load a web page with the specified URL address.

C—Alphabetic Command Listing of Methods

SetUserInfo ([in, string] Name, [in, string] AuthToken, [in] SetUserInfo ([in, string] Name, [in, string] AuthToken, [in] short Level) **+UserMan SetUserInfo "username" "password" level** sets password and level for username **User Levels** **Level 0** – administrator - full access **Level 1** – user cannot administer users and cannot shut down the server **Level –1 –** is a special case; it means that the account is disabled.
SetVideoSource ([in] WinId_t, [in] CPLiveVideoSource) **+LiveVideoSys SetVideoSource { 123 } { Set Format Type }** **+LiveVideoSys SetVideoSource { 123 } { 3 2 0 }** Set Format and Type (Set=3) to PAL (2) Composite (0)
ShowPattern ([in,string]) **+ScreenUtil ShowPattern "colorpat"** **"stdpat"** - display the Standard test pattern **"screenid"**- display the screen ID **"phasepat"** - display the Phase pattern **"bitmap"** - display a bitmap image **"colorpat"** - display the Color Bar pattern **"colorpat"** - display the Grid pattern **""** - the null function terminates the screen test Display a screen test pattern with the specified pattern name.
ShowPicture ([in] WinId_t, [in,string]) **+PictureViewerSys ShowPicture { 123 } "C:/MyPictures/APic.bmp" =00000000** Displays an image with the specified file path.
Start ([in] WinId_t) **+GalWinSys Start { 123 }** Starts capturing process for specified window **+GalWinSys** **+GenieSubSystem** **+VidStreamSys**
Stop ([in] WinId_t) **+GalWinSys Stop { 123 }** Stops capturing process for specified window **+GalWinSys** **+GenieSubSystem** **+VidStreamSys**

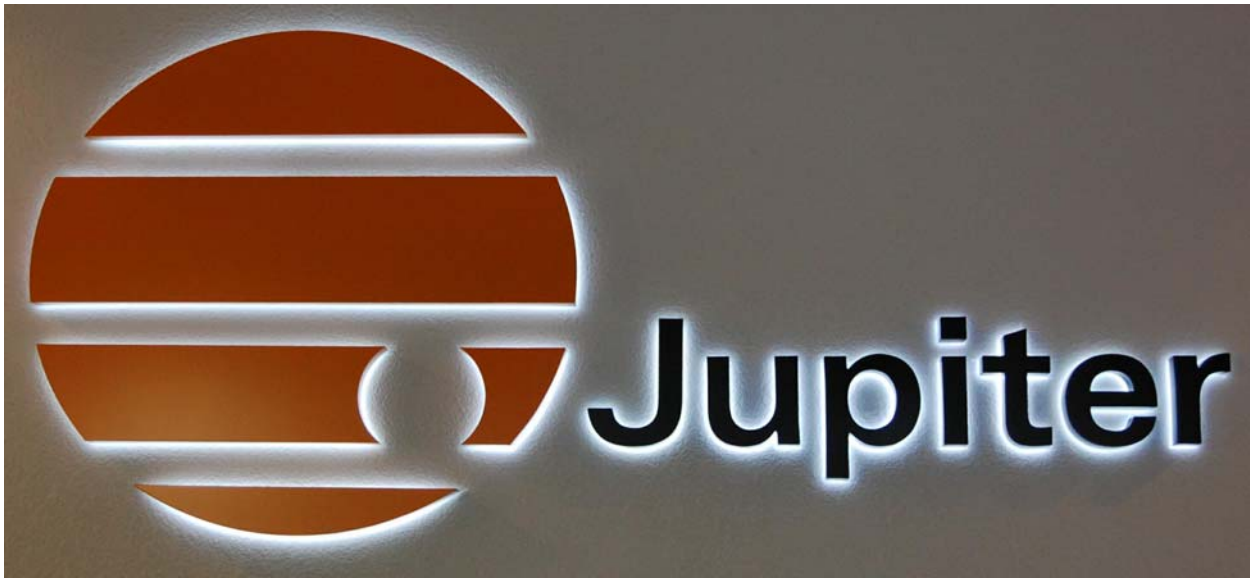
Methods

UnregisterNotifyTarget () **+WinServer UnregisterNotifyTarget { }** Unregisters the client **(**turns off) notify messages **UnregisterNotifyTarget ( )** **+EventLog UnregisterNotifyTarget** **=00000000** Unregisters the client to receive event log notifications**.** **UnregisterNotifyTarget ( )** **+SysMonEx UnregisterNotifyTarget** Unregisters the client to receive system-monitoring notifications.
ValuesChanged ([in] TCPSysMonECCInfo) **+SysMonNotifyEx ValuesChanged** **:SysMonNotifyEx ValuesChanged { 3 { 5 1696 1696 0 } { 128 -5178 -5178 0 } { 48 12464 12464 0 } }** The server calls this method on the client to supply information about values that changed. The supplied parameter is an array (**TCPSysMonValue_array_t**) of the values that have changed since the last notification.
WindowState ([in] TWindowState_array_t) **+Notify WindowsState { nCount TWindowState pData[ ] } (Id Kind nState nStateChange x y w h ZAfter)** See **Notify** listing


Jupiter

Index of Figures

Index of Figures

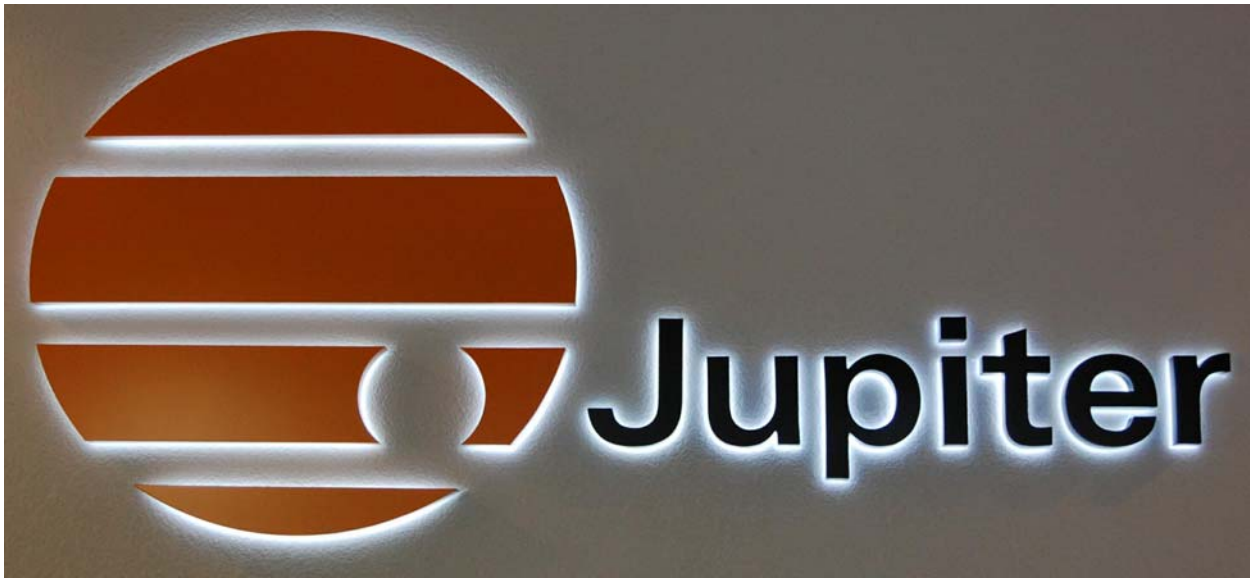

Jupiter

Index of Tables

Index of Tables

Tables

Tables


Jupiter

Index

Index

Index

D

Index

Index

Index

H

I

K

L

Index

M

N

O

P

Q

R

S

Index

Index

Z