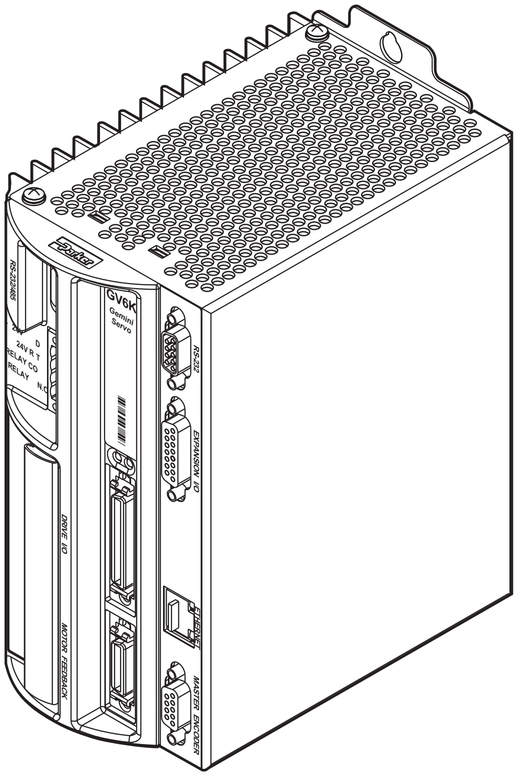
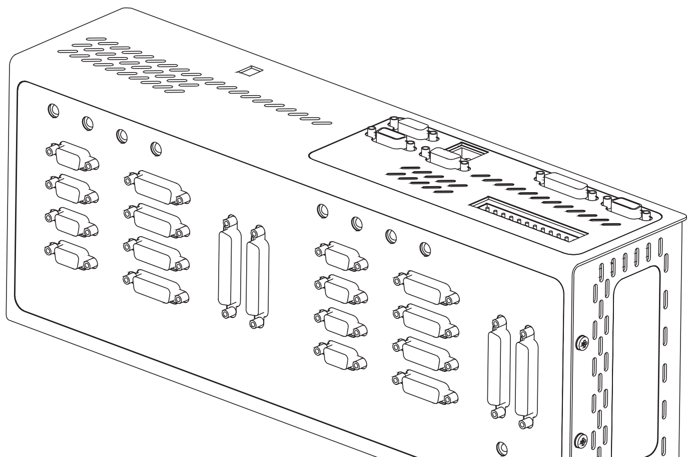
Parker
Automation

p/n 88-020680-01 A

COM6SRVR User's Guide for Gemini & 6K Series Products

Effective: April 2, 2002





IMPORTANT

User Information

⚠ WARNING ⚠

Gemini, 6K, and Gem6K Series products are used to control electrical and mechanical components of motion control systems. You should test your motion system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

Technical Assistance ⇨ *Contact your local automation technology center (ATC) or distributor,* ***or ...***

North America and Asia:
Compumotor Division of Parker Hannifin
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9070 or (707) 584-7558
Fax: (707) 584-3793
FaxBack: (800) 936-6939 or (707) 586-8586
e-mail: tech_help@cmotor.com
Internet: http://www.compumotor.com

Europe *(non-German speaking)*:
Parker Digiplan
21 Balena Close
Poole, Dorset
England BH17 7DX
Telephone: +44 (0)1202 69 9000
Fax: +44 (0)1202 69 5750

Germany, Austria, Switzerland:
HAUSER Elektronik GmbH
Postfach: 77607-1720
Robert-Bosch-Str. 22
D-77656 Offenburg
Telephone: +49 (0)781 509-0
Fax: +49 (0)781 509-176

Parker
Automation

Technical Support
E-mail: Tech_Help@cmotor.com

Communications Server (COM6SRVR)

Programming Samples

Examples may be installed with Motion Planner and are located in the Motion Planner directory (\Motion Planner\Samples):

- Visual Basic 5.0 sample
- Visual C++ sample
- Delphi 3.0 sample

NOTE: The samples are not installed as part of the "typical" installation; use the "custom" installation option.

The Compumotor Communications Server (COM6SRVR.EXE) is a 32-bit OLE automation server which facilitates communications between 6K controllers, Gemini controllers (i.e. GV6K, GT6K, GV6, and GT6), and PC software applications. It is compatible with any 32-bit software application or programming environment which can utilize an OLE automation component, including:

- Visual Basic
- Visual C++
- Delphi
- Software packages that support Microsoft's Component Object Model (COM):
  - Wonderware's Factory Suite 2000
  - National Instruments LabVIEW

The Motion Planner installation program installs COM6SRVR.EXE in the Windows\System (Windows 95/98) or WinNt\System32 (Windows NT/2000) directory.

To begin communications, an application simply needs to request a connection to a 6K controller or Gemini controller through the Communications Server. The Communications Server manages the actual connection to each controller, and can feed information from a particular controller to all client applications which require the information.

Although the Communications Server only makes one connection to each 6K or Gemini, it can feed the information from that one connection to multiple client applications. This means, for example, that a terminal application created in Visual Basic and a terminal in Motion Planner can be connected to the same 6K or Gemini at the same time. They will both receive the same responses coming from the controller, instead of competing for the data. It is also possible for an application to request connections to multiple controllers via the Communications Server. Each connection can be either Ethernet or RS-232 for the 6K, GV6K, and GT6K. The GV6 and GT6 support RS-232 only.

For RS-232 connections, you need to specify the PC COM port on which to connect. For Ethernet connections, you need to specify the controller's IP address. Each controller is set with a default IP address (192.168.10.30). If there is an address conflict with other devices on the network, you can change the 6K's or Gem6K's address with the `NTADDR` command. To do this you must cycle power or issue a reset to invoke the new address (refer to the Ethernet configuration procedures in the 6K Programmer's Guide or the Gem6K Programmer's Guide). The Communications Server can handle up to two RS-232 connections and unlimited Ethernet connections (to different IP addresses).

The syntax for requesting a connection to the Communications Server varies depending on the programming environment being used. Below are examples in the Visual Basic, Visual C++, and Delphi programming formats (refer also to the samples in the Motion Planner directory). To disconnect, refer to "How to Disconnect" instructions on page 3.

COM6SRVR Application Programming Interface (API): Once the proper object variable has been created and a connection is established, there is a standard set of methods and properties which the client application(s) can access.

- For 6K and Gem6K RS-232 methods, refer to page 4
- For GV6 and GT6 RS-232 methods, refer to page 9.
- For 6K Ethernet methods and properties, refer to page 12.
- For Gem6K Ethernet methods and properties, refer to page 33.

Visual Basic Connection Example

```
'create an object variable, initialize it to a
'6K Ethernet interface and make a connection

Dim commserver As Object
Dim ConnectReturnValue As Integer
Set commserver = CreateObject("COM6SRVR.NET")
ConnectReturnValue = commserver.Connect("192.168.10.30")

'------------------------------------------------

'create an object variable, initialize it to a
'RS-232 interface and make a connection to PC COM1

Dim MyMachine As Object
Dim ConnectReturnValue As Integer
Set MyMachine = CreateObject("COM6SRVR.RS232")
ConnectReturnValue = MyMachine.Connect(1)
```

NOTE: When using VBScript, the syntax is identical to the example above, except that the variable declaration should omit the "`As Object`" and "`As Integer`" keywords.

Visual C++ Connection Example

```
/* create an object variable, initialize it to a
Gem6K Ethernet interface and make a connection */

INet commserver;
commserver.CreateDispatch ("COM6SRVR.GEM6K");
int ConnectReturnValue = commserver.Connect("192.168.10.30");

/*==============================================*/

/* create an object variable, initialize it to a
RS-232 interface and make a connection to COM2 */

IRS232 MyMachine;
MyMachine.CreateDispatch ("COM6SRVR.RS232");
int ConnectReturnValue = MyMachine.Connect(2);
```

Delphi Connection Example

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComObj;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    CommServer: Variant;         { Create the object variable }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize CommServer object to a 6K Ethernet interface }
  CommServer := CreateOleObject('COM6SRVR.NET');
  { For RS-232, use CommServer := CreateOleObject('COM6SRVR.RS232');  }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  { Make a connection to the 6k Controller at IP 192.168.10.30 }
  CommServer.Connect('192.168.10.30');
  { To connect via RS-232 on COM2, use CommServer.Connect(2);  }
end;

end.
```

How to Disconnect

The Communications Server is designed as an "EXE" (out-of-process) server rather than a "DLL" (in-process) server. This means that it runs independently of the client application's process. This feature allows the same data from the Communications Server to be shared among several clients. It also provides a more secure connection model by insulating the 6K Communications Server from failure on any singular client.

With the use of an in-process server, the server itself runs in the client's process. If the client application fails or shuts down, the server will be shutdown along with the client. With the use of an out-of-process server, the server runs independently of the client and is therefore insulated from a failure in the client's process. If a particular client application fails, the server will continue to run and provide data to any other client applications requiring its service.

As an out-of-process server, the Communications Server does not shutdown until all client applications have disconnected from the server. In many cases, a proper disconnect does not take place if an unhandled error occurs in the client application and the program exits abnormally. This means that care must be exercised on the part of the client program to disconnect from the server on such occasions or when its services are no longer needed.

VB and VBScript

For VB/VBScript applications, an object variable is typically released when the variable loses scope. However, it is always a good practice to explicitly release the object by setting it to nothing.

```
'assuming the commserver is an object variable
'representing a Communications Server connection

Set commserver = Nothing;   'free the object - disconnect from the server
```

C++

In C++, the same rule applies to the scope of an object variable, but again it is good programming practice to explicitly release the object.

```
//assuming the commserver is an object variable
//representing a Communications Server connection

commserver.ReleaseDispatch();     // release the IDispatch connection
```

Delphi

Again, in Delphi, the same rule applies.

```
{ assuming the CommServer is an object variable        }
{ representing a Communications Server connection   }

CommServer := UnAssigned;     { release the connection }
```

Be Aware of Background Commands

During some operations in the Communications Server, it is necessary for the server to send setup commands to the controller. These commands generally affect communication port settings and are necessary for proper communications between the server and the controller.

The use of these commands may affect settings previously established in the controller by a user program, so it may be necessary to adjust the settings after certain methods in the Communications Server are exercised. The use of these commands will also affect the command count data available in FastStatus Ethernet property.

The Communications Server methods which invoke background commands are:

RS-232 Methods:	**Connect**, **GetFile**, **SendFile**, **SendFileQuiet**, and **SendOS**
Ethernet Methods:	**Connect**, **GetFile**, **SendFile, SendFileQuiet, SendFileBlocking** and **SendFileBlockingQuiet.**

For details on the background commands sent, refer to the description (below) for the respective method.

COM6SRVR.RS232 Interface – RS-232 communication with 6K or Gem6K

RS-232 Methods

NOTE
This section covers RS-232 methods, there are no RS-232 properties for the Communications Server.

Connect (port)

Description:	The Connect method opens an RS-232 connection.
Visual Basic:	object.**Connect**(port as Integer) As Integer
Visual C++:	short object.**Connect**(short commport)
Delphi:	Smallint_variable := Object_variable.**Connect**(port as Smallint)
Parameter:	port (commport) Short Integer Represents the PC's COM port number (1-6).
Return Type:	Short Integer. If the connection is successfully opened, the method returns a positive value representing the number of connected clients. If the connection is unsuccessful, then an error code is returned (see table on page 54).
Remarks:	The Server can handle up to two RS-232 connections. The RS232 server assumes 9600 Baud operation. Background Commands: After a successful connection is made, a "PORT0:" command is sent to the controller.

Flush

Description:	The Flush method removes all characters from the client's receive buffer. This method allows the programmer to clear the receive buffer prior to making a read.
Visual Basic:	object.**Flush**
Visual C++:	void object.**Flush**()
Delphi:	Object_variable.**Flush**
Parameter :	NONE
Return Type:	NONE
Remarks:	**USE WITH CAUTION.** This method allows the programmer to clear the receive buffer, such that a subsequent Read call can yield a clean response. However, data arriving in the receive buffer is asynchronous to the application program and a thorough understanding of how the application program is structured is necessary to use this method correctly (for example, it would not be beneficial to Flush the buffer if only a partial response has been received).

GetFile (filename)

Description:	The GetFile method is used to upload programs currently stored in the controller.
Visual Basic:	object.**GetFile**(filename as String) As Long
Visual C++:	long object.**GetFile**(LPCTSTR lpFileName)
Delphi:	Longint_variable := Object_variable.**GetFile**(filename as String)
Parameter:	filename String. Represents the name of the file to store the uploaded programs. If the filename is an empty string, then the user will be prompted for the filename.
Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page 54).

Remarks: Background Commands: At the beginning of a file upload operation, these commands are sent to the controller:

```
!PORT0
!ECHO0
!ERRLVL0
!EOT1,0,0
!EOL10,0,0
!TDIR
```

For each program selected for upload, a "!TPROG" command is also sent to the controller.

After the upload process is completed, these commands are sent to the controller:

```
!PORT0
!EOT13,0,0
!EOL13,10,0
!ERRLVL4
!ECHO1
```

Read ( )

Description: The Read method retrieves command responses from the controller.

Visual Basic: object.**Read**() As String

Visual C++: object.CString **Read**()

Delphi: String_variable := Object_variable.**Read**

Parameter: NONE

Return Type: String.
The Read method does not wait for incoming responses from the controller. It returns immediately with a string containing the controller's response at the time of the request. If no response is available, this method will return an empty string. The Read method response is limited to 256 characters. If the response is longer than 256 characters, the excess characters will remain in the COM6SRVR buffer. Multiple reads are necessary for long responses.

Remarks: You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

SendFile (filename)

Description: The SendFile method is used to download program files to the controller.

Visual Basic: object.**SendFile**(filename as String) As Long

Visual C++: long object.**SendFile**(LPCTSTR lpFileName)

Delphi: Longint_variable := Object_variable.**SendFile**(filename as String)

Parameter: filename String.
Represents the name of the program file (containing 6K or Gem6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.

Return Type: Long integer.
The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page 54).

Remarks: To speed up downloads, the SendFile method strips comments from the downloaded code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed.

NOTE: The SendFile method should be called when motion is not in progress and programs are not running.

Background Commands: At the beginning of a file download operation, these commands are sent to the controller:

```
!PORT0
!ECHO0
!ERRLVL0
!EOT1,0,0
!EOL10,0,0
!TDIR
```

After the download process is completed, these commands are sent to the controller:

```
!PORT0
!EOT13,0,0
!EOL13,10,0
!ERRLVL4
!ECHO1
```

NOTE: If the download process is canceled, an "END" command is sent to the controller.

SendFileQuiet (filename)	Description:	The SendFileQuiet method is used to download program files to the controller while suppressing the download status dialog message.
	Visual Basic:	object.**SendFileQuiet**(filename as String) As Long
	Visual C++:	long object.**SendFileQuiet**(LPCTSTR lpFileName)
	Delphi:	Longint_variable := Object_variable.**SendFileQuiet**(filename as String)
	Parameter:	filename String. Represents the name of the program file (containing 6K or Gem6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page 54).
	Remarks:	To speed up downloads, the SendFileQuiet method strips comments from the downloaded code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed.

NOTE: The SendFileQuiet method should be called when motion is not in progress and programs are not running.

Background Commands: At the beginning of a file download operation, these commands are sent to the controller:

```
!PORT0
!ECHO0
!ERRLVL0
!EOT1,0,0
!EOL10,0,0
!TDIR
```

After the download process is completed, these commands are sent to the controller:

```
!PORT0
!EOT13,0,0
!EOL13,10,0
!ERRLVL4
!ECHO1
```

NOTE: If the download process is canceled, an "END" command is sent to the controller.

SendOS (filename)	Description:	The SendOS method downloads the soft operating system to a 6K or Gem6K controller.
	Visual Basic:	`object.`**`SendOS`**`(filename as String) As Boolean`
	Visual C++:	`BOOL object.`**`SendOS`**`(LPCTSTR lpFileName)`
	Delphi:	`Boolean_variable := Object_variable.`**`SendOS`**`(filename as String)`
	Parameter:	filename — String. Represents the name of soft operating system file. If filename is an empty string then the user will be prompted for the operating system file name.
	Return Type:	Boolean. (This method returns a Boolean value.) The method returns a TRUE value if the operation is successful; otherwise, a FALSE value is returned.
	Remarks:	After downloading a new operating system, the appropriate NTFEN command must be sent to the controller (see the 6K Command Reference or Gem6K Command Reference) — this applies only if you will be using Ethernet communication. Background Commands: Before the operating system download process, COM6SRVR sends several setup commands to the controller, followed by a RESET command. **NOTE**: The download process uses a baud rate of 38400. This allows for fast download times. After the download process is completed, the previous baud rate is reinstated. The Communications server ALWAYS uses 9600 baud for normal communications.
SetBpsRate (baudrate)	Description:	The SetBpsRate method sets the baudrate transmission for the Com6srvr RS232 interface.
	Visual Basic:	`object.`**`SetBpsRate`**`(baudrate as Integer) As Long`
	Visual C++:	`object.CString `**`SetBpsRate`**`(int baudrate)`
	Delphi:	`String_variable := Object_variable.`**`SetBpsRate`**`(baudrate as Integer)`
	Parameter:	baudrate — Integer. An integer value representing the baudrate that Com6srvr will transmit at.
	Return Type:	Long integer. The SetBpsRate method returns the baudrate that was set.
	Remarks:	In order to communicate with a 6K or Gem6K, you must set the baudrate in the Com6srvr and the baudrate in the controller to the same value. Use the 6K command BAUD to set the controller baudrate before changing the Com6srvr baudrate.

Write (cmd)	Description:	The Write method is used to send commands to the controller.
	Visual Basic:	`object.Write(cmd as String) As Long`
	Visual C++:	`long object.Write(LPCTSTR cmd)`
	Delphi:	`Longint_variable := Object_variable.Write(cmd as String)`
	Parameter:	cmd String A string of commands to be sent. Multiple commands can be sent, but each command should be separated with a valid command delimiter (colon, carriage return, or line feed). The command string should be limited to 256 characters or less. Larger command strings may cause an overflow in the controller's command buffer.
	Return Type:	Long integer. This method returns a positive value corresponding to the number of bytes sent, or a negative error code (see table on page 54).
	Remarks:	You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

COM6SRVR.GEMINI Interface – RS-232 communication with GV6 or GT6

RS-232 Methods

NOTE

This section covers RS-232 methods, there are no RS-232 properties for the Communications Server.

The GV6 and GT6 use a binary language and the COM6SRVR.GEMINI interface incorporates a binary to ASCII translator to convert binary commands into 6000 ASCII equivalents.

Connect (port)

Description:	The Connect method opens an RS-232 connection.	
Visual Basic:	object.**Connect**(port as Integer) As Integer	
Visual C++:	short object.**Connect**(short commport)	
Delphi:	Smallint_variable := Object_variable.**Connect**(port as Smallint)	
Parameter:	port (commport)	Short Integer Represents the PC's COM port number (1-6).
Return Type:	Short Integer. If the connection is successfully opened, the method returns a positive value representing the number of connected clients. If the connection is unsuccessful, then an error code is returned (see table on page 54).	
Remarks:	The Server can handle up to two RS-232 connections. The RS232 server assumes 9600 Baud operation.	

Flush

Description:	The Flush method removes all characters from the client's receive buffer. This method allows the programmer to clear the receive buffer prior to making a read.
Visual Basic:	object.**Flush**
Visual C++:	void object.**Flush**()
Delphi:	Object_variable.**Flush**
Parameter :	NONE
Return Type:	NONE
Remarks:	**USE WITH CAUTION.** This method allows the programmer to clear the receive buffer, such that a subsequent Read call can yield a clean response. However, data arriving in the receive buffer is asynchronous to the application program and a thorough understanding of how the application program is structured is necessary to use this method correctly (for example, it would not be beneficial to Flush the buffer if only a partial response has been received).

GetFile (filename)

Description:	The GetFile method is used to upload programs currently stored in the controller.	
Visual Basic:	object.**GetFile**(filename as String) As Long	
Visual C++:	long object.**GetFile**(LPCTSTR lpFileName)	
Delphi:	Longint_variable := Object_variable.**GetFile**(filename as String)	
Parameter:	filename	String. Represents the name of the file to store the uploaded programs. If the filename is an empty string, then the user will be prompted for the filename.
Return Type:		Long integer. The method returns a positive value if the

			operation is successful; otherwise, it returns an error code (see table on page 54).
	Remarks:	Background Commands: At the beginning of a file upload operation, the "!TDIR" command is sent to the controller: For each program selected for upload, a "!TPROG" command is also sent to the controller.	

Read ( )	Description:	The Read method retrieves command responses from the controller.
	Visual Basic:	object.**Read**() As String
	Visual C++:	object.CString **Read**()
	Delphi:	String_variable := Object_variable.**Read**
	Parameter:	NONE
	Return Type:	String. The Read method does not wait for incoming responses from the controller. It returns immediately with a string containing the controller's response at the time of the request. If no response is available, this method will return an empty string. The Read method response is limited to 256 characters. If the response is longer than 256 characters, the excess characters will remain in the COM6SRVR buffer. Multiple reads are necessary for long responses.
	Remarks:	You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

SendFile (filename)	Description:	The SendFile method is used to download program files to the controller.	
	Visual Basic:	object.**SendFile**(filename as String) As Long	
	Visual C++:	long object.**SendFile**(LPCTSTR lpFileName)	
	Delphi:	Longint_variable := Object_variable.**SendFile**(filename as String)	
	Parameter:	filename	String. Represents the name of the program file (containing GV6 or GT6 programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page 54).	
	Remarks:	To speed up downloads, the SendFile method strips comments from the downloaded code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed. **NOTE**: The SendFile method should be called when motion is not in progress and programs are not running. **NOTE**: If the download process is canceled, an "END" command is sent to the controller.	

SendOS (filename)

Description:	The SendOS method downloads the soft operating system to a GV6 or GT6 controller.
Visual Basic:	object.**SendOS**(filename as String) As Boolean
Visual C++:	BOOL object.**SendOS**(LPCTSTR lpFileName)
Delphi:	Boolean_variable := Object_variable.**SendOS**(filename as String)
Parameter:	filename — String. Represents the name of soft operating system file. If filename is an empty string then the user will be prompted for the operating system file name.
Return Type:	Boolean. (This method returns a Boolean value.) The method returns a TRUE value if the operation is successful; otherwise, a FALSE value is returned.
Remarks:	Background Commands: After the operating system download process, COM6SRVR sends a RESET command.

Write (cmd)

Description:	The Write method is used to send commands to the controller.
Visual Basic:	object.Write(cmd as String) As Long
Visual C++:	long object.Write(LPCTSTR cmd)
Delphi:	Longint_variable := Object_variable.Write(cmd as String)
Parameter:	cmd — String. A string of commands to be sent. Multiple commands can be sent, but each command should be separated with a valid command delimiter (colon, carriage return, or line feed). The command string should be limited to 256 characters or less. Larger command strings may cause an overflow in the controller's command buffer.
Return Type:	Long integer. This method returns a positive value corresponding to the number of bytes sent, or a negative error code (see table on page 54).
Remarks:	You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

COM6SRVR.NET Interface – Ethernet communication with 6K

Ethernet Methods

Connect (netaddress)	Description:	The Connect method opens a connection to a 6K controller
	Visual Basic:	`object.`**`Connect`**`(netaddress as String) As Integer`
	Visual C++:	`short object.`**`Connect`**`(LPCTSTR netaddress)`
	Delphi:	`Smallint_variable := Object_variable.`**`Connect`**`(netaddress as String)`
	Parameter:	netaddress String. Represents the target controller's IP address.
	Return Type:	Short Integer. If the connection is successfully opened, the method returns a positive value representing the number of connected clients. If the connection is unsuccessful, then an error code is returned (see table on page **54**).

	Remarks:	The Server can handle unlimited Ethernet connections (to different IP addresses). The 6K takes up to one minute for an Ethernet connection to truly expire and be available for a new connection.
		
		Background Commands: After a successful connection is made, the following commands are sent to the controller:
		`!PORT0`
		`!ERRLVL4`
		`!EOT13,0,0`
		`!EOL13,10,0`
		ECHO mode is initially disabled (ECHO0) by the 6K during Ethernet communications.

Connect2 (netaddress, lMode)	Description:	The Connect2 method opens a connection to a 6K controller and allows specification of a special operating mode. Connect3, Connect2 and Connect are mutually exclusive. When the connection is established with the 6K, if this is the first client for this 6K then the special operating mode will be selected. If this 6K and Com6srvr already have a connection established, the operating mode will NOT be changed.
	Visual Basic:	`object.`**`Connect2`**`(netaddress as String, lMode As Long) As Integer`
	Visual C++:	`short object.`**`Connect2`**`(LPCTSTR netaddress, long 1Mode)`
	Delphi:	`Smallint_variable := Object_variable.`**`Connect2`**`(netaddress as String, 1Mode as Long)`
	Parameter:	netaddress String. Represents the target controller's IP address. 1Mode Long integer. A constant that specifies mode of operation. 0 specifies normal operation (identical to using Connect method). 2 specifies Extended Fast Status with 12 real variables.
	Return Type:	Short Integer. If the connection is successfully opened, the method returns a positive value representing the number of connected clients. If the connection is unsuccessful, then an error code is returned (if the specified 1Mode is illegal, the error code is ER_BADMODE).
	Remarks:	The Server can handle unlimited Ethernet connections (to different IP addresses). The 6K takes up to one minute for an Ethernet connection to truly expire and

be available for a new connection.

Background Commands: After a successful connection is made, the following commands are sent to the controller:

```
!PORT0
!ERRLVL4
!EOT13,0,0
!EOL13,10,0
```

ECHO mode is initially disabled (ECHO0) by the 6K during Ethernet communications.

Connect3 (netaddress, lMode, bQuiet, lTimeout)

Description:	The Connect3 method opens a connection to a 6K controller and allows specification of a special operating mode, dialog behavior and timeout connection period. Connect3, Connect2 and Connect are mutually exclusive. When the connection is established with the 6K, if this is the first client for this 6K then the special operating mode will be selected. If this 6K and Com6srvr already have a connection established, the operating mode will NOT be changed.
Visual Basic:	object.**Connect3**(netaddress as String, lMode As Long, bQuiet as Boolean, lTimeout as Long) As Integer
Visual C++:	short object.**Connect3**(LPCTSTR netaddress, long 1Mode, boolean bQuiet, long lTimeout)
Delphi:	Smallint_variable := Object_variable.**Connect3**(netaddress as String, 1Mode as Long, bQuiet as Boolean, lTimeout as Long)
Parameter:	netaddress String. Represents the target controller's IP address. 1Mode Long integer. A constant that specifies mode of operation. 0 specifies normal operation (identical to using Connect method). 2 specifies Extended Fast Status with 12 real variables. bQuiet Boolean Specifies whether the connection dialog will be shown. True will hide the connection dialog, false shows the connection dialog. lTimeout Long integer A constant that specifies a timeout period in mS for the Ethernet connection attempt. The range for lTimeout is 0-30 seconds.
Return Type:	Short Integer. If the connection is successfully opened, the method returns a positive value representing the number of connected clients. If the connection is unsuccessful, then an error code is returned (if the specified 1Mode is illegal, the error code is ER_BADMODE).
Remarks:	The Server can handle unlimited Ethernet connections (to different IP addresses). The 6K takes up to one minute for an Ethernet connection to truly expire and be available for a new connection.

Background Commands: After a successful connection is made, the following commands are sent to the controller:

```
!PORT0
!ERRLVL4
!EOT13,0,0
!EOL13,10,0
```

ECHO mode is initially disabled (ECHO0) by the 6K during Ethernet communications.

Flush	Description:	The Flush method removes all characters from the client's receive buffer. This method allows the programmer to clear the receive buffer prior to making a read.
	Visual Basic:	object.**Flush**
	Visual C++:	void object.**Flush**()
	Delphi:	Object_variable.**Flush**
	Parameter:	NONE
	Return Type:	NONE
	Remarks:	**USE WITH CAUTION.** This method allows the programmer to clear the receive buffer, such that a subsequent Read call can yield a clean response. However, data arriving in the receive buffer is asynchronous to the application program and a thorough understanding of how the application program is structured is necessary to use this method correctly (for example, it would not be beneficial to Flush the buffer if only a partial response has been received).
GetFile (filename)	Description:	The GetFile method is used to upload programs currently stored in the controller.
	Visual Basic:	object.**GetFile**(filename as String) As Long
	Visual C++:	long object.**GetFile**(LPCTSTR filename)
	Delphi:	Longint_variable := Object_variable.**GetFile**(filename as String)
	Parameter:	filename String. Represents the name of the file to store the uploaded programs. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page 54).
	Remarks:	Background Commands: At the beginning of a file upload operation, these commands are sent to the controller: !PORT0 !ERRLVL0 !EOT1,0,0 !EOL10,0,0 !ECHO0 !TDIR For each program selected for upload, a "!TPROG" command is also sent to the controller. After the upload process is completed, these commands are sent to the controller: !ERRLVL4 !EOT13,0,0 !EOL13,10,0

IsWatchdogTimedOut	Description:	The IsWatchdogTimedOut method interrogates the current status of the Ethernet Watchdog. The Ethernet Watchdog is a handshake established between the COM6SRVR and the 6K to monitor that the Ethernet connection is still active and "connected".
	Visual Basic:	object.**IsWatchdogTimedOut** As Boolean
	Visual C++:	BOOL **IsWatchdogTimedOut**()
	Delphi:	Boolean_variable := Object_variable.**IsWatchdogTimedOut**
	Parameter:	None
	Return Type:	Boolean. A True indicates that the Ethernet connection has been lost (possible causes: the 6K was reset, or the Ethernet connection was broken). The property is cleared when a new Ethernet connection is established.
	Remarks:	For further information, refer to the SetWatchdog method on page 21.
Ping6K(netaddress, lTimeout)	Description:	The Ping6K method attempts to ping the 6K at the IP Address specified.
	Visual Basic:	object.**Ping**(netaddress as String, lTimeout as Long) As Long
	Visual C++:	long **Ping**(LPCTSTR netaddress, long lTimeout)
	Delphi:	Long_variable := Object_variable.**Ping**(netaddress as String, lTimeout as long)
	Parameter:	netaddress String. Represents the target controller's IP address. lTimeout Long integer. Timeout period in mS for Ping6K. The range for lTimeout is (0 – 30000).
	Return Type:	Long The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page 54).
	Remarks:	
Read ( )	Description:	The Read method retrieves command responses from the controller.
	Visual Basic:	object.**Read**() As String
	Visual C++:	CString object.**Read**()
	Delphi:	String_variable := Object_variable.**Read**
	Parameter:	NONE
	Return Type:	String. The read method does not wait for incoming responses from the controller. It returns immediately with a string containing the controller's response at the time of the request. If no response is available, this method returns an empty string. The Read method response is limited to 256 characters. If the response is longer than 256 characters, the excess characters will remain in the COM6SRVR buffer. Multiple reads are necessary for long responses.
	Remarks:	You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

RequestFastStatusUpdate

Description: The RequestFastStatusUpdate method allows the COM6SRVR to request a fast status update as needed, without having to enable the fast status "Streaming Mode" (FSEnabled) or set an update interval (FSUpdateRate).

Visual Basic: object.**RequestFastStatusUpdate** As Integer

Visual C++: short object.**RequestFastStatusUpdate**

Parameter: NONE

Return Type: Short integer.

If the RequestFastStatusUpdate call is successful the method returns the number of bytes sent. If the call is unsuccessful, the method returns a negative error code (see error code table on page **54**).

Remarks: This method is one of two "On Demand" fast status update options. The other option is for the 6K to execute the NTSFS. Using an On Demand update technique is more efficient for interactive PC applications than the Streaming Mode, and reduces network traffic. For an overview of using the fast status, refer to page **57**.

SendFile (filename)

Description: The SendFile method is used to download program files to the controller.

Visual Basic: object.**SendFile**(filename as String) As Long

Visual C++: long object.**SendFile**(LPCTSTR filename)

Delphi: Longint_variable := Object_variable.**SendFile**(filename as String)

Parameter: filename — String.

Represents the name of the program file (containing 6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.

Return Type: Long integer.
The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page **54**).

Remarks: To speed up downloads, the SendFile method strips comments from the downloaded 6K code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed.

<u>Background Commands</u>: At the beginning of a file download operation, these commands are sent to the controller:

```
!PORT0
!ERRLVL0
```

After the download process is completed, these commands are sent to the controller:

```
!PORT0
!ERRLVL4
!EOT13,0,0
!EOL13,10,0
```

NOTE: If the download process is canceled, an "END" command is sent to the controller.

SendFileBlocking (filename)	Description:	This method, like the SendFile method, is used to download program files to the controller. It differs from SendFile in that it blocks the return of the method call until the 6K acknowledges that the file has been downloaded. A dialog informs the user to wait for the 6K to acknowledge. The dialog has a CANCEL button (a software specified Timeout is not provided). If the user clicks the CANCEL button, the method returns the error code –20.
	Visual Basic:	`object.`**`SendFileBlocking`**`(filename as String) As Long`
	Visual C++:	`long object.`**`SendFileBlocking`**`(LPCTSTR lpFileName)`

	Delphi:	`Longint_variable :=`
		`Object_variable.`**`SendFileBlocking`**`(filename as String)`

	Parameter:	filename String. Represents the name of the program file (containing 6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (error code -20 is returned when the user clicks the CANCEL button). Remarks: To speed up downloads, the SendFileBlocking method strips comments from the downloaded 6K code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed. <u>Background Commands</u>: (same as SendFile) **NOTE**: If the download process is canceled, an "END" command is sent to the controller and the error code (-20) is returned.
SendFileQuiet (filename)	Description:	The SendFileQuiet method is used to download program files to the controller while suppressing the download status dialog message.
	Visual Basic:	`object.`**`SendFileQuiet`**`(filename as String) As Long`
	Visual C++:	`long object.`**`SendFileQuiet`**`(LPCTSTR lpFileName)`

	Delphi:	`Longint_variable :=`
		`Object_variable.`**`SendFileQuiet`**`(filename as String)`

	Parameter:	filename String. Represents the name of the program file (containing 6K or Gem6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page **54**).

	Remarks:	To speed up downloads, the SendFileQuiet method strips comments from the downloaded code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed.
		
		NOTE: The SendFileQuiet method should be called when motion is <u>not</u> in progress and programs are <u>not</u> running.
		
		<u>Background Commands</u>: At the beginning of a file download operation, these commands are sent to the controller:
		    `!PORT0`
		    `!ECHO0`

```
!ERRLVL0
!EOT1,0,0
!EOL10,0,0
!TDIR
```

After the download process is completed, these commands are sent to the controller:

```
!PORT0
!EOT13,0,0
!EOL13,10,0
!ERRLVL4
!ECHO1
```

NOTE: If the download process is canceled, an "END" command is sent to the controller.

SendFileQuietBlocking (filename)

Description:	This method, like the SendFileQuiet method, is used to download program files to the controller while suppressing the download dialog. It differs from SendFileQuiet in that it blocks the return of the method call until the 6K acknowledges that the file has been downloaded. A dialog informs the user to wait for the 6K to acknowledge. The dialog has a CANCEL button (a software specified Timeout is not provided). If the user clicks the CANCEL button, the method returns the error code –20.
Visual Basic:	object.**SendFileQuietBlocking**(filename as String) As Long
Visual C++:	long object.**SendFileQuietBlocking**(LPCTSTR lpFileName)
Delphi:	Longint_variable := Object_variable.**SendFileQuietBlocking**(filename as String)
Parameter:	filename — String. Represents the name of the program file (containing 6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (error code -20 is returned when the user clicks the CANCEL button).
Remarks:	To speed up downloads, the SendFileQuietBlocking method strips comments from the downloaded 6K code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed. Background Commands: (same as SendFile) **NOTE**: If the download process is canceled, an "END" command is sent to the controller and the error code (-20) is returned.

SendVariable (nVariableMask, vaValue)

Description:	The SendVariable method sends one variable from the variable packet to the 6K controller.
Visual Basic:	object.**SendVariable**(nVariableMask As Long, vaValue As Variant) As Integer
Visual C++:	short object.**SendVariable**(long nVariableMask, const VARIANT FAR& vaValue)
Parameter:	nVariableMask — Long integer. Specifies the one variable to be sent. Constants are defined for the mask bits (mask bits for Visual Basic and Visual C++ are provided below). Only one bit can be set in the nVariableMask.

vaValue	Variant. Specifies the value of the variable to be sent. The actual variable being sent is specified by the nVariableMask. Because the SendVariable Method can be used to send integer, real or binary variables, the data type can either be a long integer or a double floating point value. Using a Variant parameter allows the flexibility of sending any integer type, while allowing the COM6SRVR to cast the Variant into the appropriate data type.

Return Type: Short integer.
If the SendVariable call is successful, the method returns the number of bytes sent. If the call is unsuccessful, the method returns a negative error code (see error code table on page **54**). Errors codes are returned if more than one bit is set in the nVariableMask or if the Variant data type is incompatible. Error codes are also returned if there are Ethernet communications errors.

Remarks: Refer to page **59** for an overview of using Send Variables packets.
The data range of real variables in the 6K and the number of significant figures available in a double data type in the PC programming language may cause some rounding errors. The 6K can store data with greater significance, but with a smaller range of values (refer to the VAR command in the 6K Series Command Reference and to your PC programming language reference).

Variable Packet Mask Bits for Visual Basic

```
Public Const VARI1 As Long = 1
Public Const VARI2 As Long = 2
Public Const VARI3 As Long = 4
Public Const VARI4 As Long = 8
Public Const VARI5 As Long = 16
Public Const VARI6 As Long = 32
Public Const VARI7 As Long = 64
Public Const VARI8 As Long = 128
Public Const VARI9 As Long = 256
Public Const VARI10 As Long = 512
Public Const VARI11 As Long = 1024
Public Const VARI12 As Long = 2048

Public Const VAR1 As Long = 4096
Public Const VAR2 As Long = 8192
Public Const VAR3 As Long = 16384
Public Const VAR4 As Long = 32768
Public Const VAR5 As Long = 65536
Public Const VAR6 As Long = 131072
Public Const VAR7 As Long = 262144
Public Const VAR8 As Long = 524288
Public Const VAR9 As Long = 1048576
Public Const VAR10 As Long = 2097152
Public Const VAR11 As Long = 4194304
Public Const VAR12 As Long = 8388608

Public Const VARB1 As Long = 16777216
Public Const VARB2 As Long = 33554432
Public Const VARB3 As Long = 67108864
Public Const VARB4 As Long = 134217728
Public Const VARB5 As Long = 268435456
Public Const VARB6 As Long = 536870912
Public Const VARB7 As Long = 1073741824
Public Const VARB8 As Long = &H80000000
```

Variable Packet Mask Bits for Visual C++

```
#define VARI1  0x00000001
#define VARI2  0x00000002
#define VARI3  0x00000004
#define VARI4  0x00000008
#define VARI5  0x00000010
#define VARI6  0x00000020
#define VARI7  0x00000040
#define VARI8  0x00000080
#define VARI9  0x00000100
#define VARI10 0x00000200
#define VARI11 0x00000400
#define VARI12 0x00000800

#define VAR1   0x00001000
#define VAR2   0x00002000
#define VAR3   0x00004000
#define VAR4   0x00008000
#define VAR5   0x00010000
#define VAR6   0x00020000
#define VAR7   0x00040000
#define VAR8   0x00080000
#define VAR9   0x00100000
#define VAR10  0x00200000
#define VAR11  0x00400000
#define VAR12  0x00800000

#define VARB1  0x01000000
#define VARB2  0x02000000
#define VARB3  0x04000000
#define VARB4  0x08000000
#define VARB5  0x10000000
#define VARB6  0x20000000
#define VARB7  0x40000000
#define VARB8  0x80000000
```

SendVariablePacket (vaPacket)

Description:	The SendVariablePacket method sends a packet of variables to the 6K controller. A complete packet of variables (comprising 6K integer variables 1-12, real variables 1-12 and binary variables 1-8) are always sent. (Refer to the Variable Structures listed below for VB and VC++.) Also included in the packet is a mask, which allows specific variables to be write-protected or over-written.
Visual Basic:	object.**SendVariablePacket**(vaPacket As Variant) As Integer
Visual C++:	short object.**SendVariablePacket** (const VARIANT FAR& vaPacket)
Parameter:	vaPacket Variant. An array of bytes representing the SendVariable packet. The array of bytes comprise: mask bits, reserved elements and bytes of data for the variables. To send a variable packet: 1. Create a structure (TypeDef) and populate the structure with the mask and the variable values. 2. Create an array of bytes from the structure (VB uses Windows API function CopyMemory, Visual C++ uses SAFEARRAYS). 3. Pass the array of bytes as a Variant to the SendVariablePacket method. Refer also to the examples in the SimpleOnePlus sample VB application.
Return Type:	Short integer. If the SendVariablePacket call is successful the method returns the number of bytes sent. If the call is unsuccessful, the method returns a negative error code (see error code table on page 54). Errors codes are returned if the variant data type is incompatible or if there are Ethernet communication errors.
Remarks:	Refer to page 59 for an overview of using Send Variables packets. A list of mask bits for Visual Basic and Visual C++ is provided in the SendVariables method description above. The data range of real variables in the 6K and the number of significant figures available in a double data type in the PC programming language may cause some rounding errors. The 6K can store data with greater significance, but with a smaller range of values (refer to the VAR command in the 6K Series Command Reference and to your PC programming language reference).

Variable Structure for Visual Basic

```
Type SendVariableStructure
  Mask As Long
  Reserved1 As Long
  Reserved2 As Long
  Reserved3 As Long
  VarI(1 To 12) As Long
  VarR(1 To 12) As Double
  VarB(1 To 8) As Long
End Type
```

Variable Structure for Visual C++

```
typedef struct VARIABLEPACKETStruct {
 int nVariableMask;
 int nReserved1;
 int nReserved2;
 int nReserved3;
 int VARI[12];
 double VAR[12];
 int VARB[8];
} VARIABLEPACKET, *LPVARIABLEPACKET;
```

SetSendFileDelay (delay)	Description:	SetSendFileDelay allows you to specify the delay for each character when making a call to the SendFile method. (see Remarks below for detail)
	Visual Basic:	`object.`**`SetSendFileDelay`**`(delay As Integer) As Integer`
	Visual C++:	`short object.`**`SetSendFileDelay`**`(short delay)`
	Parameter:	delay — Integer. The parameter specifies the delay for each character transmitted. Valid range is 0-100 (milliseconds). 0 = no delay.
	Return Type:	Short integer. Returns zero if the specified delay is valid. If the specified delay is out of range, error code –11 is returned.
	Remarks:	When making a call to the SendFile Ethernet method, a 2-ms per character delay is inserted to allow the commands to be transmitted and processed through the TCP/IP stack and the 6K internal buffers. In some cases the delay is not necessary, because the TCP/IP stack takes care of flow-control. In other cases, it might be desirable to allow a longer delay, such as when sending data over a very busy network. This method provides a means to control the delay, allowing a delay of 0-100 ms per character.
SetWatchdog (wTimeout, wTicker)	Description:	The SetWatchdog method enables Ethernet watchdog hand-shaking between the COM6SRVR and the 6K Controller.
	Visual Basic:	`object.`**`SetWatchdog`**`(wTimeout as Integer, wTicker as Integer) as Integer`
	Visual C++:	`short `**`SetWatchdog`**`(short wTimeout, short wTicker)`
	Delphi:	`Smallint_variable := Object_variable.`**`SetWatchdog`**`(wTimeout as Smallint, wTicker as Smallint)`
	Parameters:	wTimeout . Timeout period in seconds (see guidelines below) wTicker .. Number of "heartbeat" packets to send during the timeout period
	Return Type:	Short integer. Returns zero if successful, or a negative error value (usually –11, which indicates that an invalid configuration was specified).
	Remarks:	The Ethernet watchdog allows the COM6SRVR and 6K Controller to gracefully recover when communication between the 6K and COM6SRVR is lost. Such situations might arise from the loss of power to the 6K or to the PC while an Ethernet connection was active. By enabling the Watchdog, a heartbeat packet is sent periodically by the COM6SRVR. The 6K detects the heartbeat and echoes it back to the COM6SRVR. If the COM6SRVR does not detect the echoed heartbeat (within the constraints set by the wTimeout and wTicker parameters), the watchdog is considered timed out. If the 6K does not receive the heartbeat (within the same wTimeout and wTicker constraints), the 6K considers the watchdog timed out. Loss or delay of a single echoed heartbeat could happen quite frequently on a busy network connection. Therefore, we provide a method whereby a number of re-tries are attempted over a specific timeout period. If all re-tries fail within the timeout period, then the watchdog is considered to have timed out. This functionality is provided by the wTimeout and wTicker parameters. The constraints for these parameters are as follows: • To enable the watchdog, set wTimeout > 0 > wTicker. • To disable the watchdog, set wTimeout = 0 and set wTicker = 0.

- The wTimeout/wTicker ratio must be ≤ 65.

RECOMMENDATION: Set wTimeout = 100 and wTicker = 5, which provides a heartbeat once every twenty seconds (100 seconds / 5 tries = 20 seconds/attempt). If none of the 5 heartbeats are acknowledged in 100 seconds, the watchdog times out.

WHEN A WATCHDOG TIMEOUT OCCURS:

- In the 6K: When the 6K detects a watchdog timeout, it attempts to send an alarm packet to the COM6SRVR (AlarmStatus bit #22 – see page **24**). It then closes the Ethernet connection and reports "disconnected" in the TNT report. If the user has enabled error-checking bit #22 (ERROR.22-1), the 6K will execute a GOSUB branch to the ERRORP program. Within the ERRORP program, the watchdog timeout can be cleared by disabling ERROR bit #22 (ERROR.22-0).
- In the COM6SRVR: When the COM6SRVR detects a watchdog timeout, the IsWatchdogTimedOut method (see page **15**) returns TRUE. (If the COM6SRVR receives the alarm packet from the 6K, it will also display an alert dialog to the user.) A client application can poll the IsWatchdogTimedOut. When a timeout is detected by the COM6SRVR, the Client application should "disconnect" the COM6SRVR (if using VB, set COM6SRVR object to Nothing. If using VC++, use ReleaseDispatch). After the COM6SRVR has been disconnected, creating a new Com6srvr object and "connecting" Ethernet will clear the watchdog timeouts. All client applications for that particular 6K Ethernet connection should be disconnected.

Write (cmd)

Description:	The Write method is used to send commands to the controller.
Visual Basic:	object.**Write**(cmd as String) As Integer
Visual C++:	short object.**Write**(LPCTSTR cmd)
Delphi:	Smallint_variable := Object_variable.**Write**(cmd as String)
Parameter:	cmd — String. A string of commands to be sent. Multiple commands can be sent, but each command should be separated with a valid 6K command delimiter (colon, carriage return, or line feed). The command string should be limited to 256 characters or less. Larger command strings may cause an overflow in the 6K's command buffer.
Return Type:	Short integer. This method returns a positive value corresponding to the number of bytes sent, or a negative error code (see table on page **54**).
Remarks:	You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

WriteBlocking (cmd, timeout)	Description:	The WriteBlocking method, an alternative to the Write method, is used to send commands to the controller. The primary difference from the Write method is that WriteBlocking does not return from the method call until the commands have been executed in the controller within a specified time.	
	Visual Basic:	object.**WriteBlocking**(cmd As String, timeout As Integer) As Integer	
	Visual C++:	short object.**WriteBlocking**(LPCTSTR cmd, short timeout)	
	Parameter:	cmd	String. A string of commands to be sent. Multiple commands can be sent, but each command should be separated with a valid 6K command delimiter (colon, carriage return, or line feed). The command string should be limited to 256 characters or less. Larger command strings may cause an overflow in the 6K's command buffer.
		timeout	Integer. Specifies the time period to wait for acknowledgement from the 6K. The time is specified in milliseconds. A value of zero specifies an infinite period.
	Return Type:	Short integer. This method returns a positive value corresponding to the number of bytes sent, or a negative error code. Error code –19 indicates a timeout.	
	Remarks:	The WriteBlocking method can be used as an alternative to the Write method. The discussion below outlines the benefits of the WriteBlocking method, as compared to the Write method. The Write method sends commands to the 6K by placing commands into the TCP/IP send buffer. The Write method then returns when the last 6K command has been placed into the TCP/IP buffer. If the 6K is busy performing motion or processing commands from the buffer, the Write method is very likely to return before the recently written commands are executed within the 6K. The WriteBlocking method sends commands to the 6K by placing commands into the TCP/IP send buffer, just like the Write method. However, WriteBlocking does not return from the method call until the command has been executed within the 6K controller. To prevent a permanently blocked call, a timeout parameter has been added to allow the function to return in the event that the 6K controller does not execute the commands within a specified time. The WriteBlocking functions by requesting an acknowledgement from the 6K that the commands have been executed. A timeout can occur for several reasons: loss of power to the 6K, the Ethernet cable is disconnected, processing of commands took longer than expected, an error occurred within the applications (such as, a KILL occurred or the 6K program has jumped to the error routine). **NOTE**: You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).	

Ethernet Properties

Bit Status Convention

When retrieving bit-oriented properties (e.g., AxisStatus, ErrorStatus, Limits, SystemStatus, etc.) note that the convention in the 6K programming language differs from the convention used for C and Assembly programming languages. Compumotor's 6K convention is to refer to the bits within a 32 bit long integer as bits 1 through 32 (left to right). The C and Assembler Programmer's convention refers to these as bits 0 through 31 (right to left). When masking these bits, you should be aware of this subtle difference when referring to 6K documentation.

AlarmStatus (bit)

Description: The AlarmStatus property returns the state of the controller's alarm status.

Visual Basic: object.**AlarmStatus**(bit As Integer) as Long

Visual C++: long object.**GetAlarmStatus**(short bit)

Delphi: Longint_variable := Object_variable.**AlarmStatus**(bit as Smallint)

Parameter: bit Short Integer.

Specifies the status bit of the alarm status to return. It can be a number between 0 and 32. Values between 1-32 represent the alarm bits as described in the table below (refer also to the INTHW command). Specifying a bit value of 0 returns the entire 32 bit alarms status as a long value; otherwise a value of 1 or 0 is returned to indicate the state of any single bit. When any single bit status is retrieved using the AlarmStatus property, that bit status is automatically cleared by the Communications Server. If a bit value of 0 is used then all alarm status bits are cleared.

Return Type: Long Integer.

Remarks: When the 6K sends an alarm packet to the COM6SRVR, the FastStatus structure is automatically updated, regardless of state of FSEnabled.

Bit #	Function **	Bit #	Function
1	Software (forced) Alarm #1	17	Reserved
2	Software (forced) Alarm #2	18	Reserved
3	Software (forced) Alarm #3	19	Limit Hit - hard or soft limit, on any axis
4	Software (forced) Alarm #4	20	Stall Detected (stepper) or Position Error (servo) on any axis
5	Software (forced) Alarm #5	21	Timer (`TIMINT`)
6	Software (forced) Alarm #6	22	Ethernet fail (`RESET` or `ER.22` occurred) (also invokes an error dialog)
7	Software (forced) Alarm #7	23	Input - any of the inputs defined by `INFNCi-I` or `LIMFNCi-I`
8	Software (forced) Alarm #8	24	Command Error
9	Software (forced) Alarm #9	25	Motion Complete on Axis 1
10	Software (forced) Alarm #10	26	Motion Complete on Axis 2
11	Software (forced) Alarm #11	27	Motion Complete on Axis 3
12	Software (forced) Alarm #12	28	Motion Complete on Axis 4
13	Command Buffer Full	29	Motion Complete on Axis 5
14	ENABLE input Activated	30	Motion Complete on Axis 6
15	Program Complete	31	Motion Complete on Axis 7
16	Drive Fault on any Axis	32	Motion Complete on Axis 8

** Bits 1-12: software alarms are forced with the `INTSW` command.

AnalogInput (channel)

Description: The AnalogInput property returns the value (in counts) of the specified analog input.

Visual Basic: object.**AnalogInput**(channel As Integer) As Long

Visual C++: short object.**GetAnalogInput**(short channel)

Delphi: Smallint_variable := Object_variable.**AnalogInput**(channel as Smallint)

Parameter: channel Short Integer.

Specifies the analog input channel (channel 1 or 2) value to return. This property uses

		only the first two analog inputs detected on an I/O brick connected to the 6K, regardless of the ANIEN (analog input enable) setting.
	Return Type:	Short integer. The method returns the specified analog input value in counts.
	Remarks:	Requires fast status to be enabled with FSEnabled property.

AxisStatus (axis)	Description:	Use the AxisStatus property to retrieve the current axis status for the specified axis.
	Visual Basic:	`object.`**`AxisStatus`**`(axis As Integer) As Long`
	Visual C++:	`long object.`**`GetAxisStatus`**`(short axis)`
	Delphi:	`Longint_variable := Object_variable.`**`AxisStatus`**`(axis as Smallint)`
	Parameter:	axis — Short Integer. Specifies the axis about which the status pertains. The range for this value is 1-8.
	Return Type:	Long Integer. The long integer value represents the current axis status for the specified axis. Refer to the TAS command description for a list of the status elements.
	Remarks:	Requires fast status to be enabled with FSEnabled property.

CommandCount	Description:	Use the CommandCount property to ascertain how many 6K commands have been executed (outside of defined programs) since the 6K controller was powered up.
	Visual Basic:	`object.`**`CommandCount`**` As Long`
	Visual C++:	`long object.`**`GetCommandCount`**`()`
	Delphi:	`Longint_variable := Object_variable.`**`CommandCount`**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the number of 6K commands which have been executed outside of defined programs, since the 6K controller was powered up.
	Remarks:	This is a read-only property. This property allows users to track when commands being sent to the controller (via the communications ports) have been executed. The value is reset to zero each time power is cycled on the 6K. The return value is affected by any background commands sent in conjunction with the Connect, GetFile, and SendFile methods. This property requires fast status to be enabled with the FSEnabled property.

Counter	Description:	The Counter property returns the current Time Frame Counter value.
	Visual Basic:	`object.`**`Counter`**` As Integer`
	Visual C++:	`short object.`**`GetCounter`**`()`
	Delphi:	`Smallint_variable := Object_variable.`**`Counter`**
	Parameter:	NONE
	Return Type:	Short Integer. The values represents the current Time Frame Counter value.
	Remarks:	This is a read-only property. The Time Frame Counter is a free-running timer in the controller. The Counter is updated at the System Update Rate (2 milliseconds). This property requires fast status to be enabled with the FSEnabled property.

EncoderPos (axis)	Description:	The EncoderPos property returns the current encoder position (TPE) in counts for the specified axis.
	Visual Basic:	`object.`**`EncoderPos`**`(axis As Integer) As Long`
	Visual C++:	`long object.`**`GetEncoderPos`**`(short axis)`
	Delphi:	`Longint_variable := Object_variable.`**`EncoderPos`**`(axis)`
	Parameter:	axis Integer. Specifies the axis number of the encoder. The range for this value is 1-8.
	Return Type:	Long Integer. The value represents the current encoder position (TPE) in counts for the specified axis.
	Remarks:	This is a read-only property. Requires fast status to be enabled with FSEnabled property.
ErrorStatus	Description:	The ErrorStatus property returns the current error status (TER) of task 0 only.
	Visual Basic:	`object.`**`ErrorStatus`**` As Long`
	Visual C++:	`long object.`**`GetErrorStatus`**`()`
	Delphi:	`Longint_variable := Object_variable.`**`ErrorStatus`**
	Parameter:	NONE
	Return Type:	Long Integer. The values represents the current error status (TER) of task 0 only.
	Remarks:	Requires fast status to be enabled with FSEnabled property
ExFastStatus	Description:	The ExFastStatus property returns the entire Extended Fast Status data structure.
	Visual Basic:	`object.`**`ExFastStatus`**` As Variant`
	Visual C++:	`VARIANT object.`**`GetExFastStatus`**`()`
	Delphi:	`Variant_variable := Object_variable.`**`ExFastStatus`**
	Parameter:	NONE
	Return Type:	Variant. The variant represents the value of the entire Extended Fast Status data structure.
	Remarks:	This property allows for faster, more efficient retrieval of the Extended Fast Status information if multiple Extended Fast Status items need to be checked at once. The variant is essentially a byte array which can be mapped into an Extended Fast Status structure (see table below for Extended Fast Status and Fast Status structure). The Extended Fast Status structure comprises the regular Fast Status structure, plus 96 additional bytes for the 12 real variables. Each real variable is a 'double' and occupies 8 bytes. **NOTE**: This property requires fast status to be enabled with FSEnabled property. **NOTE**: When the object is first created, the ExFastStatus data will read zeroes. There after, it will report the most recently updated values. When FSEnabled is set to FALSE, the ExFastStatus structure will retain the values from the last update. When the 6K sends an alarm packet to the COM6SRVR, the ExFastStatus structure is automatically updated, regardless of state of FSEnabled.

FastStatus

Description:	The FastStatus property returns the entire FastStatus data structure.
Visual Basic:	`object.`**`FastStatus`**` As Variant`
Visual C++:	`VARIANT object.`**`GetFastStatus`**`()`
Delphi:	`Variant_variable := Object_variable.`**`FastStatus`**
Parameter:	NONE
Return Type:	Variant. The variant represents the value of the entire FastStatus data structure.
Remarks:	This property allows for faster, more efficient retrieval of the FastStatus information if multiple FastStatus items need to be checked at once. The variant is essentially a byte array which can be mapped into a FastStatus structure (see table below for FastStatus structure). The Fast Status structure includes ten integer (VARI) variables and ten binary (VARB) variables that you can use to customize the Fast Status content. Refer to the VB5 sample application SimpleOne in the subroutine cmdGetData_Click() for details of how to convert the byte array data into a Fast Status structure (User Defined Type). Refer to the VC5 sample application VC_Ethernet in the subroutine MakeFastStatus for details on how to convert from a byte array into a Fast Status TypeDef. VBScript does not allow use of structures – use the properties Inputs() and MotorPos(). **NOTE:** This property requires fast status to be enabled with FSEnabled property. **NOTE:** When the object is first created, the FastStatus data will read zeroes. There after, it will report the most recently updated values. When FSEnabled is set to FALSE, the FastStatus structure will retain the values from the last update. When the 6K sends an alarm packet to the COM6SRVR, the FastStatus structure is automatically updated, regardless of state of FSEnabled.

Fast Status — Packet Data Definition (280 bytes total)

Type	Description	Bytes
WORD wUpdateID	Unique update ID for synch channel	2
WORD wCounter	Time Frame Counter	2
DWORD dwMotorPos[8]	Commanded Position (TPC)	32
DWORD dwEncPos[8]	Encoder Position (TPE)	32
DWORD dwMotorVel[8]	Commanded Velocity (TVEL)	32
DWORD dwAxisStatus[8]	Axis Status (TAS)	32
DWORD dwSystemStatus	System Status (TSS)	4
DWORD dwErrorStatus	Error Status (TER)	4
DWORD dwUserStatus	User Status (TUS)	4
DWORD dwTimer	Timer (TTIM)	4
DWORD dwLimits	Limit Status (TLIM)	4
DWORD dwInputs[4]	Input Status (TIN)	16
DWORD dwOutputs[4]	Output Status (TOUT)	16
DWORD dwTriggers	Trigger Status (TTRIG)	4
WORD wAnalogIn[2]	Analog Input Value (TANI - in counts)	4
DWORD dwVarb[10]	Binary Variable Values (VARB1 through VARB10)	40
DWORD dwVari[10]	Integer Variable Values (VARI1 through VARI10)	40
DWORD dwIPAddress	IP Address (NTADDR)	4
DWORD dwCmdCount	Command Count	4
*DWORD dwVar[12]	Real Variable Values (VAR1 through VAR12)	96
	* only applicable with ExFastStatus	

FSEnabled	Description:	The FSEnabled property sets or returns the state of FastStatus polling.
	Visual Basic:	object.**FSEnabled** As Boolean
	Visual C++:	Read: BOOL object.**GetFSEnabled**() Write: void object.**SetFSEnabled**(BOOL state)
	Delphi:	Read: Boolean_variable := Object_variable.**FSEnabled** Write: Object_variable.**FSEnabled** := (state as Boolean)
	Parameter:	Boolean (read/write property).
	Return Type:	Boolean (read/write property).
	Remarks:	The table above lists the items in the FastStatus structure. If the FSEnabled property is set to TRUE, then FastStatus information is automatically retrieved from the controller on a continual basis. BE AWARE that enabling automatic FastStatus polling provides fresh data from the controller on a continual basis, but this will impair the controller's processing time and create a high volume of traffic over the Ethernet network interface. If you intend to enable automatic FastStatus polling, be sure to first set the FSUpdateRate property accordingly. If the FSEnabled property is set to FALSE, automatic FastStatus polling is turned off (but the FastStatus structure will retain the values from the last update).

FSUpdateRate	Description:	The FSUpdateRate property is used to set the millisecond interval on which the controller automatically updates its FastStatus information.
	Visual Basic:	object.**FSUpdateRate** As Integer
	Visual C++:	Read: short object.**GetFSUpdateRate**() Write: void object.**SetFSUpdateRate**(short rate)
	Delphi:	Read: Smallint_variable := Object_variable.**FSUpdateRate** Write: Object_variable.**FSUpdateRate** := (rate as Smallint)
	Parameter:	Short Integer (read/write property).
	Return Type:	Short Integer (read/write property).
	Remarks:	This property should be set before the FSEnabled property is set to TRUE. Setting a larger value for this property means that information will be update less frequently, thereby consuming less of the controller's processing resources. A small value will provide for more frequent updates, but consume more processing time. Valid values for this property are from 10 to 65536. **Visual Basic Users**: COM6SRVR interprets the FSUpdateRate as an unsigned 16-bit integer value. Visual Basic does not support the use of unsigned data types. Therefore, you have to pass a signed 16-bit integer and allow the COM6SRVR to interpret it as unsigned. Thus, to allow slower update intervals than 32767 ms, a VB programmer would pass a negative value (see examples below): Value passed is –1 (result is 65535 ms/update) Value passed is –32768 (result is +32768 ms/update) Value passed is –30000 (result is +35536 ms/update) Value passed is –25536 (result is +40000 ms/update) Value passed is +32767 (result is +32767 ms/update) Value passed is +10 (result is +10 ms/update)

Inputs (brick)	Description:	Use the Inputs property to check the current state of the inputs (TIN) on a specific brick.
	Visual Basic:	object.**Inputs**(brick As Integer) As Long
	Visual C++:	long object.**GetInputs**(short brick)
	Delphi:	Longint_variable := Object_variable.**Inputs**(brick as Smallint)
	Parameter:	brick Short Integer. Represents the number of the brick where the inputs reside. Range is 0-3. Brick 0 represents the onboard inputs. Bricks 1-3 represent expansion I/O bricks 1-3.
	Return Type:	Long Integer. The value represents the current state of the inputs (TIN) for the specified brick.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
IPAddress	Description:	The IPAddress property returns the controller’s IP Address (NTADDR).
	Visual Basic:	object.**IPAddress** As Long
	Visual C++:	long object.**GetIPAddress**()
	Delphi:	Longint_variable := Object_variable.**IPAddress**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the controller’s IP Address (NTADDR).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
Limits	Description:	The Limits property returns the current limit status (TLIM).
	Visual Basic:	object.**Limits** As Long
	Visual C++:	long object.**GetLimits**()
	Delphi:	Longint_variable := Object_variable.**Limits**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current limit status (TLIM).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
MotorPos (axis)	Description:	The MotorPos property returns the current commanded position (TPC) for the specified axis.
	Visual Basic:	object.**MotorPos**(axis As Integer) As Long
	Visual C++:	long object.**GetMotorPos**(short axis)
	Delphi:	Longint_variable := Object_variable.**MotorPos**(axis as Smallint)
	Parameter:	axis Short Integer. Specifies the axis number (range is 1-8).
	Return Type:	Long Integer. The value represents the current commanded position (TPC) in counts for the specified axis.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

MotorVel (axis)	Description:	The MotorVel property returns the current commanded motor velocity (TVEL) for the specified axis.
	Visual Basic:	`object.`**`MotorVel`**`(axis As Integer) As Long`
	Visual C++:	`long object.`**`GetMotorVel`**`(short axis)`
	Delphi:	`Longint_variable := Object_variable.`**`MotorVel`**`(axis as Smallint)`
	Parameter:	axis — Short Integer. Specifies the axis number (range is 1-8).
	Return Type:	Long Integer. The value represents the current commanded velocity (TVEL) in counts for the specified axis.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

Outputs (brick)	Description:	The Outputs property returns the state of the outputs (TOUT) on the specified brick.
	Visual Basic:	`object.`**`Outputs`**`(brick As Integer) As Long`
	Visual C++:	`long object.`**`GetOutputs`**`(short brick)`
	Delphi:	`Longint_variable := Object_variable.`**`Outputs`**`(brick as Smallint)`
	Parameter:	brick — Short Integer. Represents the number of the brick where the outputs reside. Range is 0-3. Brick 0 represents the onboard outputs. Bricks 1-3 represent expansion I/O bricks 1-3.
	Return Type:	Long Integer. The value represents the state of the outputs (TOUT) on the specified brick.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

SystemStatus	Description:	The SystemStatus property returns the system status (TSS) for task 0 only.
	Visual Basic:	`object.`**`SystemStatus`**` As Long`
	Visual C++:	`long object.`**`GetSystemStatus`**`()`
	Delphi:	`Longint_variable := Object_variable.`**`SystemStatus`**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the system status (TSS) for task 0 only.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

Timer	Description:	The Timer property returns the current Timer value (TTIM) for task 0 only.
	Visual Basic:	`object.`**`Timer`**` As Long`
	Visual C++:	`long object.`**`GetTimer`**`()`
	Delphi:	`Longint_variable := Object_variable.`**`Timer`**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current Timer value (TTIM) for task 0 only.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

Triggers	Description:	The Triggers property returns the Trigger Interrupt Status (TTRIG).
	Visual Basic:	object.**Triggers** As Long
	Visual C++:	long object.**GetTriggers**()
	Delphi:	Longint_variable := Object_variable.**Triggers**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current state of the Trigger Interrupt Status (TTRIG).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
UserStatus	Description:	The UserStatus property returns the current state of the user status register (TUS).
	Visual Basic:	object.**UserStatus** As Long
	Visual C++:	long object.**GetUserStatus**()
	Delphi:	Longint_variable := Object_variable.**UserStatus**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current state of the user status register (TUS).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
Var (varnum)	Description:	The Var property returns the value of the specified real variable (VAR). Variables VAR1 through VAR12 may be reported. **NOTE:** This property must be used in conjunction with the Connect2 method (1Mode=2).
	Visual Basic:	object.**Var**(varnum As Integer) As Double
	Visual C++:	double object.**GetVar**(short varnum)
	Delphi:	double_variable := Object_variable.**Var**(varnum as Smallint)
	Parameter:	varnum Short Integer Represents number of the real variable (VARvarnum). Range is 1-12.
	Return Type:	Double. The value represents the value of the specified real variable (VAR). The initial value is zero until an Extended Fast Status packet arrives.
	Remarks:	This property is valid only with the first client connection to 6K, when connected using Connect2 method with 1Mode=2. Read Only. Requires FastStatus to be enabled, or use of RequestFastStatusUpdate or NTSFS command, or generation of an Alarm in the 6K.

VarB (varnum)	Description:	The VarB property returns the value of the specified binary variable (VARB).
	Visual Basic:	object.**VarB**(varnum As Integer) As Long
	Visual C++:	long object.**GetVarB**(short varnum)
	Delphi:	Longint_variable := Object_variable.**VarB**(varnum as Smallint)
	Parameter:	varnum Short Integer. Represents number of the binary variable (VARBvarnum). Range is 1-10.
	Return Type:	Long Integer. The value represents the value of the specified binary variable (VARB).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

VarI (varnum)	Description:	The VarI property returns the value of the specified integer variable (VARI).
	Visual Basic:	object.**VarI**(varnum As Integer) As Long
	Visual C++:	long object.**GetVarI**(short varnum)
	Delphi:	Longint_variable := Object_variable.**VarI**(varnum as Smallint)
	Parameter:	varnum Short Integer. Represents number of the binary variable (VARIvarnum). Range is 1-10.
	Return Type:	Long Integer. The value represents the value of the specified integer variable (VARI).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

COM6SRVR.GEM6K Interface – Ethernet communication with Gem6K

Ethernet Methods

Connect (netaddress)	Description:	The Connect method opens a connection to a Gem6K controller.
	Visual Basic:	object.**Connect**(netaddress as String) As Integer
	Visual C++:	short object.**Connect**(LPCTSTR netaddress)
	Delphi:	Smallint_variable := Object_variable.**Connect**(netaddress as String)
	Parameter:	netaddress String. Represents the target controller's IP address.
	Return Type:	Short Integer. If the connection is successfully opened, the method returns a positive value representing the number of connected clients. If the connection is unsuccessful, then an error code is returned (see table on page **54**).
	Remarks:	The Server can handle unlimited Ethernet connections (to different IP addresses). The Gem6K takes up to one minute for an Ethernet connection to truly expire and be available for a new connection. Background Commands: After a successful connection is made, the following commands are sent to the controller: !PORT0 !ERRLVL4 !EOT13,0,0 !EOL13,10,0 ECHO mode is initially disabled (ECHO0) by the Gem6K during Ethernet communications.
Connect3 (netaddress, bQuiet, lTimeout)	Description:	The Connect3 method opens a connection to a Gem6K controller and allows specification of a special dialog behavior and timeout connection period. Connect3 and Connect are mutually exclusive.
	Visual Basic:	object.**Connect3**(netaddress as String, bQuiet as Boolean, lTimeout as Long) As Integer
	Visual C++:	short object.**Connect3**(LPCTSTR netaddress, boolean bQuiet, long lTimeout)
	Delphi:	Smallint_variable := Object_variable.**Connect3**(netaddress as String, bQuiet as Boolean, lTimeout as Long)
	Parameter:	netaddress String. Represents the target controller's IP address. bQuiet Boolean Specifies whether the connection dialog will be shown. True will hide the connection dialog, false shows the connection dialog. lTimeout Long integer A constant that specifies a timeout period in mS for the Ethernet connection attempt. The range for lTimeout is (0 – 60000).
	Return Type:	Short Integer. If the connection is successfully opened, the method returns a positive value representing the number of connected clients. If the connection is unsuccessful, then an error code is returned.
	Remarks:	The Server can handle unlimited Ethernet connections (to different IP addresses). The Gem6K takes up to one minute for an Ethernet connection to truly expire and be available for a new connection.

Background Commands: After a successful connection is made, the following commands are sent to the controller:

```
      !PORT0
      !ERRLVL4
      !EOT13,0,0
      !EOL13,10,0
```

ECHO mode is initially disabled (ECHO0) by the Gem6K during Ethernet communications.

Flush	Description:	The Flush method removes all characters from the client's receive buffer. This method allows the programmer to clear the receive buffer prior to making a read.
	Visual Basic:	object.**Flush**
	Visual C++:	void object.**Flush**()
	Delphi:	Object_variable.**Flush**
	Parameter :	NONE
	Return Type:	NONE
	Remarks:	**USE WITH CAUTION.** This method allows the programmer to clear the receive buffer, such that a subsequent Read call can yield a clean response. However, data arriving in the receive buffer is asynchronous to the application program and a thorough understanding of how the application program is structured is necessary to use this method correctly (for example, it would not be beneficial to Flush the buffer if only a partial response has been received).
GetFile (filename)	Description:	The GetFile method is used to upload programs currently stored in the controller.
	Visual Basic:	object.**GetFile**(filename as String) As Long
	Visual C++:	long object.**GetFile**(LPCTSTR filename)
	Delphi:	Longint_variable := Object_variable.**GetFile**(filename as String)
	Parameter:	filename String. Represents the name of the file to store the uploaded programs. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page 54).
	Remarks:	Background Commands: At the beginning of a file upload operation, these commands are sent to the controller: !PORT0 !ERRLVL0 !EOT1,0,0 !EOL10,0,0 !ECHO0 !TDIR

For each program selected for upload, a "!TPROG" command is also sent to the controller.

After the upload process is completed, these commands are sent to the controller:

```
      !ERRLVL4
      !EOT13,0,0
      !EOL13,10,0
```

IsWatchdogTimedOut	Description:	The IsWatchdogTimedOut method interrogates the current status of the Ethernet Watchdog. The Ethernet Watchdog is a handshake established between the COM6SRVR and the Gem6K to monitor that the Ethernet connection is still active and "connected".
	Visual Basic:	`object.`**`IsWatchdogTimedOut`**` As Boolean`
	Visual C++:	`BOOL `**`IsWatchdogTimedOut`**`()`
	Delphi:	`Boolean_variable := Object_variable.`**`IsWatchdogTimedOut`**
	Parameter:	NONE
	Return Type:	Boolean. A True indicates that the Ethernet connection has been lost (possible causes: the Gem6K was reset, or the Ethernet connection was broken). The property is cleared when a new Ethernet connection is established.
	Remarks:	For further information, refer to the SetWatchdog method on page **41**.
PingGem6K(netaddress, lTimeout)	Description:	The PingGem6K method attempts to ping the Gem6K at the IP Address specified.
	Visual Basic:	`object.`**`Ping`**`(netaddress as String, lTimeout as Long) As Long`
	Visual C++:	`long `**`Ping`**`(LPCTSTR netaddress, long lTimeout)`
	Delphi:	`Long_variable := Object_variable.`**`Ping`**`(netaddress as String, lTimeout as long)`
	Parameter:	netaddress String. Represents the target controller's IP address. lTimeout Long integer Timeout period in mS for PingGem6K. The range for lTimeout is (0 – 30000).
	Return Type:	Long. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page **54**).
	Remarks:	
Read ( )	Description:	The Read method retrieves command responses from the controller.
	Visual Basic:	`object.`**`Read`**`() As String`
	Visual C++:	`CString object.`**`Read`**`()`
	Delphi:	`String_variable := Object_variable.`**`Read`**
	Parameter:	NONE
	Return Type:	String. The read method does not wait for incoming responses from the controller. It returns immediately with a string containing the controller's response at the time of the request. If no response is available, this method returns an empty string. The Read method response is limited to 256 characters. If the response is longer than 256 characters, the excess characters will remain in the COM6SRVR buffer. Multiple reads are necessary for long responses.
	Remarks:	You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

RequestFastStatusUpdate	Description:	The RequestFastStatusUpdate method allows the COM6SRVR to request a fast status update as needed, without having to enable the fast status “Streaming Mode” (FSEnabled) or set an update interval (FSUpdateRate).
	Visual Basic:	`object.`**`RequestFastStatusUpdate`**` As Integer`
	Visual C++:	`short object.`**`RequestFastStatusUpdate`**
	Parameter:	NONE
	Return Type:	Short integer. If the RequestFastStatusUpdate call is successful the method returns the number of bytes sent. If the call is unsuccessful, the method returns a negative error code (see error code table on page **54**).
	Remarks:	This method is one of two “On Demand” fast status update options. The other option is for the Gem6K to execute the NTSFS command. Using an On Demand update technique is more efficient for interactive PC applications than the Streaming Mode, and reduces network traffic. For an overview of using the fast status, refer to page **57**.
SendFile (filename)	Description:	The SendFile method is used to download program files to the controller.
	Visual Basic:	`object.`**`SendFile`**`(filename as String) As Long`
	Visual C++:	`long object.`**`SendFile`**`(LPCTSTR filename)`
	Delphi:	`Longint_variable := Object_variable.`**`SendFile`**`(filename as String)`
	Parameter:	filename — String. Represents the name of the program file (containing Gem6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page **54**).

	Remarks:	To speed up downloads, the SendFile method strips comments from the downloaded Gem6K code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed.
		
		Background Commands: At the beginning of a file download operation, these commands are sent to the controller:
		`!PORT0`
		`!ERRLVL0`
		
		After the download process is completed, these commands are sent to the controller:
		`!PORT0`
		`!ERRLVL4`
		`!EOT13,0,0`
		`!EOL13,10,0`
		
		NOTE: If the download process is canceled, an “END” command is sent to the controller.

SendFileBlocking (filename)	Description:	This method, like the SendFile method,is used to download program files to the controller. It differs from SendFile in that it blocks the return of the method call until the Gem6K acknowledges that the file has been downloaded. A dialog informs the user to wait for the Gem6K to acknowledge. The dialog has a CANCEL button (a software specified Timeout is not provided). If the user clicks the CANCEL button, the method returns the error code -20.
	Visual Basic:	object.**SendFileBlocking**(filename as String) As Long
	Visual C++:	long object.**SendFileBlocking**(LPCTSTR lpFileName)
	Delphi:	Longint_variable := Object_variable.**SendFileBlocking**(filename as String)
	Parameter:	filename String. Represents the name of the program file (containing Gem6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (error code -20 is returned when the user clicks the CANCEL button).
	Remarks:	To speed up downloads, the SendFileBlocking method strips comments from the downloaded Gem6K code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed. Background Commands: (same as SendFile) **NOTE**: If the download process is canceled, an "END" command is sent to the controller and the error code (-20) is returned.
SendFileQuiet (filename)	Description:	The SendFileQuiet method is used to download program files to the controller while suppressing the download status dialog message.
	Visual Basic:	object.**SendFileQuiet**(filename as String) As Long
	Visual C++:	long object.**SendFileQuiet**(LPCTSTR lpFileName)
	Delphi:	Longint_variable := Object_variable.**SendFileQuiet**(filename as String)
	Parameter:	filename String. Represents the name of the program file (containing 6K or Gem6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.
	Return Type:	Long integer. The method returns a positive value if the operation is successful; otherwise, it returns an error code (see table on page **54**).
	Remarks:	To speed up downloads, the SendFileQuiet method strips comments from the downloaded code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed. **NOTE**: The SendFileQuiet method should be called when motion is not in progress and programs are not running. Background Commands: At the beginning of a file download operation, these commands are sent to the controller:

```
!PORT0
!ECHO0
!ERRLVL0
!EOT1,0,0
!EOL10,0,0
!TDIR
```

After the download process is completed, these commands are sent to the controller:

```
!PORT0
!EOT13,0,0
!EOL13,10,0
!ERRLVL4
!ECHO1
```

NOTE: If the download process is canceled, an "END" command is sent to the controller.

SendFileQuietBlocking (filename)

Description: This method, like the SendFileQuiet method, is used to download program files to the controller while suppressing the download dialog. It differs from SendFileQuiet in that it blocks the return of the method call until the Gem6K acknowledges that the file has been downloaded. A dialog informs the user to wait for the Gem6K to acknowledge. The dialog has a CANCEL button (a software specified Timeout is not provided). If the user clicks the CANCEL button, the method returns the error code –20.

Visual Basic: object.**SendFileQuietBlocking**(filename as String) As Long

Visual C++: long object.**SendFileQuietBlocking**(LPCTSTR lpFileName)

Delphi: Longint_variable := Object_variable.**SendFileQuietBlocking**(filename as String)

Parameter: filename — String.
Represents the name of the program file (containing Gem6K programs/code) to be downloaded. If the filename is an empty string, then the user will be prompted for the filename.

Return Type: Long integer.
The method returns a positive value if the operation is successful; otherwise, it returns an error code (error code -20 is returned when the user clicks the CANCEL button).

Remarks: To speed up downloads, the SendFileQuietBlocking method strips comments from the downloaded Gem6K code. That is, all text between the comment delimiter (semi-colon) and the command delimiter (carriage return or line feed) is removed.

Background Commands: (same as SendFile)

NOTE: If the download process is canceled, an "END" command is sent to the controller and the error code (-20) is returned.

SendVariable (nVariableMask, vaValue)

Description: The SendVariable method sends one variable from the variable packet to the Gem6K controller.

Visual Basic: object.**SendVariable**(nVariableMask As Long, vaValue As Variant) As Integer

Visual C++: short object.**SendVariable**(long nVariableMask, const VARIANT FAR& vaValue)

Parameter: nVariableMask — Long integer.
Specifies the one variable to be sent. Constants are defined for the mask bits (mask bits for Visual Basic and Visual C++ are provided below).

	Only one bit can be set in the nVariableMask.
vaValue	Variant. Specifies the value of the variable to be sent. The actual variable being sent is specified by the nVariableMask. Because the SendVariable Method can be used to send integer, real or binary variables, the data type can either be a long integer or a double floating point value. Using a Variant parameter allows the flexibility of sending any integer type, while allowing the COM6SRVR to cast the Variant into the appropriate data type.

Return Type: Short integer.
If the SendVariable call is successful, the method returns the number of bytes sent. If the call is unsuccessful, the method returns a negative error code (see error code table on page **54**). Errors codes are returned if more than one bit is set in the nVariableMask or if the Variant data type is incompatible. Error codes are also returned if there are Ethernet communications errors.

Remarks: Refer to page **59** for an overview of using Send Variables packets.
The data range of real variables in the Gem6K and the number of significant figures available in a double data type in the PC programming language may cause some rounding errors. The Gem6K can store data with greater significance, but with a smaller range of values (refer to the VAR command in the Gem6K Series Command Reference and to your PC programming language reference).

Variable Packet Mask Bits for Visual Basic

```
Public Const VARI1 As Long = 1
Public Const VARI2 As Long = 2
Public Const VARI3 As Long = 4
Public Const VARI4 As Long = 8
Public Const VARI5 As Long = 16
Public Const VARI6 As Long = 32
Public Const VARI7 As Long = 64
Public Const VARI8 As Long = 128
Public Const VARI9 As Long = 256
Public Const VARI10 As Long = 512
Public Const VARI11 As Long = 1024
Public Const VARI12 As Long = 2048

Public Const VAR1 As Long = 4096
Public Const VAR2 As Long = 8192
Public Const VAR3 As Long = 16384
Public Const VAR4 As Long = 32768
Public Const VAR5 As Long = 65536
Public Const VAR6 As Long = 131072
Public Const VAR7 As Long = 262144
Public Const VAR8 As Long = 524288
Public Const VAR9 As Long = 1048576
Public Const VAR10 As Long = 2097152
Public Const VAR11 As Long = 4194304
Public Const VAR12 As Long = 8388608

Public Const VARB1 As Long = 16777216
Public Const VARB2 As Long = 33554432
Public Const VARB3 As Long = 67108864
Public Const VARB4 As Long = 134217728
Public Const VARB5 As Long = 268435456
Public Const VARB6 As Long = 536870912
Public Const VARB7 As Long = 1073741824
Public Const VARB8 As Long = &H80000000
```

Variable Packet Mask Bits for Visual C++

```
#define VARI1  0x00000001
#define VARI2  0x00000002
#define VARI3  0x00000004
#define VARI4  0x00000008
#define VARI5  0x00000010
#define VARI6  0x00000020
#define VARI7  0x00000040
#define VARI8  0x00000080
#define VARI9  0x00000100
#define VARI10 0x00000200
#define VARI11 0x00000400
#define VARI12 0x00000800

#define VAR1   0x00001000
#define VAR2   0x00002000
#define VAR3   0x00004000
#define VAR4   0x00008000
#define VAR5   0x00010000
#define VAR6   0x00020000
#define VAR7   0x00040000
#define VAR8   0x00080000
#define VAR9   0x00100000
#define VAR10  0x00200000
#define VAR11  0x00400000
#define VAR12  0x00800000

#define VARB1  0x01000000
#define VARB2  0x02000000
#define VARB3  0x04000000
#define VARB4  0x08000000
#define VARB5  0x10000000
#define VARB6  0x20000000
#define VARB7  0x40000000
#define VARB8  0x80000000
```

SendVariablePacket (vaPacket)

Description:	The SendVariablePacket method sends a packet of variables to the Gem6K controller. A complete packet of variables (comprising Gem6K integer variables 1-12, real variables 1-12 and binary variables 1-8) are always sent. (Refer to the Variable Structures listed below for VB and VC++.) Also included in the packet is a mask, which allows specific variables to be write-protected or over-written.
Visual Basic:	object.**SendVariablePacket**(vaPacket As Variant) As Integer
Visual C++:	short object.**SendVariablePacket** (const VARIANT FAR& vaPacket)
Parameter:	vaPacket Variant. An array of bytes representing the SendVariable packet. The array of bytes comprise: mask bits, reserved elements and bytes of data for the variables. To send a variable packet: 1. Create a structure (TypeDef) and populate the structure with the mask and the variable values. 2. Create an array of bytes from the structure (VB uses Windows API function CopyMemory, Visual C++ uses SAFEARRAYS). 3. Pass the array of bytes as a Variant to the SendVariablePacket method. Refer also to the examples in the SimpleOnePlus sample VB application.
Return Type:	Short integer. If the SendVariablePacket call is successful the method returns the number of bytes sent. If the call is unsuccessful, the method returns a negative error code (see error code table on page **54**). Errors codes are returned if the variant data type is incompatible or if there are Ethernet communication errors.
Remarks:	Refer to page **59** for an overview of using Send Variables packets. A list of mask bits for Visual Basic and Visual C++ is provided in the SendVariables method description above. The data range of real variables in the Gem6K and the number of significant figures available in a double data type in the PC programming language may cause some rounding errors. The Gem6K can store data with greater significance, but with a smaller range of values (refer to the VAR command in the Gem6K Series Command Reference and to your PC programming language reference).

Variable Structure for Visual Basic

```
Type SendVariableStructure
  Mask As Long
  Reserved1 As Long
  Reserved2 As Long
  Reserved3 As Long
  VarI(1 To 12) As Long
  VarR(1 To 12) As Double
  VarB(1 To 8) As Long
End Type
```

Variable Structure for Visual C++

```
typedef struct VARIABLEPACKETStruct {
 int nVariableMask;
 int nReserved1;
 int nReserved2;
 int nReserved3;
 int VARI[12];
 double VAR[12];
 int VARB[8];
} VARIABLEPACKET, *LPVARIABLEPACKET;
```

SetSendFileDelay (delay)

Description: SetSendFileDelay allows you to specify the delay for each character when making a call to the SendFile method. (see Remarks below for detail)

Visual Basic: object.**SetSendFileDelay**(delay As Integer) As Integer

Visual C++: short object.**SetSendFileDelay**(short delay)

Parameter: delay — Integer.
The parameter specifies the delay for each character transmitted. Valid range is 0-100 (milliseconds). 0 = no delay.

Return Type: Short integer.
Returns zero if the specified delay is valid. If the specified delay is out of range, error code –11 is returned.

Remarks: When making a call to the SendFile Ethernet method, a 2-ms per character delay is inserted to allow the commands to be transmitted and processed through the TCP/IP stack and the Gem6K internal buffers. In some cases the delay is not necessary, because the TCP/IP stack takes care of flow-control. In other cases, it might be desirable to allow a longer delay, such as when sending data over a very busy network. This method provides a means to control the delay, allowing a delay of 0-100 ms per character.

SetWatchdog (wTimeout, wTicker)

Description: The SetWatchdog method enables Ethernet watchdog hand-shaking between the COM6SRVR and the Gem6K Controller.

Visual Basic: object.**SetWatchdog**(wTimeout as Integer, wTicker as Integer) as Integer

Visual C++: short **SetWatchdog**(short wTimeout, short wTicker)

Delphi: Smallint_variable := Object_variable.**SetWatchdog**(wTimeout as Smallint, wTicker as Smallint)

Parameters: wTimeout . Timeout period in seconds (see guidelines below)
wTicker .. Number of "heartbeat" packets to send during the timeout period

Return Type: Short integer.
Returns zero if successful, or a negative error value (usually –11, which indicates that an invalid configuration was specified).

Remarks: The Ethernet watchdog allows the COM6SRVR and Gem6K Controller to gracefully recover when communication between the Gem6K and COM6SRVR is lost. Such situations might arise from the loss of power to the Gem6K or to the PC while an Ethernet connection was active. By enabling the Watchdog, a heartbeat packet is sent periodically by the COM6SRVR. The Gem6K detects the heartbeat and echoes it back to the COM6SRVR. If the COM6SRVR does not detect the echoed heartbeat (within the constraints set by the wTimeout and wTicker parameters), the watchdog is considered timed out. If the Gem6K does not receive the heartbeat (within the same wTimeout and wTicker constraints), the Gem6K considers the watchdog timed out.

Loss or delay of a single echoed heartbeat could happen quite frequently on a busy network connection. Therefore, we provide a method whereby a number of re-tries are attempted over a specific timeout period. If all re-tries fail within the timeout period, then the watchdog is considered to have timed out. This functionality is provided by the wTimeout and wTicker parameters. The constraints for these parameters are as follows:

- To enable the watchdog, set wTimeout > 0 > wTicker.
- To disable the watchdog, set wTimeout = 0 and set wTicker = 0.

- The wTimeout/wTicker ratio must be ≤ 65.

RECOMMENDATION: Set wTimeout = 100 and wTicker = 5, which provides a heartbeat once every twenty seconds (100 seconds / 5 tries = 20 seconds/attempt). If none of the 5 heartbeats are acknowledged in 100 seconds, the watchdog times out.

WHEN A WATCHDOG TIMEOUT OCCURS:

- In the Gem6K: When the Gem6K detects a watchdog timeout, it attempts to send an alarm packet to the COM6SRVR (AlarmStatus bit #22 – see page **44**). It then closes the Ethernet connection and reports "disconnected" in the TNT report. If the user has enabled error-checking bit #22 (ERROR.22-1), the Gem6K will execute a GOSUB branch to the ERRORP program. Within the ERRORP program, the watchdog timeout can be cleared by disabling ERROR bit #22 (ERROR.22-0).
- In the COM6SRVR: When the COM6SRVR detects a watchdog timeout, the IsWatchdogTimedOut method (see page **35**) returns TRUE. (If the COM6SRVR receives the alarm packet from the Gem6K, it will also display an alert dialog to the user.) A client application can poll the IsWatchdogTimedOut. When a timeout is detected by the COM6SRVR, the Client application should "disconnect" the COM6SRVR (if using VB, set COM6SRVR object to Nothing. If using VC++, use ReleaseDispatch). After the COM6SRVR has been disconnected, creating a new Com6srvr object and "connecting" Ethernet will clear the watchdog timeouts. All client applications for that particular Gem6K Ethernet connection should be disconnected.

Write (cmd)

Description:	The Write method is used to send commands to the controller.	
Visual Basic:	object.**Write**(cmd as String) As Integer	
Visual C++:	short object.**Write**(LPCTSTR cmd)	
Delphi:	Smallint_variable := Object_variable.**Write**(cmd as String)	
Parameter:	cmd	String. A string of commands to be sent. Multiple commands can be sent, but each command should be separated with a valid Gem6K command delimiter (colon, carriage return, or line feed). The command string should be limited to 256 characters or less. Larger command strings may cause an overflow in the Gem6K's command buffer.
Return Type:	Short integer. This method returns a positive value corresponding to the number of bytes sent, or a negative error code (see table on page **54**).	
Remarks:	You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).	

WriteBlocking (cmd, timeout)

Description: The WriteBlocking method, an alternative to the Write method, is used to send commands to the controller. The primary difference from the Write method is that WriteBlocking does not return from the method call until the commands have been executed in the controller within a specified time.

Visual Basic: `object.`**`WriteBlocking`**`(cmd As String, timeout As Integer) As Integer`

Visual C++: `short object.`**`WriteBlocking`**`(LPCTSTR cmd, short timeout)`

Parameter:

cmd	String. A string of commands to be sent. Multiple commands can be sent, but each command should be separated with a valid Gem6K command delimiter (colon, carriage return, or line feed). The command string should be limited to 256 characters or less. Larger command strings may cause an overflow in the Gem6K's command buffer.
timeout	Integer. Specifies the time period to wait for acknowledgement from the Gem6K. The time is specified in milliseconds. A value of zero specifies an infinite period.

Return Type: Short integer.
This method returns a positive value corresponding to the number of bytes sent, or a negative error code. Error code –19 indicates a timeout.

Remarks: The WriteBlocking method can be used as an alternative to the Write method. The discussion below outlines the benefits of the WriteBlocking method, as compared to the Write method.

The Write method sends commands to the Gem6K by placing commands into the TCP/IP send buffer. The Write method then returns when the last Gem6K command has been placed into the TCP/IP buffer. If the Gem6K is busy performing motion or processing commands from the buffer, the Write method is very likely to return before the recently written commands are executed within the Gem6K.

The WriteBlocking method sends commands to the Gem6K by placing commands into the TCP/IP send buffer, just like the Write method. However, WriteBlocking does not return from the method call until the command has been executed within the Gem6K controller. To prevent a permanently blocked call, a timeout parameter has been added to allow the function to return in the event that the Gem6K controller does not execute the commands within a specified time.

The WriteBlocking functions by requesting an acknowledgement from the Gem6K that the commands have been executed. A timeout can occur for several reasons: loss of power to the Gem6K, the Ethernet cable is disconnected, processing of commands took longer than expected, an error occurred within the applications (such as, a KILL occurred or the Gem6K program has jumped to the error routine).

NOTE: You should disable Timer events in VB5 and VBScript when reading and writing to the COM6SRVR (see Microsoft Support Online Article ID176399).

Ethernet Properties

Bit Status Convention

When retrieving bit-oriented properties (e.g., AxisStatus, ErrorStatus, Limits, SystemStatus, etc.) note that the convention in the Gem6K programming language differs from the convention used for C and Assembly programming languages. Compumotor's Gem6K convention is to refer to the bits within a 32 bit long integer as bits 1 through 32 (left to right). The C and Assembler Programmer's convention refers to these as bits 0 through 31 (right to left). When masking these bits, you should be aware of this subtle difference when referring to Gem6K documentation.

ActualAccel

Description:	The ActualAccel property returns the current actual acceleration (TACCA).
Visual Basic:	object.**ActualAccel** as Long
Visual C++:	long object.**GetActualAccel**()
Delphi:	Longint_variable := Object_variable.**ActualAccel**
Parameter:	NONE
Return Type:	Long Integer. The value represents the current actual acceleration (TACCA) in counts.
Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

ActualTorque

Description:	The ActualTorque property returns the current actual torque (TTRQA).
Visual Basic:	object.**ActualTorque** as Long
Visual C++:	long object.**GetActualTorque**()
Delphi:	Longint_variable := Object_variable.**ActualTorque**
Parameter:	NONE
Return Type:	Short Integer. The value represents the current actual torque (TTRQA). A value of -32,768 corresponds to 100% of torque or force in the negative direction based on the DMTSCL command. A value of +32,767 corresponds to 100% of torque or force in the positive direction based on the DMTSCL command.
Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

ActualVelocity

Description:	The ActualVelocity property returns the current actual velocity (TVELA).
Visual Basic:	object.**ActualVelocity** as Long
Visual C++:	long object.**GetActualVelocity**()
Delphi:	Longint_variable := Object_variable.**ActualVelocity**
Parameter:	NONE
Return Type:	Long Integer. The value represents the current actual velocity (TVELA) in counts.
Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

AlarmStatus (bit)

Description:	The AlarmStatus property returns the state of the controller's alarm status.
Visual Basic:	object.**AlarmStatus**(bit As Integer) as Long
Visual C++:	long object.**GetAlarmStatus**(short bit)
Delphi:	Longint_variable := Object_variable.**AlarmStatus**(bit as Smallint)
Parameter:	bit Short Integer. Specifies the status bit of the alarm status to return. It can be a number between 0 and 32.

Values between 1-32 represent the alarm bits as described in the table below (refer also to the INTHW command). Specifying a bit value of 0 returns the entire 32 bit alarms status as a long value; otherwise a value of 1 or 0 is returned to indicate the state of any single bit. When any single bit status is retrieved using the AlarmStatus property, that bit status is automatically cleared by the Communications Server. If a bit value of 0 is used then all alarm status bits are cleared.

Return Type: Long Integer.

Remarks: When the Gem6K sends an alarm packet to the COM6SRVR, the FastStatus structure is automatically updated, regardless of state of FSEnabled.

Bit #	Function **	Bit #	Function
1	Software (forced) Alarm #1	17	Reserved
2	Software (forced) Alarm #2	18	Reserved
3	Software (forced) Alarm #3	19	Limit Hit - hard or soft limit
4	Software (forced) Alarm #4	20	Stall Detected (stepper) or Position Error (servo)
5	Software (forced) Alarm #5	21	Timer (`TIMINT`)
6	Software (forced) Alarm #6	22	Ethernet fail (`RESET` or `ER.22` occurred) (also invokes an error dialog)
7	Software (forced) Alarm #7	23	Input - any of the inputs defined by `INFNCi-I` or `LIMFNCi-I`
8	Software (forced) Alarm #8	24	Command Error
9	Software (forced) Alarm #9	25	Motion Complete
10	Software (forced) Alarm #10	26	Reserved
11	Software (forced) Alarm #11	27	Reserved
12	Software (forced) Alarm #12	28	Reserved
13	Command Buffer Full	29	Reserved
14	ENABLE input Activated	30	Reserved
15	Program Complete	31	Reserved
16	Drive Fault	32	Reserved

** Bits 1-12: software alarms are forced with the `INTSW` command.

AnalogInput (channel)

Description: The AnalogInput property returns the value (in counts) of the specified analog input.

Visual Basic: object.**AnalogInput**(channel As Integer) As Long

Visual C++: short object.**GetAnalogInput**(short channel)

Delphi: Smallint_variable := Object_variable.**AnalogInput**(channel as Smallint)

Parameter: channel Short Integer.
Specifies the analog input channel (channel 1 or 2) value to return. This property uses only the first two analog inputs detected on an I/O brick connected to the Gem6K, regardless of the ANIEN (analog input enable) setting.

Return Type: Short integer.
The method returns the specified analog input value in counts.

Remarks: Requires fast status to be enabled with FSEnabled property.

AxisStatus

Description: Use the AxisStatus property to retrieve the current axis status (TAS).

Visual Basic: object.**AxisStatus** As Long

Visual C++: long object.**GetAxisStatus**()

Delphi: Longint_variable := Object_variable.**AxisStatus**

Parameter: NONE

Return Type: Long Integer.
The long integer value represents the current axis

		status. Refer to the TAS command description for a list of the status elements.
	Remarks:	Requires fast status to be enabled with FSEnabled property.
CommandCount	Description:	Use the CommandCount property to ascertain how many Gem6K commands have been executed (outside of defined programs) since the Gem6K controller was powered up.
	Visual Basic:	object.**CommandCount** As Long
	Visual C++:	long object.**GetCommandCount**()
	Delphi:	Longint_variable := Object_variable.**CommandCount**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the number of Gem6K commands which have been executed outside of defined programs, since the Gem6K controller was powered up.
	Remarks:	This is a read-only property. This property allows users to track when commands being sent to the controller (via the communications ports) have been executed. The value is reset to zero each time power is cycled on the Gem6K. The return value is affected by any background commands sent in conjunction with the Connect, GetFile, and SendFile methods. This property requires fast status to be enabled with the FSEnabled property.
CommandedTorque	Description:	The CommandedTorque property returns the current commanded torque (TTRQ).
	Visual Basic:	object.**CommandedTorque** as Long
	Visual C++:	long object.**GetCommandedTorque**()
	Delphi:	Longint_variable := Object_variable.**CommandedTorque**
	Parameter:	NONE
	Return Type:	Short Integer. The value represents the current commanded torque (TTRQ). A value of -32,768 corresponds to 100% of torque or force in the negative direction based on the DMTSCL command. A value of +32,767 corresponds to 100% of torque or force in the positive direction based on the DMTSCL command.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
ConfigurationStatus	Description:	Use the ConfigurationStatus property to retrieve the current configuration status (TCS).
	Visual Basic:	object.**ConfigurationStatus** As Long
	Visual C++:	long object.**GetConfigurationStatus**()
	Delphi:	Longint_variable := Object_variable. **ConfigurationStatus**
	Parameter:	NONE
	Return Type:	Short Integer. The short integer value represents the current configuration status. Refer to the TCS command description for a list of the status elements.
	Remarks:	Requires fast status to be enabled with FSEnabled property.

Counter	Description:	The Counter property returns the current Time Frame Counter value.
	Visual Basic:	object.**Counter** As Integer
	Visual C++:	short object.**GetCounter**()
	Delphi:	Smallint_variable := Object_variable.**Counter**
	Parameter:	NONE
	Return Type:	Short Integer. The values represents the current Time Frame Counter value.
	Remarks:	This is a read-only property. The Time Frame Counter is a free-running timer in the controller. The Counter is updated at the System Update Rate (2 milliseconds). This property requires fast status to be enabled with the FSEnabled property.
EncoderPos	Description:	The EncoderPos property returns the current encoder position (TPE) in counts.
	Visual Basic:	object.**EncoderPos** As Long
	Visual C++:	long object.**GetEncoderPos**()
	Delphi:	Longint_variable := Object_variable.**EncoderPos**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current encoder position (TPE) in counts.
	Remarks:	This is a read-only property. Requires fast status to be enabled with FSEnabled property.
ErrorStatus	Description:	The ErrorStatus property returns the current error status (TER) of task 0 only.
	Visual Basic:	object.**ErrorStatus** As Long
	Visual C++:	long object.**GetErrorStatus**()
	Delphi:	Longint_variable := Object_variable.**ErrorStatus**
	Parameter:	NONE
	Return Type:	Long Integer. The values represents the current error status (TER) of task 0 only.
	Remarks:	Requires fast status to be enabled with FSEnabled property
ExtendedAxisStatus	Description:	Use the ExtendedAxisStatus property to retrieve the current extended axis status (TASX).
	Visual Basic:	object.**ExtendedAxisStatus** As Long
	Visual C++:	long object.**GetExtendedAxisStatus**()
	Delphi:	Longint_variable := Object_variable.**ExtendedAxisStatus**
	Parameter:	NONE
	Return Type:	Long Integer. The long integer value represents the current extended axis status. Refer to the TASX command description for a list of the status elements.
	Remarks:	Requires fast status to be enabled with FSEnabled property.
FastStatus	Description:	The FastStatus property returns the entire FastStatus data structure.
	Visual Basic:	object.**FastStatus** As Variant
	Visual C++:	VARIANT object.**GetFastStatus**()
	Delphi:	Variant_variable := Object_variable.**FastStatus**
	Parameter:	NONE

Return Type: Variant.

The variant represents the value of the entire FastStatus data structure.

Remarks: This property allows for faster, more efficient retrieval of the FastStatus information if multiple FastStatus items need to be checked at once. The variant is essentially a byte array which can be mapped into a FastStatus structure (see table below for FastStatus structure). The Fast Status structure includes ten integer (VARI) variables and ten binary (VARB) variables that you can use to customize the Fast Status content.

Refer to the VB5 sample application SimpleOne in the subroutine cmdGetData_Click() for details of how to convert the byte array data into a Fast Status structure (User Defined Type). Refer to the VC5 sample application VC_Ethernet in the subroutine MakeFastStatus for details on how to convert from a byte array into a Fast Status TypeDef. VBScript does not allow use of structures – use the properties Inputs() and MotorPos().

NOTE: This property requires fast status to be enabled with FSEnabled property.

NOTE: When the object is first created, the FastStatus data will read zeroes. There after, it will report the most recently updated values. When FSEnabled is set to FALSE, the FastStatus structure will retain the values from the last update. When the Gem6K sends an alarm packet to the COM6SRVR, the FastStatus structure is automatically updated, regardless of state of FSEnabled.

Fast Status — Packet Data Definition (280 bytes total)

Type	Description	Bytes
WORD wUpdateID	Unique update ID for synch channel	2
WORD wCounter	Time Frame Counter	2
DWORD dwMotorPos	Commanded Position (TPC)	4
DWORD dwEncPos	Encoder Position (TPE)	4
DWORD dwMotorVel	Commanded Velocity (TVEL)	4
DWORD dwAxisStatus	Axis Status (TAS)	4
DWORD dwSystemStatus	System Status (TSS)	4
DWORD dwErrorStatus	Error Status (TER)	4
DWORD dwUserStatus	User Status (TUS)	4
DWORD dwTimer	Timer (TTIM)	4
DWORD dwLimits	Limit Status (TLIM)	4
DWORD dwInputs[4]	Input Status (TIN)	16
DWORD dwOutputs[4]	Output Status (TOUT)	16
DWORD dwTriggers	Trigger Status (TTRIG)	4
WORD wAnalogIn[2]	Analog Input Value (TANI - in counts)	4
DWORD dwVarb[10]	Binary Variable Values (VARB1 through VARB10)	40
DWORD dwVari[10]	Integer Variable Values (VARI1 through VARI10)	40
DWORD dwIPAddress	IP Address (NTADDR)	4
DWORD dwCmdCount	Command Count	4
DWORD dwVar[12]	Real Variable Values (VAR1 through VAR12)	96
DWORD dwAccActual	Actual Acceleration (TACCA)	4
DWORD dwExAxisStatus	Extended Axis Status (TASX)	4
WORD wConfigurationStatus	Configuration Status (TCS)	2
WORD wSettlingTime	Settling Time (TSTLT)	2
WORD wTorqueCmd	Commanded Torque (TTRQ)	2
WORD wTorqueActual	Actual Torque (TTRQA)	2
DWORD dwVelActual	Actual Velocity (TVELA)	4

FSEnabled	Description:	The FSEnabled property sets or returns the state of FastStatus polling.
	Visual Basic:	object.**FSEnabled** As Boolean
	Visual C++:	Read: BOOL object.**GetFSEnabled**() Write: void object.**SetFSEnabled**(BOOL state)
	Delphi:	Read: Boolean_variable := Object_variable.**FSEnabled** Write: Object_variable.**FSEnabled** := (state as Boolean)
	Parameter:	Boolean (read/write property).
	Return Type:	Boolean (read/write property).
	Remarks:	The table above lists the items in the FastStatus structure. If the FSEnabled property is set to TRUE, then FastStatus information is automatically retrieved from the controller on a continual basis. BE AWARE that enabling automatic FastStatus polling provides fresh data from the controller on a continual basis, but this will impair the controller's processing time and create a high volume of traffic over the Ethernet network interface. If you intend to enable automatic FastStatus polling, be sure to first set the FSUpdateRate property accordingly. If the FSEnabled property is set to FALSE, automatic FastStatus polling is turned off (but the FastStatus structure will retain the values from the last update).
FSUpdateRate	Description:	The FSUpdateRate property is used to set the millisecond interval on which the controller automatically updates its FastStatus information.
	Visual Basic:	object.**FSUpdateRate** As Integer
	Visual C++:	Read: short object.**GetFSUpdateRate**() Write: void object.**SetFSUpdateRate**(short rate)
	Delphi:	Read: Smallint_variable := Object_variable.**FSUpdateRate** Write: Object_variable.**FSUpdateRate** := (rate as Smallint)
	Parameter:	Short Integer (read/write property).
	Return Type:	Short Integer (read/write property).
	Remarks:	This property should be set before the FSEnabled property is set to TRUE. Setting a larger value for this property means that information will be update less frequently, thereby consuming less of the controller's processing resources. A small value will provide for more frequent updates, but consume more processing time. Valid values for this property are from 10 to 65536. **Visual Basic Users**: COM6SRVR interprets the FSUpdateRate as an unsigned 16-bit integer value. Visual Basic does not support the use of unsigned data types. Therefore, you have to pass a signed 16-bit integer and allow the COM6SRVR to interpret it as unsigned. Thus, to allow slower update intervals than 32767 ms, a VB programmer would pass a negative value (see examples below): Value passed is –1 (result is 65535 ms/update) Value passed is –32768 (result is +32768 ms/update) Value passed is –30000 (result is +35536 ms/update) Value passed is –25536 (result is +40000 ms/update) Value passed is +32767 (result is +32767 ms/update) Value passed is +10 (result is +10 ms/update)

Inputs (brick)	Description:	Use the Inputs property to check the current state of the inputs (TIN) on a specific brick.
	Visual Basic:	object.**Inputs**(brick As Integer) As Long
	Visual C++:	long object.**GetInputs**(short brick)
	Delphi:	Longint_variable := Object_variable.**Inputs**(brick as Smallint)
	Parameter:	brick Short Integer. Represents the number of the brick where the inputs reside. Range is 0-3. Brick 0 represents the onboard inputs. Bricks 1-3 represent expansion I/O bricks 1-3.
	Return Type:	Long Integer. The value represents the current state of the inputs (TIN) for the specified brick.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
IPAddress	Description:	The IPAddress property returns the controller’s IP Address (NTADDR).
	Visual Basic:	object.**IPAddress** As Long
	Visual C++:	long object.**GetIPAddress**()
	Delphi:	Longint_variable := Object_variable.**IPAddress**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the controller’s IP Address (NTADDR).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
Limits	Description:	The Limits property returns the current limit status (TLIM).
	Visual Basic:	object.**Limits** As Long
	Visual C++:	long object.**GetLimits**()
	Delphi:	Longint_variable := Object_variable.**Limits**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current limit status (TLIM).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
MotorPos	Description:	The MotorPos property returns the current commanded position (TPC).
	Visual Basic:	object.**MotorPos** As Long
	Visual C++:	long object.**GetMotorPos**()
	Delphi:	Longint_variable := Object_variable.**MotorPos**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current commanded position (TPC) in counts.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

MotorVel	Description:	The MotorVel property returns the current commanded motor velocity (TVEL).
	Visual Basic:	`object.`**`MotorVel`**` As Long`
	Visual C++:	`long object.`**`GetMotorVel`**`()`
	Delphi:	`Longint_variable := Object_variable.`**`MotorVel`**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current commanded velocity (TVEL) in counts.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
Outputs (brick)	Description:	The Outputs property returns the state of the outputs (TOUT) on the specified brick.
	Visual Basic:	`object.`**`Outputs`**`(brick As Integer) As Long`
	Visual C++:	`long object.`**`GetOutputs`**`(short brick)`
	Delphi:	`Longint_variable := Object_variable.`**`Outputs`**`(brick as Smallint)`
	Parameter:	brick — Short Integer. Represents the number of the brick where the outputs reside. Range is 0-3. Brick 0 represents the onboard outputs. Bricks 1-3 represent expansion I/O bricks 1-3.
	Return Type:	Long Integer. The value represents the state of the outputs (TOUT) on the specified brick.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
SettlingTime	Description:	The SettlingTime property returns the time it took for the last move to settle within into the target zone (TSTLT).
	Visual Basic:	`object.`**`SettlingTime`**
	Visual C++:	`long object.`**`GetSettlingTime`**`()`
	Delphi:	`Longint_variable := Object_variable.`**`SettlingTime`**
	Parameter:	NONE
	Return Type:	Short Integer. The value represents the number of milliseconds it took for the last move to settle within the target zone.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
SystemStatus	Description:	The SystemStatus property returns the system status (TSS) for task 0 only.
	Visual Basic:	`object.`**`SystemStatus`**` As Long`
	Visual C++:	`long object.`**`GetSystemStatus`**`()`
	Delphi:	`Longint_variable := Object_variable.`**`SystemStatus`**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the system status (TSS) for task 0 only.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

Timer	Description:	The Timer property returns the current Timer value (TTIM) for task 0 only.
	Visual Basic:	object.**Timer** As Long
	Visual C++:	long object.**GetTimer**()
	Delphi:	Longint_variable := Object_variable.**Timer**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current Timer value (TTIM) for task 0 only.
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
Triggers	Description:	The Triggers property returns the Trigger Interrupt Status (TTRIG).
	Visual Basic:	object.**Triggers** As Long
	Visual C++:	long object.**GetTriggers**()
	Delphi:	Longint_variable := Object_variable.**Triggers**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current state of the Trigger Interrupt Status (TTRIG).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
UserStatus	Description:	The UserStatus property returns the current state of the user status register (TUS).
	Visual Basic:	object.**UserStatus** As Long
	Visual C++:	long object.**GetUserStatus**()
	Delphi:	Longint_variable := Object_variable.**UserStatus**
	Parameter:	NONE
	Return Type:	Long Integer. The value represents the current state of the user status register (TUS).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.
Var (varnum)	Description:	The Var property returns the value of the specified real variable (VAR). Variables VAR1 through VAR12 may be reported. **NOTE:** This property must be used in conjunction with the Connect2 method (1Mode=2).
	Visual Basic:	object.**Var**(varnum As Integer) As Double
	Visual C++:	double object.**GetVar**(short varnum)
	Delphi:	double_variable := Object_variable.**Var**(varnum as Smallint)
	Parameter:	varnum — Short Integer Represents number of the real variable (VARvarnum). Range is 1-12.
	Return Type:	Double. The value represents the value of the specified real variable (VAR). The initial value is zero until an Extended Fast Status packet arrives.
	Remarks:	This property is valid only with the first client connection to Gem6K, when connected using Connect2 method with 1Mode=2. Read Only. Requires FastStatus to be enabled, or use of RequestFastStatusUpdate or NTSFS command, or generation of an Alarm in the Gem6K.

VarB (varnum)	Description:	The VarB property returns the value of the specified binary variable (VARB).
	Visual Basic:	`object.`**`VarB`**`(varnum As Integer) As Long`
	Visual C++:	`long object.`**`GetVarB`**`(short varnum)`
	Delphi:	`Longint_variable := Object_variable.`**`VarB`**`(varnum as Smallint)`
	Parameter:	varnum Short Integer. Represents number of the binary variable (VARBvarnum). Range is 1-10.
	Return Type:	Long Integer. The value represents the value of the specified binary variable (VARB).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

VarI (varnum)	Description:	The VarI property returns the value of the specified integer variable (VARI).
	Visual Basic:	`object.`**`VarI`**`(varnum As Integer) As Long`
	Visual C++:	`long object.`**`GetVarI`**`(short varnum)`
	Delphi:	`Longint_variable := Object_variable.`**`VarI`**`(varnum as Smallint)`
	Parameter:	varnum Short Integer. Represents number of the binary variable (VARIvarnum). Range is 1-10.
	Return Type:	Long Integer. The value represents the value of the specified integer variable (VARI).
	Remarks:	This is a read-only property. This property requires fast status to be enabled with FSEnabled property.

COM6SRVR Error Codes

Error Code	Description
-1	Bad Ethernet connection due to socket error
-2	Ethernet connection was shut down
-3	Connection attempt failed
-4	Maximum number of Ethernet connections exceeded
-5	Ethernet or RS232 connection not yet established
-6	No filename specified
-7	Unable to locate specified file
-8	Unable to open specified file
-9	Unable to ping Ethernet connection
-10	Unable to create Ethernet socket
-11	Invalid parameter passed to function
-12	Unable to create or connect Ethernet watchdog socket
-13	Unable to create or connect Ethernet fast status socket
-14	Unable to create or connect Ethernet alarm socket
-15	Unable to create or connect Ethernet command socket
-16	Unable to create client ring buffer for Ethernet command socket
-17	SetWatchdog returns this error when Windows runs out of timers.
-18	Unable to write due to XOFF condition (Gemini server only)
-19	WriteBlocking timed out
-20	SendFileBlocking was canceled

COM6SRVR Programming Notes

Calls to COM6SRVR

All calls to the COM6SRVR are blocking calls. Programming control does not return to the client program until the requested operation has been completed in the COM6SRVR. During the call, the Windows operating system continues to process other system calls and timer messages.

Be careful to avoid multiple, simultaneous calls to the server from within the same process. This situation typically arises if there are multiple timer messages being processed by the client application. Because of the nature of COM design, an error would be generated if a timer message initiates a request to the server while another server request from the same client is already in progress.

How to Include the COM6SRVR in a Visual Basic application

All of the Visual Basic samples use a technique known as late binding to interface with the COM6SRVR. The COM6SRVR is not linked at compile time; rather, the link is created dynamically at run time. Unlike an OCX control that needs to be added to a VB form, the COM6SRVR does not require to be added to a form. Creating and using the COM6SRVR is all performed in software. Creating the COM6SRVR is performed with the `CreateObject` VB function.

How to upgrade a Visual Basic application to use the latest COM6SRVR

You should create your Visual Basic application using the late binding technique to interface with the COM6SRVR (see above). The benefit, here, is that whenever a new version of the COM6SRVR is available or installed on the PC, all applications written in Visual Basic with the late binding technique should continue to run satisfactorily.

The latest COM6SRVR may be downloaded from the Support portion of the Compumotor web site at http://www.compumotor.com.

How to include the COM6SRVR in a Visual C++ application

The process below demonstrates how to create a minimal dialog-based application that includes the Com6srvr. Refer also to the sample applications installed in the Motion Planner\Samples\Vc5 directory. (**NOTE:** The samples are installed only if you use the "custom" installation option.)

1. Using Visual C++ AppWizard …
   a. Create an MFC Exe application.
   b. Select the "Dialog based" option.
   c. In the Wizard steps it is not necessary to select the "ActiveX" or "Automation" check boxes.
2. When the application is created, include the afxole.h and afxdisp.h header files. These header files add libraries required for Ole Automation and the COleDispatchDriver class.
3. Initialize the Ole libraries with a call to AfxOleInit. (Refer to sample applications.)
4. Using Visual C++ Class Wizard …
   a. Click the "Add Class…" button and select "From a Type Library" from the drop-down menu.
   b. In the "Import From Type Library" dialog, locate the Com6srvr.tlb file and click the Open button. (The Com6srvr.tlb file is included on the Motion Planner CD-ROM. Recommendation: Copy this file to your project directory.)
   c. Select the relevant interface class ("INet" for Gem6K Ethernet, "IRS232" for Gem6K RS232, or "IGemini" for the Gemini drives). Once the class is imported, the wizard creates two new files and adds them to the project: Com6srvr.cpp and Com6srvr.h.
5. Add a class instance for the interface class you selected in Step 4.c. above (INet, IRS232 or IGemini). Refer to the m_NetServer variable in the VC_Ethernet sample.
6. To understand how to use the libraries, study the sample applications. Pay particular attention to the code related to the Connect, Write and Read methods.

How do I rebuild my Visual C++ project with the new COM6SRVR?

If the COM6SRVR "Interface" changes (because Compumotor has added new properties or methods), it is necessary to re-build your VC++ application and link in the new COM6SRVR to take advantage of any of the new features. Use this step-by-step procedure:

1. Make a backup of your project.
2. Make a backup of your current Com6srvr.exe file (typically located in the Windows\System\ directory or in the Program Files\Compumotor\Motion Planner\ directory).
3. Replace the Com6srvr.exe by overwriting the existing version with the new one from the CD-ROM. Register the new Com6srvr by executing the Com6srvr.exe once. The mouse pointer may change to an hourglass for a couple of seconds while it's being registered.
4. Start VC++ and open the Project Workspace (File > Open Workspace).
5. Use menu item View > Workspace and select the File View tab.
6. Expand the Source Files node and highlight the Com6srvr.cpp file. Remove the file from the project by pressing the DEL key.
7. Expand the Header Files node in the Workspace window. Highlight the Com6srvr.h file and remove it from the project by pressing the DEL key.
8. Save the Workspace (File > Save Workspace) and exit VC++.
9. Start Windows Explorer and locate the project directory.
10. Delete the Com6srvr.cpp, Com6srvr.h and Com6srvr.tlb files.
11. Copy the new Com6srvr.tlb from the CD-ROM into the project directory.
12. Locate the project's Class Wizard database file (.clw file) and delete it.
13. Start VC++ and Open the project Workspace.
14. Note that the COM6SRVR Interfaces (IGemini, IRS232 or INet) no longer appear in the Workspace Window's Class View.
15. Run the class wizard (View > Class Wizard).
16. A dialog appears stating that the class wizard database file does not exist and prompts you to re-build it from your source files. Click the YES button.
17. A "Select Source Files" dialog appears. Click OK.
18. The MFC Class Wizard dialog now appears. Click the Add Class button and select "From a type library" on the drop-down menu.
19. Locate the Com6srvr.tlb file and click the Open button.
20. A "Confirm Classes" dialog appears. All classes (IRS232, INet and Igemini) are highlighted. Select only the classes that are of particular interest. Accept the recommended Header and Implementation File names of Com6srvr.h and Com6srvr.cpp. Click the OK button.
21. Back at the MFC Class Wizard, click OK button.
22. Rebuild the complete project (Build > Rebuild All).
23. Save the Workspace (File > Save Workspace).

Overview: VB5 Samples on the CD-ROM

The VB5 samples are installed if you select Custom installation when installing Motion Planner. The samples are installed on the hard disk in the …\Motion Planner\Samples\VB5 directory. Each sample has its own sub-directory.

The VB5 samples are located on the CD-ROM in the directory Gem6K\Samples\VB5. Again, each sample has its own sub-directory. To copy the files from the CD-ROM use Windows Explorer to copy the sub-directory and its contents to a new location on your hard disk. NOTE, however, that the file attributes will be set to read-only, because the CD-ROM is read-only media. To change the attributes: Using Windows Explorer, locate the sub-directory. Open the sub-directory such that all files are now visible. Use the Edit > Select All menu to select all files. Then, use the menu File > Properties, the dialog shows a Read-Only box with a check mark against it. Uncheck the box and click OK. Now it is possible to open the files and save them in Visual Basic. When using Motion Planner's Custom install to install the samples, the read-only attribute is automatically reset.

There are four VB5 Samples:

- SimpleOne

- SimpleOnePlus
- Terminal
- Gem6K_Capture_Sample

To become familiar with the COM6SRVR and how it is used in Visual Basic, review the SimpleOne application. This sample demonstrates the absolute basics. It deals with creating an instance of the COM6SRVR using Visual Basic's `CreateObject` function. Then it demonstrates how to make a connection to the Gem6K and how to close or disconnect a connection. The main form also has two buttons, "Send Command" and "Read Response," to demonstrate the Write and Read methods of the COM6SRVR. The "Easy Get Data" and "Get Data" buttons demonstrate two techniques to get data from the Ethernet Fast Status.

The Terminal application builds upon the basics introduced by the SimpleOne application. It creates a Terminal to communicate with the Gem6K via Ethernet or RS-232. Buttons to demonstrate the `SendFile`, `GetFile` and `SendOS` methods are included. The Fast Status and Alarms are demonstrated in the form frmFastStatus.

Gem6K_Capture_Sample is a utility as much as a sample. It can be used to capture fast status packets from the Gem6K and store the data for later use.

Overview: Using Fast Status

Fast Status is a tool you can use to keep the COM6SRVR informed of conditions within the Gem6K (conditions such as the values of motor position, encoder position, axis status, and the current value of some of the integer and binary variables). A client application can interrogate the COM6SRVR's fast status data to check various Gem6K conditions (input states, variable values, system conditions, etc.) in two ways:

- Interrogate individual fast status elements via their respective properties. This is the easiest to implement.
- Interrogate the entire fast status data structure. This is more complex, is advantageous when you need to check many data elements at one time.

Interrogating Individual Fast Status Elements

The fast status data structure comprises many elements. Each element may be interrogated through the use of its respective property. The table below lists each fast status data element and its respective property.

Fast Status — Packet Data Definition (280 bytes total)
***ExFastStatus — Packet Data Definition (376 bytes total)**

Type	**Description**	**Bytes**	**Property** (see pages 24-32 and 44-53)
WORD wUpdateID	Unique update ID for synch channel	2	*No property available*
WORD wCounter	Time Frame Counter	2	**Counter**
DWORD dwMotorPos[8]	Commanded Position (`TPC`)	32	**MotorPos**
DWORD dwEncPos[8]	Encoder Position (`TPE`)	32	**EncoderPos**
DWORD dwMotorVel[8]	Commanded Velocity (`TVEL`)	32	**MotorVel**
DWORD dwAxisStatus[8]	Axis Status (`TAS`)	32	**AxisStatus**
DWORD dwSystemStatus	System Status (`TSS`)	4	**SystemStatus**
DWORD dwErrorStatus	Error Status (`TER`)	4	**ErrorStatus**
DWORD dwUserStatus	User Status (`TUS`)	4	**UserStatus**
DWORD dwTimer	Timer (`TTIM`)	4	**Timer**
DWORD dwLimits	Limit Status (`TLIM`)	4	**Limits**
DWORD dwInputs[4]	Input Status (`TIN`)	16	**Inputs**
DWORD dwOutputs[4]	Output Status (`TOUT`)	16	**Outputs**
DWORD dwTriggers	Trigger Status (`TTRIG`)	4	**Triggers**
WORD wAnalogIn[2]	Analog Input Value (`TANI` - in counts)	4	**AnalogInput**
DWORD dwVarb[10]	Binary Variable Values (`VARB1` – `VARB10`)	40	**VarB**
DWORD dwVari[10]	Integer Variable Values (`VARI1` – `VARI10`)	40	**VarI**
DWORD dwIPAddress	IP Address (`NTADDR`)	4	**IPAddress**
DWORD dwCmdCount	Command Count	4	**CommandCount**
*DWORD dwVar[12]	Real Variable Values (`VAR1 - VAR12`)	96	**Var**

NOTE: Each call to the COM6SRVR incurs overhead; therefore, you should be aware that interrogating many different fast status elements at one time is slower than interrogating the entire data structure (see below).

Interrogating the Entire Fast Status Structure

Accessing data in the fast status structure is more complex than interrogating individual elements, but it has the advantage that only one call to the COM6SRVR is required to access the entire fast status structure — this reduces overhead when multiple data elements are required.

The COM6SRVR FastStatus property (see pages **27** and **47**) returns a VARIANT data type, which is actually an array of bytes. The array of bytes is copied into a structure. Ordering the array of bytes and structure elements is very important to ensure that the correct data bytes are copied into the correct structure elements. (Do not change the fast status structure or TypeDef.) In Visual Basic, use the Windows API CopyMemory function to copy bytes into the structure; in Visual C++, the copying is performed through use of SAFEARRAYS (refer to the MakeFastStatus function in the VC_Ethernet sample provided).

There are two techniques for updating the Fast Status: Streaming and On Demand.

- The Streaming technique is particularly useful for HMI type applications where a constant stream of data at a set interval is required. To use streaming, set a streaming interval with the FSUpdateRate property (see pages **28** and **49**) and enable streaming by setting the FSEnabled property to TRUE (see pages **28** and **49**).
- The On Demand technique can reduce network traffic by sending fast status packets only when required, rather than at a pre-defined interval. The fast status packet can be updated by a call to the COM6SRVR RequestFastStatusUpdate method (see pages **16** and **36**), or under Gem6K program control with the Gem6K command `NTSFS`. Similarly, when a Gem6K generates an alarm, such as `INTSW1`, a fast status packet (280 bytes) is automatically sent to the COM6SRVR. Note that the On Demand technique operates independent of the FSEnabled property and the FSUpdateRate property. The sample VB program called SimpleOnePlus demonstrates the use of RequestFastStatusUpdate.

Customizing Fast Status

The Fast Status structure (see page **57**) provides most of the frequently required system parameters (e.g., motor position and axis status). However, there are times when a specific status condition or parameter is required, but not provided by default. One such example might be Following Status (`TFS`).

The Fast Status packet always includes ten integer variables (`VARI1` – `VARI10`) and ten binary variables (`VARB1` – `VARB10`). This allows you to customize part of the Fast Status packet by copying status conditions of interest into the `VARI` or `VARB` variables. Below are two scenarios that demonstrate two methods of using variables to customize the Fast Status.

Scenario 1: An HMI application must inform the operator of the total number of products made, the number of products that meet specification (passes), and number of products that fail to meet specification. In this type of scenario, the Total Number is assigned to `VARI1`, the number of Passes to `VARI2`, and the number of failures to `VARI3`. These variables are then updated as needed. For example, when a product is made, `VARI1` is incremented. This insures the data is automatically updated in the Fast Status packet; then the HMI application can use the VARI(1) property to interrogate the `VARI1` value.

```
DEL PLCP1
DEF PLCP1
  VARI1 = VARI1 + 1
  VARB1 = 1FS
END

PCOMP PLCP1

VARI1=0

SCANP PLCP1
```

Scenario 2: Another method to maintain the information is to use a `PLCP` program, launched in the Scan Mode with the `SCANP` command. The PLCP program updates the variable (`VARI` or `VARB`). In this manner, the parameter of interest can be mapped to a specific variable. Example: Use `VARB` to allow monitoring of Following Status (`TFS`) and allow `VARI1` to auto-increment (see code at left).

Then use the “ActiveXFastPanel” in Motion Planner’s Panel Maker to monitor the “Variables” grid. You’ll notice that `VARI1` is continually incrementing and that as you enable and disable Following on axis 1 (`FOLEN1` / `FOLEN0`), bit 6 of `VARB1` is set and cleared.

Overview: Using Variable Packets

Using variable packets, you can quickly and efficiently transfer large amounts of data from the COM6SRVR to the Gem6K. The variables packet comprises: integer variables 1-12 (VARI1 – VARI12), real variables 1-12 (VAR1 – VAR12), and binary variables 1-8 (VARB1 – VARB8).

Variable packets can be sent by one of two COM6SRVR methods:

- The SendVariable method (see pagesx **18** and **38**) allows transmission of one variable. To use SendVariable, you define: (a) a mask to specify which variable to update, and (b) the value of the variable. Mask bits for Visual Basic and Visual C++ are provided below.
- The SendVariablePacket method (see pages **20** and **40**) allows one or all variables to be sent in one packet. The SendVariablePacket method requires a SendVariable structure to be populated, copied into a variant and then pass the variant to the SendVariablePacket Method. The SendVariable structure comprises several elements: a Mask, several reserved elements and an array of twelve elements for integer variables, an array of twelve elements for real variables and an array of eight elements for binary variables (see structure lists below). Several mask constants can be Or'ed together to allow Gem6K variables to be updated (see mast lists below).

TIP: Using the a variable packet method (SendVariable or SendVariablePacket) and an On Demand Fast Status interrogation technique (RequestFastStatusUpdate method or the Gem6K command `NTSFS`) provides a clean and efficient communication tool between the client application and the Gem6K program. Although variables can be sent as a command using the Write method, the time taken to parse the command and check data validity is longer than the time to send an entire variable packet.

A sample VB program called SimpleOnePlus is provided; it demonstrates the use of the SendVariable and SendVariablePacket COM6SRVR methods.

Variable Structure for Visual Basic

```
Type SendVariableStructure
  Mask As Long
  Reserved1 As Long
  Reserved2 As Long
  Reserved3 As Long
  VarI(1 To 12) As Long
  VarR(1 To 12) As Double
  VarB(1 To 8) As Long
End Type
```

Variable Structure for Visual C++

```
typedef struct VARIABLEPACKETStruct {
 int nVariableMask;
 int nReserved1;
 int nReserved2;
 int nReserved3;
 int VARI[12];
 double VAR[12];
 int VARB[8];
} VARIABLEPACKET, *LPVARIABLEPACKET;
```

Variable Packet Mask Bits for Visual Basic

```
Public Const VARI1 As Long = 1
Public Const VARI2 As Long = 2
Public Const VARI3 As Long = 4
Public Const VARI4 As Long = 8
Public Const VARI5 As Long = 16
Public Const VARI6 As Long = 32
Public Const VARI7 As Long = 64
Public Const VARI8 As Long = 128
Public Const VARI9 As Long = 256
Public Const VARI10 As Long = 512
Public Const VARI11 As Long = 1024
Public Const VARI12 As Long = 2048

Public Const VAR1 As Long = 4096
Public Const VAR2 As Long = 8192
Public Const VAR3 As Long = 16384
Public Const VAR4 As Long = 32768
Public Const VAR5 As Long = 65536
Public Const VAR6 As Long = 131072
Public Const VAR7 As Long = 262144
Public Const VAR8 As Long = 524288
Public Const VAR9 As Long = 1048576
Public Const VAR10 As Long = 2097152
Public Const VAR11 As Long = 4194304
Public Const VAR12 As Long = 8388608

Public Const VARB1 As Long = 16777216
Public Const VARB2 As Long = 33554432
Public Const VARB3 As Long = 67108864
Public Const VARB4 As Long = 134217728
Public Const VARB5 As Long = 268435456
Public Const VARB6 As Long = 536870912
Public Const VARB7 As Long = 1073741824
Public Const VARB8 As Long = &H80000000
```

Variable Packet Mask Bits for Visual C++

```
#define VARI1  0x00000001
#define VARI2  0x00000002
#define VARI3  0x00000004
#define VARI4  0x00000008
#define VARI5  0x00000010
#define VARI6  0x00000020
#define VARI7  0x00000040
#define VARI8  0x00000080
#define VARI9  0x00000100
#define VARI10 0x00000200
#define VARI11 0x00000400
#define VARI12 0x00000800

#define VAR1   0x00001000
#define VAR2   0x00002000
#define VAR3   0x00004000
#define VAR4   0x00008000
#define VAR5   0x00010000
#define VAR6   0x00020000
#define VAR7   0x00040000
#define VAR8   0x00080000
#define VAR9   0x00100000
#define VAR10  0x00200000
#define VAR11  0x00400000
#define VAR12  0x00800000

#define VARB1  0x01000000
#define VARB2  0x02000000
#define VARB3  0x04000000
#define VARB4  0x08000000
#define VARB5  0x10000000
#define VARB6  0x20000000
#define VARB7  0x40000000
#define VARB8  0x80000000
```