ACR Motion Controllers

88-030044-01A

ComACRServer6 User's Guide

Effective: October 2009

ENGINEERING YOUR SUCCESS.

User Information

Warning — ACR 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.

North America and Asia
Parker Hannifin Corporation
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9068 or (707) 584-7558
Fax: (707) 584-3793
Email: emn_support@parker.com
Internet: http://www.parkermotion.com

Germany, Austria, Switzerland
Parker Hannifin GmbH&Co.KG
Postfach: 77607-1720
Robert-Bosch-Str. 22
D-77656 Offenburg
Telephone: +49 (0) 781 509-0
Fax: +49 (0) 781 509-98176
Email: sales.hauser@parker.com
Internet: http://www.parker-emd.com

Europe (non-German speaking)
Parker Hannifin plc
Electromechanical Automation, Europe
Arena Business Centre
Holy Rood Close
Poole
Dorset, UK
BH17 7BA
Telephone: +44 (0) 1202 606300
Fax: +44 (0) 1202 606301
Email: support.digiplan@parker.com
Internet: http://www.parker-emd.com

Italy
Parker Hannifin
20092 Cinisello Balsamo
Milan, Italy via Gounod, 1
Telephone: +39 02 66012478
Fax: +39 02 66012808
Email: sales.sbc@parker.com
Internet: http://www.parker-emd.com

Parker Hannifin Technical Support E-mail: **emn_support@parker.com**

Table of Contents

Communications Server

The communications server, ComACRServer.exe, is a 32-bit OLE automation server that provides communications between ACR controllers and PC (personal computer) software applications. It is compatible with any 32-bit software application or programming environment that uses an OLE automation component, including the following:

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

The ACR-View 6.x installation program installs the ComACRServer.exe file in the ACR-View 6.0 directory, typically \Program Files\Parker\ACR-View 6.0.

Supported Controllers

The ComACRServer.exe communications server (comserver) supports the following ACR family controllers using available Ethernet, USB, and RS232 communication ports:

- ACR9000
- ACR9030
- ACR9040
- ACR9600
- ACR9630
- ACR9640
- Aries Controller (AR-xxCE)

Note: PC card (1505, 8020) controllers are not supported by the ComACRServer.exe.

Legacy Communications Server: ComACRsrvr.dll

The ComACRServer.exe is based on the previous generation of ACR communications server known as the ComACRsrvr.dll. Significant changes to the underlying architecture result in increased performance and robustness. The format of properties and methods available in the ComACRsrvr.dll has been retained to allow an easy transition for existing users.

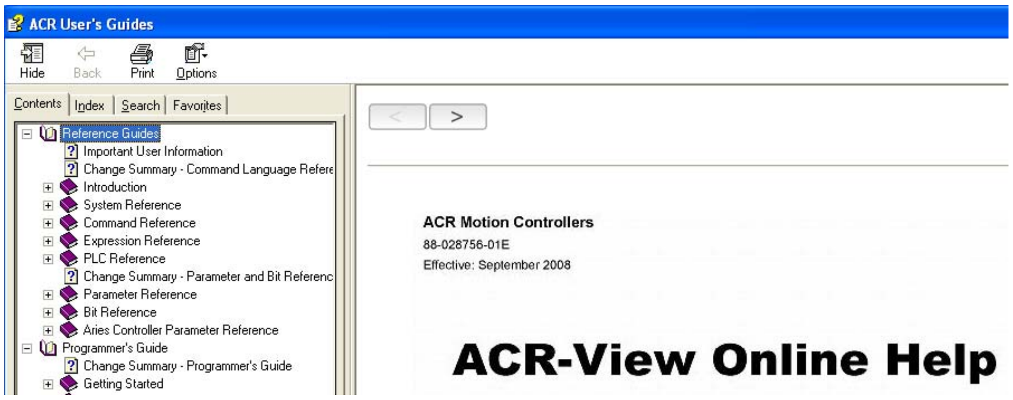
Getting Started

Before developing a ComACRServer application, users should become familiar with the ACR controller by using ACR-View, the development software for the controllers.

ACR-View

- Verify PC to controller communication.
- Configure the controller (how many axes, stepper or servo, scaling, etc.).
- Tune servo axes.
- Test axis connections and I/O functions (e.g., limit and home switches).
- Develop any programs to be stored on the controller.
- Monitor values as they are sent and received from the ComACRServer application.

ACR-View Online Help contains information about AcroBasic commands, the controller's native language. Most of these commands can be sent thru the ComACRServer using the **Write** method. The bit and parameter reference describes all of the available data elements on the controller, which can be accessed thru the **Control** and **Status** methods. Figure 1 shows the reference and programmer's guides available in the online help.



ACR Motion Controllers
88-028756-01E
Effective: September 2008

ACR-View Online Help

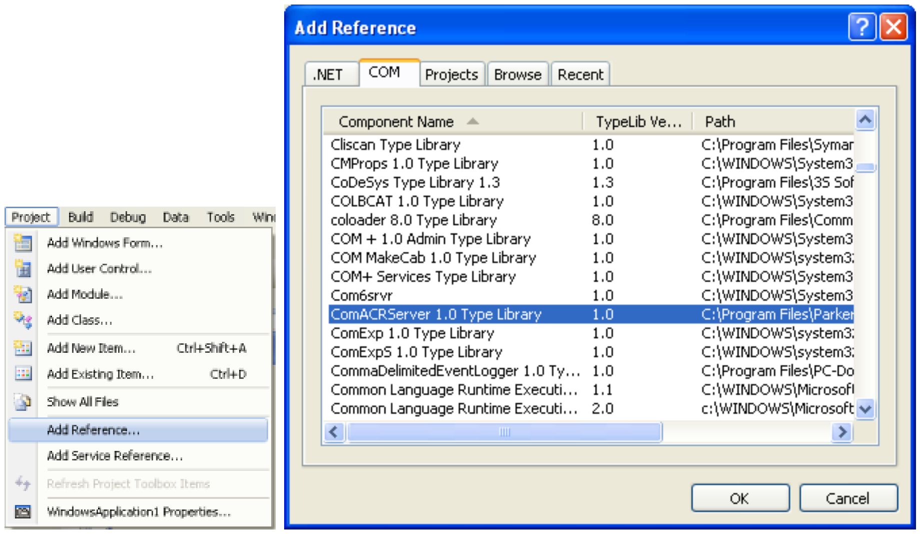
Figure 1: ACR-View Online Help

Installation

The ComACRServer.exe is installed and registered with ACR-View 6. The default installation folder for ACR-View and ComACRServer.exe is C:\Program Files\Parker\ACR-View 6.

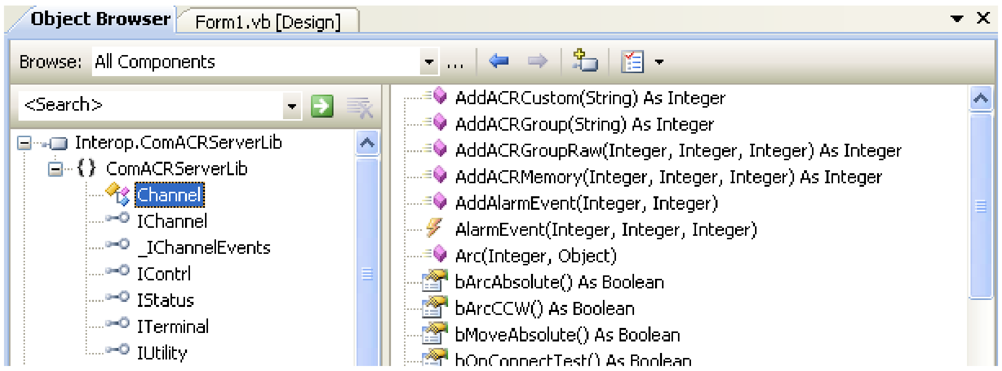
Creating a Project with the ComACRServer

To create a project, select Project on the Visual Studio main menu, and then select Add Reference. In the Add Reference dialog box, select ComACRServer x.x Type Library, and click OK. Figure 2 illustrates these steps.



Component Name	TypeLib Ve...	Path
Cliscan Type Library	1.0	C:\Program Files\Symar
CMProps 1.0 Type Library	1.0	C:\WINDOWS\System3
CoDeSys Type Library 1.3	1.3	C:\Program Files\3S Sof
COLBCAT 1.0 Type Library	1.0	C:\WINDOWS\System3
coloader 8.0 Type Library	8.0	C:\Program Files\Comm
COM + 1.0 Admin Type Library	1.0	C:\WINDOWS\System3
COM MakeCab 1.0 Type Library	1.0	C:\WINDOWS\system3:
COM+ Services Type Library	1.0	C:\WINDOWS\System3
Com6srvr	1.0	C:\WINDOWS\System3
ComACRServer 1.0 Type Library	1.0	C:\Program Files\Parker
ComExp 1.0 Type Library	1.0	C:\WINDOWS\system3:
ComExpS 1.0 Type Library	1.0	C:\WINDOWS\system3:
CommaDelimitedEventLogger 1.0 Ty...	1.0	C:\Program Files\PC-Do
Common Language Runtime Executi...	1.1	C:\WINDOWS\Microsof
Common Language Runtime Executi...	2.0	c:\WINDOWS\Microsoft

Figure 2: ComACRServer Project Menu: Add Reference


Object Browser | Form1.vb [Design]

Browse: All Components

<Search>

```
Interop.ComACRServerLib
  {} ComACRServerLib
      Channel
      IChannel
      _IChannelEvents
      IContrl
      IStatus
      ITerminal
      IUtility
```

```
AddACRCustom(String) As Integer
AddACRGroup(String) As Integer
AddACRGroupRaw(Integer, Integer, Integer) As Integer
AddACRMemory(Integer, Integer, Integer) As Integer
AddAlarmEvent(Integer, Integer)
AlarmEvent(Integer, Integer, Integer)
Arc(Integer, Object)
bArcAbsolute() As Boolean
bArcCCW() As Boolean
bMoveAbsolute() As Boolean
bOnConnectTest() As Boolean
```

Figure 3: ComACRServerLib Channel Class

Classes and Interfaces

The ComACRServerLib contains all the available properties and methods in a single class called Channel (See Figure 3). A single object can be declared to access all the properties and methods.

```
 Dim WithEvents Controller As ComACRServerLib.Channel
```

The previous version of the comserver (BoxBridgeLib) separated the properties and methods into four separate classes, Terminal, Status, Control, and Utility. This required declaring and connecting to up to four separate objects to access all the functionality of the comserver.

```
Dim cntl As BOXBRIDGELib.Control
Dim WithEvents stat As BOXBRIDGELib.Status
Dim WithEvents term As BOXBRIDGELib.Terminal
Dim util As BOXBRIDGELib.Utility
```

In order to minimize development time when upgrading projects developed using the BoxBridgeLib, the ComACRServerLib includes interfaces that group properties and methods in a similar way.

```
    Dim cntl As ComACRServerLib.IContrl
    Dim stat As ComACRServerLib.IStatus
    Dim term As ComACRServerLib.ITerminal
    Dim util As ComACRServerLib.IUtility

'Create new object for each Interface
        cntl = New ComACRServerLib.Channel
        stat = New ComACRServerLib.Channel
        term = New ComACRServerLib.Channel
        util = New ComACRServerLib.Channel

'Requires a call to Connect for each Interface
        cntl.bstrIP = “192.168.10.40”
        stat.bstrIP = “192.168.10.40”
        term.bstrIP = "192.168.10.40"
        util.bstrIP = "192.168.10.40"
        cntl.Connect(3, 0)
        stat.Connect(3, 0)
        term.Connect(3, 0)
        util.Connect(3, 0)
```

Alternatively, create only one Channel object and reference the other interfaces. This allows the application to call **Connect** only one time.

```
'Create new object for each Interface
        cntl = New ComACRServerLib.Channel
        stat = cntl
        term = cntl
        util = cntl

'Connect
        cntl.bstrIP = "192.168.10.40"
        cntl.Connect(3, 0)
```

Events are only available in the Channel interface:

```
    Dim WithEvents stat As ComACRServerLib.Channel
    Dim WithEvents term As ComACRServerLib.Channel
```

Communications Overview

To begin communications, an application requests a connection to the ACR controller through the comserver. The comserver manages the actual connection to each controller, and can feed information from a particular controller to all client applications that require the information.

The comserver makes one connection for each ACR controller per communication media (Ethernet, USB, or serial, depending on the communication options). This connection is then shared across all client users of that connection/controller pair. For example, a terminal application created in Visual Basic and a terminal in ACR-View can maintain connections to the same ACR controller's USB connection. Both applications receive the responses coming from the controller and do not compete for data.

Alternatively, a status application created in C++ can use the comserver to get information from one ACR controller while ACR-View uses the comserver to talk to a different ACR controller.

User Guide Examples

Programming examples are provided at the end of each chapter. Examples may contain only the portion of the code needed to illustrate the functionality. All examples are written in VB.NET. For simplicity, the examples do not include exceptions handling, which is recommended for most applications. Complete example applications in VB.NET, C#, C++, and VB6 are provided on the ACR-View CD and can be downloaded from www.parkermotion.com.

Connection Properties and Methods

For an application to communicate with an ACR controller, the application must first call the **Connect**() method. When calling the **Connect**() method, the comserver performs two steps.

The comserver establishes a connection with the physical medium (serial, Ethernet, etc.) using the device drivers on the system to secure the communication resource (ports, sockets, etc.) needed to talk to the connected ACR controller. This is only performed by the very first Channel to call the **Connect**() method. Subsequent calls to the **Connect**() method do not perform this step. Instead, subsequent Channels register themselves as users. When the last user closes down, the resources are freed.

By default, the comserver tries to verify that an ACR controller is connected to the PC. To do this, the comserver sends information to the ACR controller and inspects the reply.

This behavior can be disabled, and the check not performed. Prior to calling the **Connect**() method, set the **bOnConnectTest** to False. For example, do this when you know the ACR controller is present.

Alternatively, call the **TestConnect**() method to test, at any time, if a connection is present.

After successfully executing the **Connect**() method, all other methods on that interface are available.

The following properties and methods are used to connect to a controller. Descriptions of each one follow the list:

- Property **nPort**Error! Bookmark not defined.() As Integer
- Property **bstrIP**Error! Bookmark not defined.() As String
- Property **nBPS**() As Integer
- Property **bOnConnectTest**() As Boolean
- ReadOnly Property **isOffline**() As Boolean
- **Connect** (ByVal *nTransport* As Integer, ByVal *nIndex* As Integer)
- Sub **Disconnect**()
- **TestConnect**() As Boolean
- **SetWatchdog**(ByVal *nInterval* As Integer, ByVal *nRetries* As Integer)

- Event **WatchdogReconnect**()
- Event **WatchdogTimeout**()

Descriptions

Property **nPort**() As Integer

Summary: Returns or sets the communications port on the PC used for connection to the controller

Range: 1–30

Default: 1

Remarks:

- Set this value when using serial or USB communications.

Property **bstrIP**() As String

Summary: The Ethernet IP address of the ACR device in dot notation—xxx.xxx.xxx.xxx

Default: 0.0.0.0

Remarks:

- To communicate over a network using TCP/IP, the network settings must be configured for the PC and ACR device. For more information, see the IP and IP MASK commands for the ACR Series controller.

Property **nBPS**() As Integer

Summary: The speed of the serial port in bits per second (BPS); to set the communications port (COM1, etc.) for serial communications.

Range: 9600, 19200, 38400

Default: 38400

Remarks:

- The comserver can use any BPS rate that is supported by the PC; however, the ACR controllers can only support rates of 9600, 19200, and 38400 BPS.

Property **bOnConnectTest**() As Boolean

Summary: This allows (or disallows) automatic ACR verification as part of **Connect**

Default: True

Remarks:

- When **bOnConnectTest** is set to True, the **Connect** method verifies that an ACR device is physically connected and responding after successfully connecting to a communications transport. This extra verification is done using an implicit call to **TestConnect**. See the TestConnect description for more information. If no controller is found, the **Connect** method throws an exception.
- If **bOnConnectTest** is set to False, some communication transports will not fail when using **Connect**, even when no ACR device is physically connected. For example, a serial port only needs to be present on the PC at the nPort communication port address for **Connect** to return success.
- Some applications may not need to perform this extra check, in which case set this value to False. Setting this value to False will slightly speed up **Connect**.

ReadOnly Property **isOffline**() As Boolean

Summary: Indicates whether the Transport type is set to Offline

Default: True

Remarks:

- If True, the transport type is set to Offline, which indicates that the interface is not connected to an ACR device. See the **Connect** description, which follows, for the transport types.
- After a successful connection, return to Offline mode by either explicitly calling **Connect** with the Offline transport type, or calling **Disconnect**().
- Many methods throw an exception if called in the Offline mode; Check this property first to avoid throwing an exception.
- **isOffline**=False does not indicate that the comserver is communicating with the controller, only that connection is open and trying to connect. Use the **TestConnect** method when trying to determine if the controller is actively communicating and ready to accept commands.

Connect(ByVal *nTransport* As Integer, ByVal *nIndex* As Integer)

Summary: Establishes a connection to an ACR controller

Parameters:

nTransport: Indicates the physical communication layer being used. Valid values are the following:

nTransport	Description
0	Offline
2	Serial
3	Ethernet
4	USB

nIndex: Parameter is not used; provided for compatibility with legacy code.

Remarks:

- Each transport type has its own data requirements for connecting. Any transport-specific properties (i.e., bstrIP for Ethernet, etc.) should be set prior to calling **Connect**.
- After **Connect** is successfully called (except for *nTransport*=Offline), the ACR device is connected and ready.
- To ensure an ACR controller is physically present, set the **bOnConnectTest** property to True (the default) or call the **TestConnect** method once connected.
- If the comserver cannot complete a connection only because it does not receive a response from the device, the following happens:
    - If **bOnConnectTest**=True, an E_PENDING error will be signaled.
    - The comserver will periodically attempt to reconnect to the control and will fire the **WatchdogReconnect** event when it succeeds.
    - **isOffline** will return False.
    - The client must expressly disconnect a Channel after receiving E_PENDING if it no longer wants to connect to the control.
- If a Channel is connected when **Connect** is called, the old Channel will be disconnected before the new Channel is connected.
- Calling **Connect** with *nTransport*=Offline is the equivalent of calling **Disconnect**.

Sub **Disconnect**()

Summary: Disconnects from the current communication transport

Remarks: Implicitly calls **Connect**(`0,0`) to switch to Offline mode

---

TestConnect() As Boolean

Summary: Verifies that an ACR controller is connected

Remarks:

- The **TestConnect** method sends a binary command to the controller and verifies the returned data.
- If this process succeeds, the controller's presence is presumed and True is returned. Otherwise False is returned. When the Transport type = Offline, this method always returns False.
- This is the same test as done by **Connect**() when **bOnConnectTest** is set to True.

SetWatchdog(ByVal *nInterval* As Integer, ByVal *nRetries* As Integer)

Summary: Modifies the ACR controller's Ethernet Watchdog values

Parameters:

nInterval: The time, in milliseconds, between sending test keep-alive strings to the ACR device

nRetries: The number of times the keep-alive test string message is sent to the ACR device, with no valid reply, before attempting to re-connect to the ACR device

Remarks:

- The Ethernet transport currently has Watchdog functionality. The ACR controller uses a separate port to receive a coded command string (keep-alive message).
- If the ACR controller does not receive the keep-alive, a successful command string in *nInterval* * *nRetries* milliseconds, the ACR controller disconnects the regular ACR Ethernet connection it is watching.
- This method has no effect on any transport except Ethernet, and now serves only to provide the heartbeat for the ACR Ethernet watchdog mechanism.
- This setting no longer has an effect on the timing of the comserver's **WatchdogTimeout** event, which occurs when two successive polls 500 milliseconds apart fail.
- The initial settings of the Ethernet Watchdog are as follows:

- *nInterval*=2000 ms (2 seconds)
- *nRetries*=4

Event **WatchdogReconnect**()

Summary: Callback method acts as an event signaling that Ethernet communications has been re-established after watchdog timer previously timed out.

Remarks:

- The COM event model uses a callback mechanism to generate events. The client program implements and registers this method.
- The comserver calls the method after the watchdog timer reconnects to the controller after a previous **WatchdogTimeout** event.

Event **WatchdogTimeout**()

Summary: Callback method acts as an event signaling that the watchdog timer has timed out on an Ethernet connection

Remarks:

- The COM event model uses a callback mechanism to generate events. The client program implements and registers this method.
- The comserver watchdog times out when two successive polls 500 milliseconds apart fail.
- The comserver calls the method after the watchdog timer times out due to a loss of Ethernet communications.

Connection Example

```
Dim WithEvents Controller As ComACRServerLib.Channel
Public Sub New()
    Controller = New ComACRServerLib.Channel
End Sub
Public Sub ConnectEthernet()
    Dim strIP As String
    strIP = "192.168.10.40"
    Controller.bstrIP = strIP
    Controller.Connect(3, 0)
End Sub
Public Sub ConnectUSB()
    Dim USBPort As Integer
    USBPort = 10
    Controller.nPort = USBPort
    Controller.Connect(4, 0)
End Sub
Public Sub ConnectSerial()
    Dim ComPort As Integer
    Dim BaudRate As Integer
    ComPort = 2
```

```
        BaudRate = 38400
        Controller.nBPS = BaudRate
        Controller.nPort = ComPort
        Controller.Connect(2, 0)
    End Sub
    Public Sub CloseConnection()
        If Controller.isOffline = False Then
           Controller.Disconnect()
        End If
    End Sub
```

Terminal Properties and Methods

The following Terminal properties and methods provide functionality for doing simple request/reply with the controller using ASCII characters. Details of each one follow the list. Refer to the ACR User's Guide, Command Reference for information about valid commands and syntax used with the controller. To send ASCII character strings to the ACR controller, use **Write**(). To receive data from the ACR controller, use **Read**(). A COM event, **DataWaiting**(), is included for alerting the client program that data is ready to be read.

- Property **bTerminalLock**() As Boolean
- Property **nDataWaitRate**() As Integer
- Function **Read**() As String
- Sub **Write**(ByVal send As String)
- Event **DataWaiting**()

Descriptions

Property **bTerminalLock**() As Boolean

Summary: Prevents competing accesses to a control's ASCII stream

Default: False

Remarks:

- While **bTerminalLock** is True, all **Writes** from competing Channels connected to the same controller are locked out. They will wait briefly for **bTerminalLock** to be cleared, but then they return with E_PENDING timeout status.
- When an application sends the controller an ASCII request via **Write**() and expects to read the resulting status via **Read**(), it must lock the terminal stream via **bTerminalLock**. Otherwise, a competing Channel can also write data

causing the terminal thread to show interleaved results, making interpretation very difficult or impossible.

Property **nDataWaitRate**() As Integer

Summary: The minimum time between status alerts in milliseconds

Default: 50 ms

Remarks:

- If the read buffer has new data available, a COM event is generated between the time the data comes into the read buffer and **nDataWaitRate** milliseconds past that time. Using this property, you can set the minimum time between events, in milliseconds. The default setting is well below the perceivable time a user would notice a delay, but well above the value that would tax PC resources. Setting this value to zero will disable alerts, and the client software will be required to poll the **Read**() method for data.
- Changes to this value only take effect if set before the **Connect**() method is called.

Read() As String

Summary: Retrieves ASCII data from the controller

Remarks:

- This returns the data in the controller's output buffer.
- Typically, call this method when a **DataWaiting**() event is received.

Write(ByVal *send* As String) As String

Summary: Sends ASCII data to the controller

Remarks:

- Most ACR commands require a carriage return <cr> (hex x0D) to execute.
- ASCII commands will be buffered behind any **Move** or **Arc** commands that are currently waiting in the buffer. Binary commands (see the Control Methods section) are executed immediately ahead of **Moves** and ASCII in the buffer and should be utilized when possible.
- In certain cases (during a file or OS download) the **Write**() method is locked out; subsequently, the function times out. In this instance, characters may not get to the controller.

Event **DataWaiting**()

Summary: Callback method acts as an event signaling that there is data to read

Remarks:

- The COM event model (also know as Connection Points) uses a callback mechanism to generate events. The client program implements and registers this method. The ComACRServer calls the method when there is data in the read buffer, suggesting to the client to call **Read**().
- **DataWaiting** Events are launched on a separate thread. In .NET, the event should not try to directly access a Form Control, otherwise a "Cross-thread operation not valid" error may result. Delegates can be used in this case. See the example below.

Terminal Example

Example 1

```
Dim WithEvents Controller As ComACRServerLib.Channel

   Private Sub SendAxisSettings()
        Dim strSend As String
        strSend = "AXIS0 JOG VEL 10"
        SendASCII(strSend)
        strSend = "AXIS0 JOG ACC 100"
        SendASCII(strSend)
   End Sub
   Private Sub SendASCII(ByVal strSend As String)
        strSend += vbCr    ' add carriage return to end of string
        Controller.Write(strSend)
    End Sub

    Private Function ReadASCII() As String
        Dim strRead As String
        strRead = Controller.Read
        Return strRead
    End Function
```

Example 2

```
' use Delegate to output controller responses to RichTextBox
    Private Delegate Sub DelegateTexttoTerm(ByVal strTerm As String)

    Private Sub TexttoTerm(ByVal strTerm As String)
        RichTextBox1.Text = strTerm
    End Sub

    Private Sub Controller_DataWaiting() Handles Controller.DataWaiting
        Dim textTerm As New DelegateTexttoTerm(AddressOf TexttoTerm)
        Dim strRead As String
            strRead = ReadASCII()
            'RichTextBox1.Text = strRead
'  above code will result in  "Cross-thread operation not valid" error
            RichTextBox1.Invoke(textTerm, strRead)
    End Sub
```

Status Properties and Methods

The ComACRServer provides an efficient method of retrieving controller p-Parameters using ACR Binary syntax. A single value or a related group of 8 values can be read at one time (the Groups of specific p-Parameters are documented in ACR-View online help). The Status methods convert p-Parameters into their binary equivalent commands.

The Status Interface can be used in two ways: in a simple request/reply form or as an event driven alerting request queue. The request/reply method is straightforward. Call a method that begins with **GetACR***, and receive the information requested.

The event driver method requires more steps. First use the **AddACR*** methods to put a request into the queue. Second, wait for the COM event **StatusWaiting**() to fire. When **StatusWaiting**() fires, use **GetStatus**() to get the information. When done watching the status, use **DelStatus**() to remove the status from the request queue.

Properties

- Property **nStatusWaitRate**() As Integer
- Property **SupressStatusDelete**() As Integer
- Property **SendSynchronousEvents**() As Boolean

Methods

- **GetACRCustom**(ByVal *bstrRequest* As String) As Object
- **GetACRGroup**(ByVal *bstrRequest* As String) As Object
- **GetACRMemory**(ByVal *nType* As Integer, ByVal *nAddress* As Integer, ByVal *nCount* As Integer) As Object
- **GetLocalAddr**(ByVal *nProg* As Integer, ByVal *nType* As Integer, ByRef *nSize* As Integer) As Integer
- **GetLocalArrayAddr**(ByVal *nProg* As Integer, ByVal nType As Integer, ByVal *nArray* As Integer, ByRef *nSize* As Integer) As Integer
- **GetParmAddr**(ByVal *nParameter* As Integer) As Integer
- **IsFlagSet**(ByVal *nFlagGrp* As Integer, ByVal *nFlagNdx* As Integer) As Boolean
- **AddACRCustom**(ByVal *bstrRequest* As String) As Integer

- **AddACRGroup**(ByVal *bstrRequest* As String) As Integer
- **AddACRMemory**(ByVal *nType* As Integer, ByVal *nAddress* As Integer, ByVal *nCount* As Integer) As Integer
- **DelStatus**(ByVal *nMsgid* As Integer)
- **GetStatus**(ByVal *nMsgid* As Integer) As Object
- Event **StatusWaiting**(ByVal *msgID* As Integer, ByVal *error* As Integer)

Properties Descriptions

Property **nStatusWaitRate**() As Integer

Summary: The minimum time between status alerts in milliseconds.

Default: 10 ms

Remarks:

- If a status request has new data available, a COM event is generated in the time between when a status changes and nStatusWaitRate milliseconds past the time a status changes. Setting this value to zero will disable alerts. Use this property to set the minimum time in milliseconds between alert events.
- Changes to this value only take effect if set before calling the **Connect**() method.

Property **SupressStatusDelete**() As Integer

Summary: Indicates the action the comserver should take when an attempt to read a Status Queue data item fails

Default: 0

Range: 0–2

0: Fire Error **StatusWaiting** and Delete Entry

1: Fire Error **StatusWaiting** Event only

2: Do Not Fire **StatusWaiting** or Delete

Remarks:

- If a read error occurs when **SupressStatusDelete** is 0, a **StatusWaiting** event is fired with an error value specified in the error parameter. Then the failed status entry is automatically deleted from the queue.
- If a read error occurs when **SupressStatusDelete** is 1, a **StatusWaiting** event is fired with an error value specified in

the error parameter. The failed status entry is not automatically deleted, and the error **StatusWaiting** event will continue to be fired at the poll interval until the outage is rectified or the event deleted through **DelStatus**().

- If a read error occurs when **SupressStatusDelete** is 2, no **StatusWaiting** event is fired. The client application can learn about the outage by fielding **WatchdogTimeout** events.
- Errors reading status queue items typically occur when communications have temporarily been lost. A **WatchdogTimeout** event is also fired for new communication outages.
- Changes to this value only take effect if set before calling the **Connect** method.

Property **SendSynchronousEvents**() As Boolean

Summary: Indicates whether the comserver should send **WatchdogTimeout** and **WatchdogReconnect** events that occur in response to client requests

Default: True

Remarks:

- When True, the server will both send a **WatchdogTimeout** event and signal E_PENDING when a communications outage is first detected in response to a client command.
- When False, the **WatchdogTimeout** event is suppressed and the client application is expected to derive Channel status from the E_PENDING state.
- The property has a similar effect when a Channel is determined to have reconnected after an outage.
- The server always fires **Timeout** and **Reconnect** events when there is no corresponding notification on the Channel, such as when the outage is detected during a heartbeat poll or on another Channel's request.

Methods Descriptions

GetACRCustom(ByVal *bstrRequest* As String) As Object

Summary: Returns the value(s) of requested p-Parameter(s)

Parameters:

bstrRequest: String of up to 32 p-Parameters, comma delimited.

Remarks:

- The **GetACRCustom** method returns an array of up to 32 data elements.
- Requested data can be either Float or Long parameters. Data types can be mixed within a single **GetACRCustom** request.
- Each p-Parameter in the request returns the values of the type as defined in the Parameters Reference section of the ACR-View User's Guide online help.

GetACRGroup(ByVal *bstrRequest* As String) As Object

Summary: Returns the values of requested p-Parameter group(s)

Parameters:

bstrRequest: String of up to 4 p-Parameters, comma delimited. Parameters in the request are used to look up the parameter group, which contains 8 similar parameters.

Remarks:

- Returns an array of up to 32 data elements. Each p-Parameter in the request results in a group of 8 values of the same type.
- Requested data can be either Float or Long parameters. Data types can be mixed within a single request.
- Each p-Parameter in the request returns the values of the type as defined in the Parameters Reference section of the ACR-View User's Guide online help.
- Any p-Parameter within a group can be used to identify the group.

GetACRMemory(ByVal *nType* As Integer, ByVal *nAddress* As Integer, ByVal *nCount* As Integer) As Object

Summary: Returns values of parameters or variables from a specified memory location

Parameters:

nType: ACR Data type of the values being requested

0 = LV, LA (Long)

1 = DV, DA (Float 64-bit)

2 = SV, SA (Float 32-bit)

nAddress: Memory address of the data requested

nSize: The number of values to be read from memory, starting at *nAddress*.

Remarks:

- The returned array can be of any size but is limited to a single data type.
- *nAddress* is the value retrieved from **GetParmAddr**, **GetLocalAddr**, or **GetLocalArrayAddr**.
- *nSize* should be set equal or less than the value returned from **GetLocalAddr** or **GetLocalArrayAddr**.

GetLocalAddr(ByVal *nProg* As Integer, ByVal *nType* As Integer, ByRef *nSize* As Integer) As Integer

Summary: Gets the memory address for AcroBasic program local variables

Parameters:

nProg: Program number of where the requested local variables are dimensioned

nType: ACR Data type of the values being requested

0 = LV (Long)

1 = DV (Float 64-bit)

2 = SV (Float 32-bit)

nSize: Method returns the count of the variables dimensioned

Remarks:

The returned address can be used in either the **GetACRMemory**, **AddACRMemory**, or **SetACRMemory** methods.

A return value of 0 indicates that this variable type is not dimensioned in the selected program.

GetLocalArrayAddr(ByVal *nProg* As Integer, ByVal *nType* As Integer, ByVal *nArray* As Integer, ByRef *nSize* As Integer) As Integer

Summary: Gets the memory address for AcroBasic program local array variables

Parameters:

nProg: Program number of where the requested local variables are dimensioned

nType: ACR Data type of the arrays being requested

0 = LA (Long)

1 = DA (Float 64-bit)

2 = SA (Float 32-bit)

nArray: Specific array number within the program

nSize: Method returns the size of the array

Remarks:

- The returned address can be used in either the **GetACRMemory**, **AddACRMemory**, or **SetACRMemory** methods.
- A return value of 0 indicates that this variable type is not dimensioned in the selected program.

GetParmAddr(ByVal *nParameter* As Integer) As Integer

Summary: Gets the memory address for a p-Parameter

Parameters:

nParameter: Any numeric p-Parameter

Remarks:

- The returned address can be used in either the **GetACRMemory**, **AddACRMemory**, or **SetACRMemory** methods.

IsFlagSet(ByVal *nFlagGrp* As Integer, ByVal *nFlagNdx* As Integer) As Boolean

Summary: Utility for identifying a bit in a 32-bit Long

Parameters:

nFlagGrp: A value of type Long containing flags (as bits)

nFlagNdx: Index of the flag

Remarks:

- **IsFlagSet** returns True if bit at *nFlagNdx* is 1, and returns False when the bit is 0.
- ACR Flag values are stored in 32-bit Longs, which is the lowest level of granularity provided by the **Status** methods. *nFlagGrp* is the Long value, *nFlagNdx* is the position of the flag in the Long.
- This function can also be called when the comserver is offline.

AddACRCustom(ByVal *bstrRequest* As String) As Integer

Summary: Adds a custom p-Parameter request into the status queue.

Parameters:

bstrRequest: String of up to 32 p-Parameters, comma delimited.

Remarks:

- Returns a key (*msgID*) identifying the request in the queue.
- The key can be used to retrieve data using **GetStatus**()
- Calling the Add routines places the specific status request into a constantly updated queue of requests. As data is retrieved from the controller for each request in the queue, that data is compared to existing data.
- The first time data is retrieved, and whenever the retrieved data has changed, a **StatusWaiting** event is generated with a key (*msgID)*

AddACRGroup (ByVal *bstrRequest* As String) As Integer

Summary: Add a group request into the status queue

Parameters:

bstrRequest: String of up to 4 p-Parameters, comma delimited. These parameters are used to look up the group, which is then used to return the 8 p-Parameter values for each group.

Remarks:

- Returns a key (*msgID*) identifying the request in the queue.
- The key can be used to retrieve data using **GetStatus**()
- Any p-Parameter in a group can be used to identify a group. Up to 4 groups can be requested and any undocumented/reserved items in a group are returned as zero.
- Calling the Add routines places the specific status request into a constantly updated queue of requests. As data is retrieved from the controller for each request in the queue, that data is compared to existing data.
- The first time data is retrieved, and whenever the retrieved data has changed, a **StatusWaiting** event is generated with a key (*msgID)*.

AddACRMemory(ByVal *nType* As Integer, ByVal *nAddress* As Integer, ByVal *nCount* As Integer) As Integer

Summary: Adds a memory value request into the status queue

Parameters:

nType: ACR Data type of the values being requested

0 = LV, LA (Long)

1 = DV, DA (Float 64-bit)

2 = SV, SA (Float 32-bit)

nAddress: Memory address of the data requested

nCount: The number of values to be read from memory, starting at *nAddress*

Remarks:

- Returns a key (*msgID*) identifying the request in the queue.
- The key can be used to retrieve data using **GetStatus**().
- *nAddress* is the value retrieved from **GetParmAddr**, **GetLocalAddr**, or **GetLocalArrayAddr.**
- *nCount* should be set equal or less than the value returned from **GetLocalAddr** or **GetLocalArrayAddr**.
- Calling the Add routines places the specific status request into a constantly updated queue of requests. As data is retrieved from the controller for each request in the queue, that data is compared to existing data. The first time data is retrieved, and whenever the retrieved data has changed, a **StatusWaiting** event is generated with a key (*msgID).*

DelStatus(ByVal *nMsgid* As Integer) As Object

Summary: Deletes a status request from the status queue

Parameters:

nMsgID: The key to a specific status request as returned by one of the ADD routines

Remarks:

- Removing unused status requests will speed up the update of the other requests in the queue.
- To clear all status requests in a queue, use **DelStatus**(−1).

GetStatus(ByVal *nMsgid* As Integer) As Object

Summary: Retrieves the specified status information

Parameters:

nMsgID: The key to a specific status request as returned by one of the ADD routines

Remarks:

- The returned array can be any size. It holds the values in Variants, either type Long or Float.

Event **StatusWaiting**(ByVal *msgID* As Integer, ByVal *error* As Integer)

Summary: Callback method acts as an event signaling that a there is data to read

Parameters:

msgID: The key to a specific status request as returned by one of the Add routines

error: If an error occurred getting a status update, it is reported in the *error* parameter. When a request encounters an error, the request is deleted from the queue.

Remarks:

- The COM event model (also know as Connection Points) uses a callback mechanism to generate events. The client program implements and registers this method.
- The ComACRServer calls the method when a status request (key=*msgID*) has been updated and is ready to read using the **GetStatus**() method.
- If an error occurred getting a status update, it is reported in the *error* parameter. When a request encounters an error the **SuppressStatusDelete** property value determines whether the error is ignored or if the item is deleted from the status queue.
- **StatusWaiting** events are launched on a separate thread. In .NET, the event should not try to directly access a Form Control, otherwise a "Cross-thread operation not valid" error may result. Delegates can be used in this case.

Status Example

Example 1

```
Dim objData As Object
Dim ACRClock As Integer
Dim OnBoardInputs As Integer
Dim bInput3 As Boolean
Dim CANInputs As Integer

'get the ACR system clock value in milliseconds
objData = Controller.GetACRCustom("P6916")
ACRClock = objData(0)

'get input on-board and CANopen inputs
objData = Controller.GetACRCustom("P4096,P4456")
OnBoardInputs = objData(0)
CANInputs = objData(1)

'check state of On-board input 3
bInput3 = Controller.IsFlagSet(OnBoardInputs, 3)
```

Example 2

A program is created on the controller to teach position values. The values are captured as encoder counts in a Long array, LA0. The values are uploaded to the PC, scaled by the AXIS PPU then returned to the controller in floating point array, SA1. The local variables in Program 1 are as follows:

```
P01>DIM
DIM SV(3)
DIM SA(2)
DIM SA0(10)
DIM SA1(10)
DIM LV(10)
DIM LA(2)
DIM LA0(10)
DIM LA1(10)

Const nLONG = 0
Const nSINGLE = 2
Dim MemAddr As Integer
Dim returnArr As System.Array
Dim iDataSize As Integer
Dim nProgram As Integer
Dim iPositions As System.Array
Dim arrPPU As System.Array
Dim PPU As Single

Dim fVel As Single
Dim fAcc As Single

'get the axis scaling factor: PPU
arrPPU = Controller.GetACRCustom("P12375")
PPU = arrPPU(0)

nProgram = 1

'get values stored in local single variables SV
'these values will be used for a move profile
MemAddr = Controller.GetLocalAddr(nProgram, nSINGLE, iDataSize)
returnArr = Controller.GetACRMemory(nSINGLE, MemAddr, iDataSize)
fVel=returnArr(0)
fAcc=returnArr(1)
```

```
'get values stored in local long array LA0
MemAddr = Controller.GetLocalArrayAddr(nProgram, nLONG, 0, iDataSize)
iPositions = Controller.GetACRMemory(nLONG, MemAddr, iDataSize)

'create new Object to contain scaled values
Dim fPositions(iDataSize - 1) As Object
For i As Integer = 0 To iDataSize - 1
   fPositions(i) = iPositions(i) / PPU
Next

'send these values back to controller, store in local single array SA1
MemAddr = Controller.GetLocalArrayAddr(nProgram, nSINGLE, 1, iDataSize)
Controller.SetACRMemory(nSINGLE, MemAddr, fPositions)
```

Control Methods

The ComACRServer provides methods for updating the ACR controller's state and action. Use the methods and their descriptions provided in this section to set and clear controller flags/bits and assign values to parameters. Refer to ACR-View online help for complete listing of parameters and flags, along with data types and read/write availability.

All updates can be sent via binary commands (immediate). Methods with a *bFast* parameter have the option of utilizing binary command or the ASCII interface (queued). In most cases, binary commands are preferred.

- When the *bFast* parameter is True, a binary command is sent to the controller. Binary commands are executed prior to any ASCII commands and will be executed even if *Move/Arc* commands are queued in the move buffer. The Terminal does not see binary commands.
- When the *bFast* parameter is False, an ASCII command is sent to the controller. ASCII commands queue in a command stack; they are visible in the Terminal interface. The command stack includes any ASCII commands (**Write**) and any *Move/Arc* commands.
- Setting *bFast* to False is the equivalent of using the **Write** method.

Use care to change only p-Parameters that are allowed to change. Modifying read-only parameter values has undefined behavior (and generally will not do what is intended).

Methods

- **SetFlag**
- **SetParmFloat**
- **SetParmLong**
- **SetParmLongMask**
- **SetGlobal**
- **SetACRMemory**
- **SetACRMemoryMask**

Descriptions

SetFlag(ByVal *nBit* As Integer, ByVal *bValue* As Boolean, ByVal *bFast* As Boolean)

Summary: Changes the value of a specific bit/flag on the ACR controller

Parameters:

nBit: Bit number on the ACR controller

bValue: Value of the bit to set

True: Sets the bit

False: Clears the bit

bFast: How to send the command

True: Binary

False: ASCII

Remarks:

- Many ACR commands have a bit/flag to perform the same action; for example, **Write**("AXIS0 DRIVE ON") could be replaced by **SetFlag**(8465, True,True).
- Whenever possible, utilize control flags in place of **Writes**.
- See the Control Example section for common control flags.

SetParmFloat(ByVal *nPparm* As Integer, ByVal *fValue* As Single, ByVal *bFast* As Boolean)

Summary: Changes the value of a specific p-Parameter of type Float

Parameters:

nPparm: p-Parameter number to change

dValue: Value to assign p-Parameter

bFast: How to send the command

True: Binary

False: ASCII

Remarks:

- On the ACR controller, parameters can have the following types: Long, Float, or Double. Use this method to assign a value to Floats and Doubles. All Floats and Doubles are sent to the controller as Floats.
- Modifying p-Parameters that are 64-bit (e.g., the user global P-variables, P0 through P4095) using the binary command (*bFast*=True) results in a less than exact translation of the fractional part of the number. In addition, because **SetParmFloat**() only allows a 32-bit number in *fValue*, the *bFast*=False setting will not be able to take advantage of the extra precision provided in the 64-bit values.
- The **SetGlobal**() method can also be used to set global user p-Parameters.

SetParmLong(ByVal *nPparm* As Integer, ByVal *nValue* As Integer, ByVal bFast As Boolean)

Summary: Changes the value of a specific p-Parameter of type Long

Parameters:

nPparm: p-Parameter number to change

nValue: Value to assign to p-Parameter

bFast: How to send the command

True: Binary

False: ASCII

Remarks:

- On the ACR controller, parameters can have the following types: Long, Float, or Double. Use this method to assign a value to Longs.

SetParmLongMask(ByVal *nPparm* As Integer, ByVal *nNAND* As Integer, ByVal *nOR* As Integer)

Summary: Changes the value of a specific parameter on the ACR controller

Parameters:

nPparm: p-Parameter number to change

nNAND: Used to clear bits

nOR: Used to set bits

Remarks:

- *nPparm* must point to a variable of type Long for the mask to properly work.
- The *nNAND* mask is used to clear bits, and the *nOR* mask is used to set bits.
- The data is modified as follows: data = (data AND NOT *nNAND*) OR *nOR.*

SetGlobal(ByVal nCard As Integer, ByVal *nGlobal* As Integer, ByVal *dValue* As Double, ByVal *bFast* As Boolean)

Summary: Changes the value of a specific, pre-dimensioned global parameter

Parameters:

nCard: This parameter is no longer used. It has been retained for backward compatibility

nGlobal: Global p-Parameter number that is to be changed

dValue: Value to assign p-Parameter

bFast: How to send the command

True: Binary

False: ASCII

Remarks:

- The range of global parameters is 0 through 4095. They are optionally allocated (using the DIM command) and are stored internally as 64-bit floating-point values.
- The *bFast* parameter of this method determines if the parameters are to be assigned a value using a binary (*bFast*=True) or an ASCII (*bFast*=False) command.
- **SetGlobal** is now equivalent to **SetParmFloat**().
- In previous versions of the comserver (BoxBridgeLib), *nCard* was used to determine the controller hardware type. The

hardware type is now determined automatically; *nCard* is ignored. Set *nCard*=0 in new applications.

SetACRMemory(ByVal *nType* As Integer, ByVal *nAddress* As Integer, ByVal *values* As Object)

Summary: Changes the value of a specific memory address on the ACR controller

Parameters:

nType: Data type of the values being set

0 = Integer (Long)

1 = Float (64-bit)

2 = Float (32-bit)

nAddress: The starting physical memory address on the ACR product

values: The data to be placed in memory starting at the address

Remarks:

- Any number of values can be placed into the ACR memory, but they must be all of the same type.
- *nAddress* is the memory address location returned from **GetParmAddr**, **GetLocalAddr**, or **GetLocalArrayAddress.**

SetACRMemoryMask(ByVal *nAddress* As Integer, ByVal *nNAND* As Integer, ByVal *nOR* As Integer)

Summary: Changes the value of a specific memory address on the ACR controller

Parameters:

nAddress: The starting physical memory address on the ACR product. This address must point to a variable of type Long for the mask to properly work.

nNAND: Used to clear bits

nOR: Used to set bits

Remarks:

- The two bit masks are combined with the value at the address to result in a new bit image for the data.
- The address must point to a Long integer storage area.
- The *nNAND* mask is used to clear bits; the *nOR* mask is used to set bits.

- The data is modified as follows: data = (data AND NOT *nNAND*) OR *nOR*
- *nAddress* is the memory address location returned from **GetParmAddr**, **GetLocalAddr**, or **GetLocalArrayAddress.**

Control Example

Example1

Controller bits/flags are grouped to allow for easy access to multiple objects (axes, Masters, programs, etc) by simply applying an offset. For example, the drive enable flag of axis0 is 8465, Axis 1 is 8497, Axis2= 8529. The offset between axes is 32: BaseFlag + (32 x Axis#).

```
Private Sub DriveEnable(ByVal AxNumb As Integer, ByVal bEnable As
Boolean)
     Dim bitOffset As Integer = 32 * AxNumb
     Controller.SetFlag(8465 + bitOffset, bEnable, True)
End Sub

Private Sub RunProgram(ByVal ProgNumb As Integer, ByVal bStart As
Boolean)
        Dim bitOffset As Integer = 32 * ProgNumb
        If bStart = True Then
            'Run Request Flag
            Controller.SetFlag(1032 + bitOffset, True, True)
        Else
            'Halt Request Flag
            Controller.SetFlag(1033 + bitOffset, True, True)
        End If

End Sub

 Private Sub EPLNetwork(ByVal bStart As Boolean)
        If bStart = True Then
            'EPL Start Flag
            Controller.SetFlag(16640, True, True)
        Else
            'EPL Reset Flag
            Controller.SetFlag(16641, True, True)
        End If
End Sub

Private Sub CANNetwork(ByVal bStart As Boolean)
        If bStart = True Then
            'CAN Start Flag
            Controller.SetFlag(11265, True, True)
        Else
            'CAN Reset Flag
            Controller.SetFlag(11266, True, True)
        End If
End Sub

Private Sub KillAxis(ByVal AxNumb As Integer, ByVal bKill As Boolean)
            Dim bitOffset As Integer = 32 * AxNumb
            Controller.SetFlag(8469 + bitOffset, bKill, True)
        Catch ex As Exception
End Sub

Private Sub JogFwd(ByVal AxNumb As Integer, ByVal bOn As Boolean)
     Dim bitOffset As Integer = 32 * AxNumb
     Controller.SetFlag(796 + bitOffset, bOn, True)
```

```
End Sub

Private Sub JogRev(ByVal AxNumb As Integer, ByVal bOn As Boolean)
     Dim bitOffset As Integer = 32 * AxNumb
     Controller.SetFlag(797 + bitOffset, bOn, True)
End Sub
```

Example 2

Parameters are grouped to allow for easy access to multiple objects (axes, Masters, programs, etc) by simply applying an offset. For example, the Jog Velocity parameter of Axis0 = P12348, Axis1 = P12604, Axis2 = P12860. The offset between axes is 256.

```
Private Sub SetJogVelocity(ByVal AxNumb As Integer, ByVal vel As
Single)
    Dim pOffset As Integer = 256 * AxNumb
    Controller.SetParmFloat(12348, vel, True)
End Sub
```

Example 3

Controller bits and flags are stored as a group of 32 within a P-parameter. Bits can be controlled by using SetFlag for an individual bit, by setting the P-parameter with SetParmLong, or by using **SetParmLongMask**. The following example refers to User Flags 128 thru 159, stored in P4100.

```
 'set bits 128-135, clears bits 136-143
 For i As Integer = 0 To 7 Step 1
     Controller.SetFlag(128 + i, True, True)
     Controller.SetFlag(136 + i, False, True)
 Next

'clears bits 128-135, sets bits 136-143
 For i As Integer = 0 To 7 Step 1
      Controller.SetFlag(128 + i, False, True)
Controller.SetFlag(136 + i, True, True)
 Next

'set bits 128-135, clears all others in P4100
Controller.SetParmLong(4100, 255, True)

'set bits 136-143, clears all others in P4100
Controller.SetParmLong(4100, 65280, True)

'clears bits 128-135, sets bits 136-143
Controller.SetParmLongMask(4100, 255, 65280)

'set bits 128-135, clears bits 136-143
Controller.SetParmLongMask(4100,65280,255)
```

Move Properties and Methods

The ComACRServer provides the following properties and methods for sending move commands to the controller. The controller should be fully configured, typically using ACR-View, before sending any moves. Descriptions of the properties and methods follow the list.

Properties

- **bArcAbsolute**() As Boolean
- **bArcCCW**() As Boolean
- **bMoveAbsolute**() As Boolean
- **fMoveACC**() As Single
- **fMoveFVEL**() As Single
- **fMoveVEL**() As Single
- **nMoveProfile**() As Integer
- **nArcMode**() As Integer
- **nMoveCounter**() As Integer
- **nMoveMode**() As Integer

Methods

- **Arc**(ByVal *nMask* As Integer, ByVal *targets* As Object)
- **Move**(ByVal *nMask* As Integer, ByVal *targets* As Object)
- **SendRES**(ByVal *nMask* As Integer)
- **Stop**(ByVal *bDecel* As Boolean)

Example

The following sample configuration describes the use of the ComACRServer **Move** and **Arc** commands.

Using the terminal in ACR-View, type **ATTACH** at the SYS> prompt to return the controller configuration.

```
SYS> ATTACH
PROG0
ATTACH MASTER0
ATTACH SLAVE0 AXIS0 "X"
ATTACH SLAVE1 AXIS1 "Y"
ATTACH SLAVE2 AXIS2 "Z"
PROG1
ATTACH MASTER1
ATTACH SLAVE0 AXIS3 "A"
ATTACH SLAVE1 AXIS4 "B"
ATTACH SLAVE2 AXIS7 "V"
ATTACH MASTER2
ATTACH SLAVE0 AXIS5 "C"
ATTACH SLAVE1 AXIS6 "U"
```

Remarks:

- **nMoveProfile** refers to the Master number which designates the group of axes to move.
- The *nMask* parameter in **Arc** and **Move** commands is treated as a field of 32 bits, created using the Slave numbers of the axes attached to the Master. Each bit in the mask corresponds to an attached Slave. The axis number is not considered, only the order in which the axes are attached as Slaves to the Master.

Slave #	Value in *nMask*
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

Example

nProfile(Master#)	Axes included in move	*nMask*	Binary Mask
0	0 and 1	1+2 = 3	0011
0	0, 1 and 2	1+2+4=7	0111
0	2 and 3	2+4=6	0110
1	3 and 4	1+2 =3	0011
1	3, 4 and 7	1+2+4=7	0111
1	3 and 7	1+4=5	0101
2	6	2	0010

Descriptions

Property **bArcAbsolute**() As Boolean

Summary: Determines if arc centers are treated in absolute or relative terms

Default: True

Remarks:

- When True, the target parameters of the **Arc** method becomes the new center of the arc.
- When False, the target parameters of the **Arc** method are adjusted incrementally, relative to the current position.

Property **bArcCCW**() As Boolean

Summary: Determines the direction of the **Arc** move

Default: True

Remarks:

- When True, a counter-clockwise (CCW) arc is defined from the positive primary axis toward the positive secondary axis.
- When False, a clockwise (CW) arc is defined from the positive primary axis toward the negative secondary axis.
- *nArcMode* defines the primary and secondary axes.

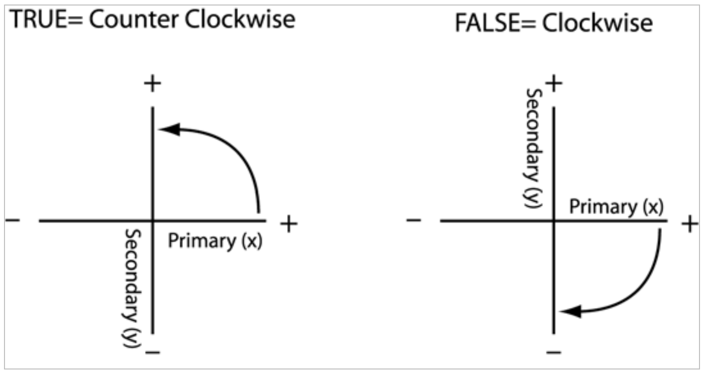


Figure 4: Arc Move Directions

Property **bMoveAbsolute**() As Boolean

Summary: Determines if **Move**() targets are treated in absolute or relative terms

Default: True

Remarks:

- When True, the target values of the **Move**() method are treated as the new absolute position.
- When False, the move is relative to the current position (either backward or forward from the current position by the target amount).

Property **fMoveACC**() As Single

Summary: Sets a new Profile Acceleration/Deceleration for the next move

Default: −1

Remarks:

- Part of a motion profile is the acceleration and deceleration, which can be set prior to the move, e.g., during configuration. Use this property to set a new acceleration.
- Refer to the **ACC**, **DEC**, and **STP** commands in the ACR user's guide and command reference for additional information about Master profiles.
- If this value is negative (default), it will be ignored and the existing profile acceleration will be used.
- The value is not sent to the controller until the next **Move**() or **Arc**() command is sent. Once a **Move** or **Arc** command is sent, and **fMoveACC** is a value other than −1, the **ACC** for the profile will be changed, as if the **ACC** command had been issued.
- **fMoveACC** is a global setting and is not directly associated with a Master.

Warning—There is no provision in the move command structure to adjust the JRK (jerk or S-curve) or Feedrate Override (FOV or ROV). If the JRK, FOV, or ROV has been calculated and set based on an existing VEL and ACC/DEL, changing the acceleration using this property can result in an unexpected motion profile. The FOV and ROV can be set independently with FOV and ROV commands.

Property **fMoveFVEL**() As Single

Summary: Sets a new Profile Final Velocity for the next move

Default: −1

Remarks:

- Part of a motion profile is the final velocity, which can be set prior to the move, e.g., during configuration. Use this property to set a new final velocity.
- If this value is negative (the default), it will be ignored and the existing profile final velocity will be used.
- The effect of setting this value to a value other than −1 will be to permanently change the **FVEL** for the profile, as if the **FVEL** command had been issued.

- **fMoveFVEL** is a global setting and is not directly associated with a Master. The value is not sent to the controller until the next **Move**() or **Arc**() command is sent.

Property **fMoveVEL**() As Single

Summary: Sets a new Profile Velocity for the next move

Default: −1

Remarks:

- Part of a motion profile is the target velocity, which can be set prior to the move, e.g., during configuration. Use this property to set a new velocity.
- If this value is negative (default), it will be ignored and the existing profile velocity will be used.
- The effect of setting this value to a value other than −1 will be to permanently change the **VEL** for the profile, as if the **VEL** command had been issued.
- *fMoveVEL* is a global setting and is not directly associated with a Master. The value is not sent to the controller until the next **Move**() or **Arc**() command is sent.

Warning—There is no provision in the **Move** command structure to adjust the **JRK** (jerk or S-curve) or **FOV** or **ROV** (feedrate override). If the **JRK**, **FOV**, or **ROV** has been calculated and set based on an existing **VEL** and **ACC/DEL**, changing the acceleration using this property can result in an unexpected motion profile. The **FOV** and **ROV** can be set independently with **FOV** and **ROV** commands.

Property **nMoveProfile**() As Integer

Summary: Specifies the Master profile to use for the move

Default: 0

Range: 0–15

Remarks:

- **nMoveProfile** refers to the Master number, which designates the group of axes to move
- For a move to succeed a Master profile must be configured and have the physical axes attached.
- The profile must also include information about velocity, acceleration, deceleration, jerk, and feedrate override.

Property **nArcMode**() As Integer

Summary: Determines primary and secondary axes when performing an arc move

Default: 0

Range: 0–2

The arc mode defines the primary and secondary axes for the arc as follows:

0	Primary is Axis 0, Secondary Axis 1
1	Primary is Axis 1, Secondary Axis 2
2	Primary is Axis 2, Secondary Axis 0

Remarks:

- To define an arc, first assign the axes that will produce the compound motion to a Master.
- Using the **nArcMode** property defines the primary and secondary axes: the primary axis is usually the X-axis; the secondary axis is usually the Y-axis.
- Axes references are relative to the profile in use in the same way as *nMask* is defined.

Property **nMoveCounter**() As Integer

Summary: Activates and sets the direction of the move counter

Default: 1

Range: −1 to 1

The move counter has three (3) possible modes:

−1	Counter ON and counting DOWN
0	Counter OFF
1	Counter ON and counting UP

Remarks:

- When in mode −1 or 1, the move counter parameter is updated when a move starts, and can be monitored through the (Long) parameter values below.
- Monitor the move counter through these Master Parameters (Profile 0−15): P8208, P8464, P8720, P8976, P9232, P9488, P9744, P10000, P10256, P10512, P10768, P11024, P11280, P11536, P11792, P12048.
- The **nMoveCounter** property sets the mode for the move counter on a device (not the Bus Device Driver Move Counter as stated in some ACR documentation. The Bus Device Driver Move Counter is manipulated using the **GetMoveCounter**() and **SetMoveCounter**() methods.

Property **nMoveMode**() As Integer

Summary: Selects the move mode

Default: 2

Range: 0–3

There are four (4) possible move modes:

> *Continuous*: Uses **ACC** to get to **VEL * FOV** and stays there.
>
> *Cornering*: Uses **ACC** to get to **VEL * FOV** and **DEC** to get to **FVEL**
>
> *Start/Stop*: Uses **ACC** to get to **VEL * FOV** and **STP** to **Stop** (**VEL**=0)
>
> *Rapid Start/Stop*: Uses **ACC** to get to **VEL * ROV** and **STP** to **Stop** (**VEL**=0)

Remarks:

- Figure 5 , which follows, shows the four move modes.

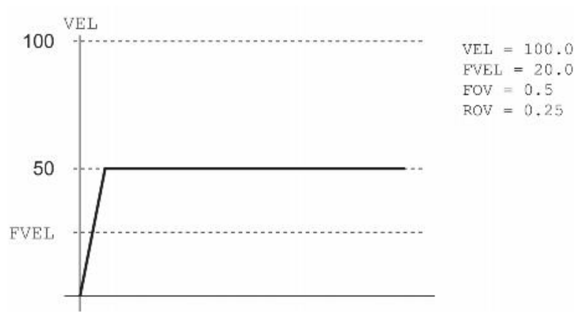
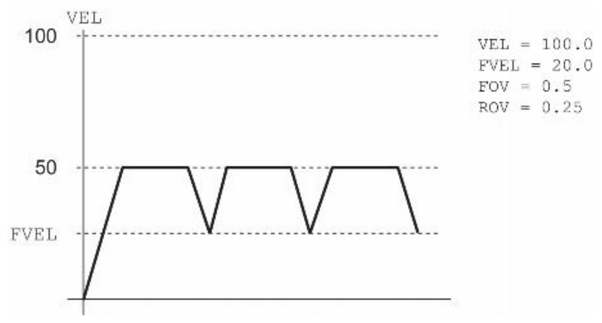
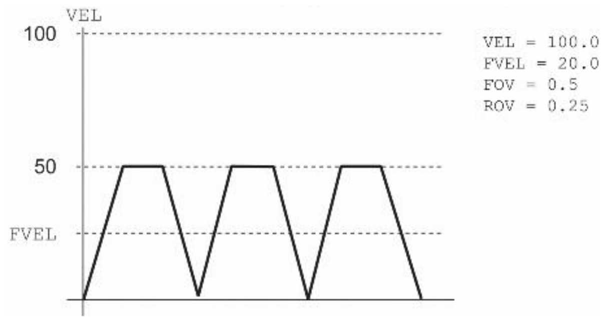
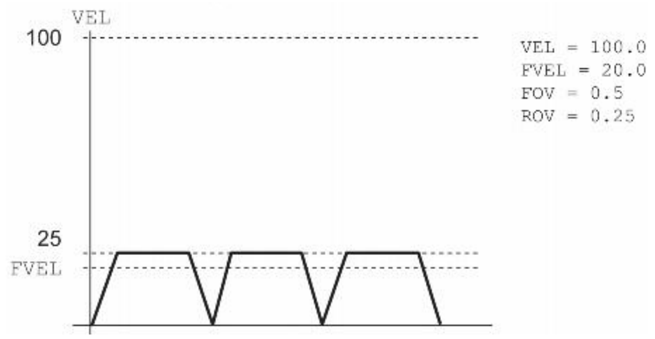
Mode 0 Continuous	Mode 1 Cornering
	
Mode 2 Start/Stop	**Mode 3 Rapid**
	

*Figure 5: **nMoveMode** Move Modes*

Arc(ByVal *nMask* As Integer, ByVal *targets* As Object)

Summary: Generates an arc move

Parameters:

nMask: Specifies which axes to use for the arc move

targets: The arc centers and target position information for each axis

Remarks:

- The **Arc** method allows from 1 to 16 attached axes to be part of a move. This happens by setting one or more target positions for each axis.

- To perform an **Arc** move, the profile defined in **nMoveProfile** property must have one or more axes attached.
- In the **Arc**() method, the target positions are stored in the *targets* array parameter, while the *nMask* specifies to which axes the data is linked.
- The first two elements of the *targets* array are the primary and secondary centers for the arc. The target positions are placed beginning at the third element of the array.

Move(ByVal *nMask* As Integer, ByVal *targets* As Object)

Summary: Generates a move

Parameters:

nMask: Specifies which axes to use for the move

targets: The target position information for each axis

Remarks:

- The **Move**() method allows from 1 to 16 attached axes to be part of a move. This happens by setting one or more target positions for each axis.
- To perform a move, the Master (axis group) defined in **nMoveProfile** property must have one or more axes attached.
- The *targets* array contains the data used to make the move, e.g. the target positions of the move. The array should only hold data for axes that are flagged in the *nMask* parameter. For example, setting *nMask* to 5 (binary 0101) tells the **Move**() method to move Axis0 and Axis2. Therefore, the targets array needs to hold two values. The targets array value at index 0 applies to the Axis0 move, the value at index 1 applies to the Axis2 move.
- If the number of target values does not match the number of axes in *nMask*, the comserver will throw a {"Value does not fall within the expected range."} error.
- The *targets* array contains variants because the target data can be either Float or Long, but not a combination of Floats or Longs. The first data type found is the data type used for all data in the array.

SendRES(ByVal *nMask* As Integer)

Summary: Send a **RES** command to an axis

Parameters:

nMask: Specifies which axes to apply the **RES**

Remarks:

- The **RES** command resets the encoder and other position counters on the axis to zero for the specified axes attached to the profile defined in the **nMoveProfile** property.
- See the notes about *nMask* in the previous general remarks about **Move** properties and methods.
- This command does not use the binary syntax, so it will be queued up in the command stack behind moves, etc.

Stop(ByVal *bDecel* As Boolean)

Summary: Stops commanded motion for the profile specified in **nMoveProfile**

Parameters:

bDecel: How to stop motion

True: Stop All Moves

False: Kill All Moves

Remarks:

- When the *bDecel* parameter is True, a Stop All Moves flag is set using the binary command. It uses the existing **DEC** and **JRK** values.
- When the *bDecel* parameter is False, a Kill All Moves flag is set using binary command. Ignores the existing **DEC** and **JRK** values.
- After using **Stop**(), the application must clear the Kill All Moves flag before any new move can be made. This is the case for either value of *bDecel*, the Stop All Moves flag self clears and sets the Kill All Moves flag.
- Stop All Moves Bits values for **nMoveProfile:** 0–15 Master Flags are: BIT523, BIT555, BIT587, BIT619, BIT651, BIT683, BIT715, BIT747, BIT7435, BIT7467, BIT7499, BIT7531, BIT7563, BIT7595, BIT7627, BIT7659.
- Kill All Moves Bits Values for **nMoveProfile:** 0–15 Master Flags are: BIT522, BIT554, BIT586, BIT618, BIT650, BIT682, BIT714, BIT746, BIT7434, BIT7466, BIT7498, BIT7530, BIT7562, BIT7594, BIT7626, BIT7658.
- The Stop All Moves or the Kill All Moves Request flag stops commanded motion at the Master level. It is equivalent to calling **SetFlag**(BITx, True), where x=a Stop All Moves or Kill All Moves flag number.

Note: To stop axis-based moves like JOG, CAM, and GEAR requires knowing which axis is moving. The comserver does not maintain that information. Therefore, the Stop method is only able to stop motion for at the Master level. There are specific bit flags for stopping JOG, CAM, and GEAR, which can be set using SetFlag().

- Another option available to kill commanded motion and/or **JOG**, **CAM**, and **GEAR** is to use **SetFlag**(KAMR flag, True, True) with the KAMR flag associated with any axis attached to the Master profile. The Kill All Motion Request (KAMR) flag (Axis0-15: BIT8467–BIT8947) stops all motion, without regard to deceleration, on all axes connected to the common Master profile of the axis this command is issued for. This flag must be cleared on all axes it is set on before any motion can be commanded (or anything else done). The Kill All Moves flag (Axis0-7: BIT522–BIT746, Axis8-15: BIT7434–BIT7658) for the profile will also be set, and must be cleared prior to doing motion.

Alarm Events

The ComACRServer provides users with the ability to subscribe to controller-generated events. *These events are only available when connecting via Ethernet.* Alarm events utilize the ACR controller's alarm logic to subscribe to events specified in the previous table. Alarm entries resemble status entries in that an event is fired whenever the monitored data value changes. The difference is that alarm entries are monitored by the controller itself and every transition is guaranteed to be detected and counted. (Status events require server polling, and events are lost if two or more transitions occur during a single polling interval.) When the controller detects that a monitored alarm value has transitioned a signal is sent to the comserver, which fires an **AlarmEvent**. Since it is possible that multiple transitions can occur in the time it takes to generate an event, the **AlarmEvent** returns the number of alarms since the last Alarm Event, so the client software can take the appropriate action.

Note: Controller-generated events are only available when connecting through Ethernet.

Events

- Sub **AddAlarmEvent**(ByVal *iAlarmEvent* As Integer, ByVal *iAlarmParm* As Integer)
- Sub **DeleteAlarmEvent**(ByVal *iAlarmEvent* As Integer, ByVal *iAlarmParm* As Integer)
- Event **AlarmEvent**(ByVal *iAlarmType* As Integer, ByVal iAlarmParm As Integer, ByVal *iAlarmCnt* As Integer)

Descriptions

Sub **AddAlarmEvent**(ByVal *iAlarmEvent* As Integer, ByVal *iAlarmParm* As Integer)

Summary: Subscribes to an ACR alarm event

Parameters:

iAlarmEvent: The type of control alarm being subscribed to

iAlarmParm: Adds qualifiers to *iAlarmEvent*, defining the axis, Master, or program associated with the alarm

Alarm Types

iAlarmEvent	Description	iAlarmParm
0	Move Started	Master #
1	Move Ended	Master #
2	Enable Lost	0
3	Drive Fault	Axis #
4	Limit Hit	Axis #
5	Stall Detect	Axis #
6	Program Done	Program #
7	Encoder Disconnected	Encoder #
8	Encoder Fail	Encoder #
9	CANOpen Fail	0
10	Move Counter Update	Master #

Remarks:

- Add events after a connection has been established.
- This event is only available for Ethernet connections.

Sub **DeleteAddAlarmEvent**(ByVal *iAlarmEvent* As Integer, ByVal *iAlarmParm* As Integer)

Summary: Cancels the subscription to ACR alarm event

Parameters:

iAlarmEvent: The type of control alarm being cancelled

iAlarmParm: Adds qualifiers to *iAlarmEvent*, defining the axis, Master, or program associated with the alarm

---

Event **AlarmEvent**(ByVal *iAlarmType* As Integer, ByVal *iAlarmParm* As Integer, ByVal *iAlarmCnt* As Integer)

Summary: Callback method acts as an event signaling that a controller alarm has occurred at least once since the last alarm event

Parameters:

iAlarmEvent: Identifies the type of alarm occurring. See the parameter table in the **AddAlarmEvent** description.

iAlarmParm: Provides further identification about the alarm occurring

iAlarmEvent: Provides the number of times the controller detected the alarm since this alarm was last generated

Alarm Example

```
Dim WithEvents Controller As ComACRServerLib.Channel


Controller.Connect(3, 0)
Controller.AddAlarmEvent(2, 0)
'// Add alarm for ACR9000 Motion Enable input
Controller.AddAlarmEvent(6, 2)
'// Add alarm for Program Done, Program #2
Controller.AddAlarmEvent(6, 3)
'// Add alarm for Program Done, Program #3


Private Sub Controller_AlarmEvent(ByVal iAlarmType As Integer, ByVal
iAlarmParm As Integer, ByVal iAlarmCnt As Integer) Handles
Controller.AlarmEvent
        If iAlarmType = 2 Then
            MsgBox("Motion Enable Input is Open")
        End If
        If iAlarmType = 6 Then
            If iAlarmParm = 2 Then
                MsgBox("Program 2 is done")
            ElseIf iAlarmParm = 3 Then
                MsgBox("Program 2 is done")
            End If
        End If
 End Sub
```

Utility Methods

The Utility Interface provides functionality for transferring files between the PC and the ACR product. The File transfer to the ACR controller (both OS and program files) is non-blocking. This allows the transfers to be monitored for status and canceled by the user. The file transfer from the ACR controller to the PC, file upload, is a blocking transfer.

The non-blocking transfers allow the client software to be responsive during a download and inform the user of progress, but they also imply that code running after a transfer has started should not assume the transfer has completed. The code should check the status before attempting to talk to the controller. For example, a function that downloaded a program cannot expect to run that program after returning from the download call, but must wait until the status information indicates the transfer is complete.

Properties

- Property **nProgramDownloadEcho**() As Integer

Methods

- Sub **DownloadFile**(ByVal *bstrPrg* As String, ByVal *bstrFile* As String)
- **GetStatusDL**(ByRef *nTotal* As Integer, ByRef *nBytes* As Integer) As Integer
- **GetStatusDLEx**(ByRef *nTotal* As Integer, ByRef *nBytes* As Integer, ByRef *bstrExtendedErrorMessage* As String) As Integer
- **StopDownload**()
- **UploadFile**(ByVal *bstrPrg* As String, ByVal *bstrFile* As String)

Descriptions

Property **nProgramDownloadEcho**() As Integer

Summary: Determines the echo settings used during file downloads

Default: 3

Range: 1–3

Echo Types		
Value	Description	ACR ECHO Mode
1	Full Echo	1
2	Silent Mode	6
3	Errors Only	4

Remarks:

- This property will set the controller's ECHO (see the ACR command reference for details) mode. Normally, the controller will echo back all ASCII characters, the command prompt, and any error messages (e.g., Syntax Error). The comserver defaults to only echoing back errors when downloading programs. This increases the downloading speed.
- After the program download, the controller is returned to the current ECHO setting.
- *nProgramDownloadEcho*=1 corresponds to ECHO1, returning command prompt, all characters and error messages.
- *nProgramDownloadEcho*=2 corresponds to ECHO6, turning off all echos.
- *nProgramDownloadEcho*=3 corresponds to ECHO4, returning only error messages.

Sub **DownloadFile**(ByVal *bstrPrg* As String, ByVal *bstrFile* As String)

Summary: Transfers a text file to the ACR controller

Parameters:

bstrPrg: Specifies the location to which files are downloaded (for example: SYS or PROG01 or PLC01). This parameter is not required and can be an empty string.

bstrFile: Specifies the fully qualified name of the file to download.

Remarks:

- If the program or PLC download is attempted (e.g., *bstrPrg*=PROG2):
    - *bTerminalLock* is set to True.
    - That ProgramPLC is stopped using the Halt Request flag.
    - **NEW** command is sent to clear the existing program.

  - *bstrgPrg* is sent to the controller.
  - Program text is sent line by line.

- This method is non-blocking, returning as soon as the file has started to transfer.
- Check the **GetStatusDLEx**() for completion updates during the use of this method.
- When receiving a downloaded AcroBasic or PLC program, the controller may issue error messages in response to program lines. These may result from coding errors in the program, or they may result from communication glitches.
- The download will continue until ten of these errors are detected, at which time the server will stop the download and erase the partial program. **GetStatusDLEx**() will return a value of 21 in this case.
- If the control reports 1–9 errors, the download will continue, and **GetStatusDLEx**() will have a return value of 21 and return an error string containing the program line containing the error and the error text returned by the control.

Note: Once downloaded, a program/PLC is in memory but has not been saved to flash. For information on permanently saving a downloaded program or PLC, see the FLASH SAVE/FLASH IMAGE commands in the ACR command reference. Remember that these flash commands only work if no programs are currently running.

GetStatusDL(ByRef *nTotal* As Integer, ByRef *nBytes* As Integer) As Integer

Summary: Returns current status of the active download

Parameters:

nTotal: Total bytes to be transferred

nBytes: Total number of bytes transferred so far

Return:

Download Status	
Value	Description
0	No transfer in progress
1	Transfer in progress
2	End of transfer
3	User cancelled transfer
4	Error reading from file
5	Too many errors during transfer

Download Status	
Value	Description
6	Transfer has timed out waiting for response
7	The ACR OS failed to verify against the hardware description file (config image)
8	Save to flash of OS in progress.
9	Problem encountered in saving OS to flash
16	Program space dimensioned too small
21	Download complete, but control reported errors
22	Attempt to dimension program space failed
Negative number	Unexpected error

Remarks:

- If an OS download is in process, the *nBytes* parameter value is continually updated with the amount of data sent to the ACR controller.
- The return value indicates the state of the download.

GetStatusDLEx(ByRef *nTotal* As Integer, ByRef *nBytes* As Integer, ByRef *bstrExtendedErrorMessage* As String) As Integer

Summary: Returns current status of the active download

Parameters:

nTotal: Total bytes to be transferred.

nBytes: Total number of bytes transferred so far

bstrExtendedErrorMessage: A message providing more information about an error return

Return:

Download Status	
Value	Description
0	No transfer in progress
1	Transfer in progress
2	End of transfer
3	User cancelled transfer
4	Error reading from file
5	Too many errors during transfer
6	Transfer has timed out waiting for response
7	The ACR OS failed to verify against the hardware description file (config image)
8	Save to flash of OS in progress.

Download Status	
Value	Description
9	Problem encountered in saving OS to flash
16	Program space dimensioned too small
21	Download complete, but control reported errors
22	Attempt to dimension program space failed
Negative number	Unexpected error

Remarks:

- If an OS download is in process, the *nBytes* parameter value is continually updated with the amount of data sent to the ACR controller.
- The return value indicates the state of the download.
- When receiving a downloaded AcroBasic or PLC program, the controller may issue error messages in response to program lines. These may result from coding errors in the program, or they may result from communication glitches.
- When the controller returns an error in response to a program line, *bstrExtendedErrorMessage* has the format [program line with error] >>> [error message].
- Note that an error line will not appear in an upload or **LIST** listing.

StopDownload()

Summary: Aborts the file transfer to the controller

Remarks:

- Call this method to cancel a download. The status value of 3 will be returned from **GetStatusDL**() or **GetStatusDLex**() after this method is called.

UploadFile(ByVal *bstrPrg* As String, ByVal *bstrFile* As String)

Summary: Uploads an AcroBasic program or PLC program from the controller to the PC

Parameters:

bstrPrg: Specifies the location to which files are uploaded from (for example: SYS or PROG01 or PLC01)

bstrFile: Specifies the fully qualified name of the file to receive the upload

Remarks:

- This method blocks any other instructions from running until the upload is complete. There is no checking of the code uploaded.
- When uploading, the *bstrPrg* parameter is sent to change the command prompt prior to sending a **LIST** command and capturing all the data returned.

Error Messages

The ComACRServer uses the standard COM/OLE error codes found in the following table when it encounters problems. These error values are returned as part of a Microsoft COM exception.

Error Code and Name	Description
X8000000A E_PENDING	Operation requires connection to an inaccessible device ("disconnected" return)
X80004001 E_NOTIMPL	Interface defined but not implemented. This can be returned for denigrated interfaces, such as bstrUSBSerialNumber.
X80004004 E_ABORT	Unrecoverable communications failure (Non-existent COM port, for instance)
X80004005 E_FAIL	Controller rejected requested operation.
X8007000E E_OUTOFMEMORY	Could not allocate sufficient memory for operation
X80070057 E_INVALIDARG	One or more call arguments is invalid
X8000FFFF E_UNEXPECTED	Internal ComACRServer error

Messages

- ReadOnly Property **bstrLastError**() As String
- ReadOnly Property **nLastErrorCode**() As Integer

Descriptions

ReadOnly Property **bstrLastError**() As String

Summary: Returns legacy error code of last error occurring

Default: True

Remarks:

- The **bstrLastError** property returns a textual description of the last error reported. Most often, it is an error message from the table below, but in cases where ASCII commands are sent to the ACR control, it may be an ASCII error message created by the control.
- Sometimes when the comserver issues ASCII commands (as it does during file download and upload), **bstrLastError** is set to the literal error message returned by the ACR control.

ReadOnly Property **nLastErrorCode**() As Integer

Summary: Returns legacy error code of last error occurring

Default: True

Remarks:

- When the ComACRServer encounters an error, it generates an exception using standard COM/OLE error codes.
- The Legacy Error code (also described in the Error Messages section) of the last error detected may be retrieved through the **nLastErrorCode** property.

The comserver will provides more detailed information about each error condition after it throws one of the above exceptions. The **nLastErrorCode** property returns a code from the following table describing the last error reported.

Error Code	Error Message
17000	Unknown Error Intercepted.
17001	The value requested is not present in Resource file.
17002	The PPU must be a number greater than zero.
17003	The valid range for the encoder resolution multiplier is between −4 and 4.
17004	The minimum value for a stream is 256.
17005	The value is out of range.
17006	The file prefix number must be positive.
17007	The data collected failed to pass final validation.
17008	The file name is not valid.

Error Code	Error Message
17009	Cannot open file, check that directory exists and you have the correct permissions access the file.
17010	The expected Map Index within the file was not found.
17011	The space reserved for storing loop information has become full.
17012	The line positioning information for the loop is not present.
17013	Problem trying to calculate value, Right Parentheses expected.
17014	Problem trying to calculate value, Function is not defined.
17015	Problem trying to calculate value, expected a primary expression.
17016	Problem trying to calculate value, trying to divide by zero.
17017	Problem trying to calculate value, an unrecognized character is being used.
17018	There is insufficient space to store the generated PLC programs. As much as possible has been stored.
17049	Some precondition for transferring the file has not been met.
17050	The Port is already connected. Only call the **Open**() method once per object or after a **Close**().
17051	The serial port number must be between 1 and 36.
17052	The maximum number of concurrently opened ports has been reached.
17053	There is a problem allocating memory for the port connection.
17054	Problem accessing port. Check that no other application is using the port and that the port settings are valid.
17055	Unable to create the read thread for the port.
17056	To perform read or write operations on the port, it must first be open.
17057	The Settings for the port are incorrect.
17058	The change to the BPS rate encountered some problem. BPS rate is unchanged.
17059	No file name provided. To transfer a file a file name must be provided.
17060	The File Transfer method requested is not provided at this time.
17061	Only one file transfer per port at one time. Please wait until the current file transfer has completed and retry.
17062	Unable to create the file transfer thread.
17063	Unable to create the mutex for the read thread.
17064	The file name provided could not be read.
17065	Unable to create or set the file transfer event for the port.
17066	The function has terminated because it encountered too many errors.
17067	The write function has timed out. Check that communications are working, or reduce the amount of data being written.

Error Code	Error Message
17068	The program is unable to process incoming data fast enough and is losing it. Make sure external device is using flow control.
17069	The communications flow control could not be changed due to some error.
17070	Unable to set the specified flow control characters.
17071	Unable to resize the Transmit or Receive buffers.
17072	The data provided to populate the object state failed in some way. The data appears to be corrupt.
17073	The expected return value for the transfer was not found.
17074	Unable to create the status thread.
17075	Unable to create or set the status event.
17076	The key provided to collect status is invalid or improperly formed.
17077	There is no status information available for the specified message key.
17078	Unable to create the mutex for the status.
17079	The specified device type or index is invalid.
17080	Problem communicating with controller. Check the port settings are valid and that the controller is powered up and connected.
17081	There was a problem reported while trying to wake up the device. The device may not be responding.
17082	The transfer was canceled by the user.
17083	The communication attempt has timed out because it did not get the expected response.
17084	One of the p-Parameters provided in the status request is unknown or poorly formed.
17085	The provided hex data string representation contains characters that are not valid hex numbers.
17086	The data provided in the array did not match the expected conditions or is not there at all.
17087	The socket type provided is not currently supported by the software.
17088	The Application Window cannot be found. The application may not be running.
17089	Old version of file found and read. New items and features added since this file was created will be set at their defaults.
17090	The destination device on the Network is not reachable at this time.
17091	There was a time out trying to communicate with the device on the Network.
17092	Some failure caused the echo request to fail.
17093	This version of the Ping Request requires the icmp.dll file be present on the computer. It was not found.

Error Code	Error Message
17094	The Host name or IP address cannot be resolved to a know host device. Check that IP or Host name is correct.
17095	The transfer was canceled at the other end of the connection.
17096	Failed to Create the desired Thread.
17097	Unable to Start the desired Thread.
17098	Unable to Create the desired Thread Event.
17200	Foreign Error Source
17201	Standard Library Exception
17202	Windows Socket Architecture Library Exception

INDEX