

# Compumotor

## Motion OCX Toolkit™ User Guide

*OCX Controls for  
6000 Series Bus-Based Controllers*

Compumotor Division  
Parker Hannifin Corporation  
p/n 88-016062-01 C eptember



## **Motion OCX Toolkit™**

Information in this document is subject to change without notice and does not represent a commitment on the part of Parker Hannifin Corporation. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be installed and used only in accordance with the terms of the agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use with permission of Parker Hannifin Corporation.

Motion Architect is a registered trademark, and Motion OCX Toolkit is a trademark of Parker Hannifin Corporation.

Microsoft is a registered trademark, and Visual Basic and Visual C++ are trademarks of Microsoft Corporation.

Delphi is a trademark of Borland International, Inc.

© Copyright 1996-7 by Parker Hannifin Corporation.

All rights reserved.

Printed in the United States of America.



### **WARNING**



Motion OCX Toolkit provides tools to control your motion system's electrical and mechanical components. Therefore, you should test your 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 ATC or distributor, or ...

#### **North America**

**Parker Hannifin Corporation**  
Compumotor Division  
5500 Business Park Drive  
Rohnert Park, CA 94928  
Telephone: (800) 358-9070 or (707) 584-7558  
Fax: (707) 584-3793  
BBS: (707) 584-4059  
FaxBack: (800) 936-6939 or (707) 586-8586  
Internet: [www.compumotor.com](http://www.compumotor.com)  
Email: [tech\\_help@cmotor.com](mailto:tech_help@cmotor.com)

#### **Europe**

**Parker Hannifin plc**  
Electromechanical Division - Digiplan  
21 Balena Close  
Poole, Dorset. BH17 7DX UK  
Telephone: +44 (0)1202 69 9000  
Fax: +44 (0)1202 69 5750

**Parker Hannifin GmbH & Co. KG**  
Electromechanical Division - Hauser  
Robert-Bosch-Str. 22  
D-77656 Offenburg, Germany  
Telephone: +49 (0)781 509-0  
Fax: +49 (0)781 509-176



**Product Feedback Welcome**

**Email: [6000user@cmotor.com](mailto:6000user@cmotor.com)**

# Contents

## Introducing Motion OCX Toolkit

Overview .....	1
Skills Required .....	2
System Requirements .....	2
Installation .....	2
About This Manual .....	2
Reference Documentation ( <i>provided in your product's ship kit</i> ) .....	2
Visual Basic Orientation .....	2
Convention for Code Examples .....	3
Basic Operations .....	3
Registering Your 6000 Product .....	3
Using Motion OCX Controls in Visual Basic 4.0 .....	4
Getting Online Help .....	6
Auto-Loading the 6000 Product Operating System .....	6
Distributing Applications ( <i>with Embedded Motion OCX Controls</i> ) .....	7

## Communications Shell OCX

Comm6000 Overview .....	9
Comm6000 Properties .....	9
Controller .....	10
Enabled .....	10
TimeOut .....	10
PollResponse .....	10
Comm6000 Methods .....	10
EnableInterrupt .....	10
GetFastStatusItem .....	10
ReceiveFile .....	12
Register6000 .....	12
SendCommand .....	12
SendFile .....	13
UpdateFastStatus .....	13
Comm6000 Events .....	14
GPInterrupt .....	14
Response .....	14

<b>Terminal OCX</b>	
Term6000 Overview .....	15
Term6000 Properties .....	16
BackColor .....	16
Controller.....	16
Enabled.....	16
Font.....	16
ForeColor .....	16
Term6000 Methods .....	17
Clear .....	17
Copy.....	17
<b>Fast Status Polling OCX</b>	
Poll6000 Overview .....	19
Poll6000 Properties .....	20
BackColor .....	20
Caption.....	20
CaptionPosition .....	20
Controller.....	20
Enabled.....	20
Font.....	20
ForeColor .....	20
InnerBorder .....	21
Item.....	21
OuterBorder .....	22
ScaleFactor .....	22
StatusBit.....	22
StatusBitOffText .....	26
StatusBitOnText .....	26
TextAlignment .....	26
UpdateRate .....	26
Value.....	26
Poll6000 Methods .....	27
Update .....	27
Poll6000 Events .....	27
ValueChange .....	27
<b>Appendix A: Tutorial.....</b>	<b>29</b>
<b>Appendix B: Using Controls with Visual C++ Non-Dialog Containers.....</b>	<b>39</b>

*Introducing ...*

# Motion OCX Toolkit

---

## Overview

Motion OCX Toolkit provides 32-bit Ole Custom Control Extensions (OCXs) designed to run under Windows 95 or Windows NT. The OCX controls can be used with Visual Basic 4.0, Delphi 2.0, Visual C++ 4.x, or any 32-bit development environment that can contain OCX controls. With the Motion OCX Toolkit, you can quickly develop your own custom operator interface to a Compumotor 6000 Series bus-based controller.

Motion OCX Toolkit includes these controls:

- **Communications Shell OCX (*Comm6000*):**  
Use Comm6000 to control basic communication with the 6000 product, including interrupt handling and sending/receiving files. For details, see page 9.
- **Terminal OCX (*Term6000*):**  
Use Term6000's active communication interface to the 6000 product for executing 6000 commands, checking program responses and status reports, and viewing error messages. For details see page 15.
- **Fast-Status Polling OCX (*Poll6000*):**  
Use Poll6000 to poll the 6000 product's fast status register and display information such as position, velocity, axis status, system status, etc. For details, see page 19.

Each control has a custom properties dialog box for easy setup. While in the design environment, if you need help, press F1 to access the online help system.

### **Skills Required**

To effectively use the Motion OCX Toolkit controls, you should be proficient at using your development tool (Visual Basic, Delphi, Visual C++, etc.). This manual addresses features that are specific to the Motion OCX controls. For general information on implementing OCX controls, refer to the documentation that accompanied your development tool. **NOTE:** The Tutorial provided on page 29 demonstrates how to implement the OCXs in a sample application.

### **System Requirements**

- 8 MB RAM (16 MB recommended)
- Windows 95 or Windows NT
- Visual Basic 4.0 (32-bit version), Visual C++ 4.x, Delphi 2.0 or any 32-bit development environment that can contain OCX controls

### **Installation**

Insert the Motion OCX Toolkit diskette into floppy disk drive A and run A:\Setup. The installer program will lead you through the installation process. The default location for Motion OCX is C:\MOTOCX.

- You must also install your 6000 product's operating system (.OPS) file, which is found on the "Operating System & DOS Support Software" diskette provided in the 6000 product's ship kit. See page 6 for details.
- NT Users: You must have an *administrative* security level to install the OCX controls and register the 6000 controller(s); however, only *user privileges* are required to run an application using the OCX controls.

---

## **About This Manual**

### **Reference Documentation** *(provided in your product's ship kit)*

For information about hardware installation and tuning, refer to your 6000 product's *Installation Guide*. For information on the 6000 command language and how to implement certain features, refer to the *6000 Series Software Reference* and the *6000 Series Programmer's Guide*. **NOTE:** These reference documents (including this manual) are also available as PDF files (read with Adobe Acrobat) from our web site at [www.compumotor.com](http://www.compumotor.com).

### **Visual Basic Orientation**

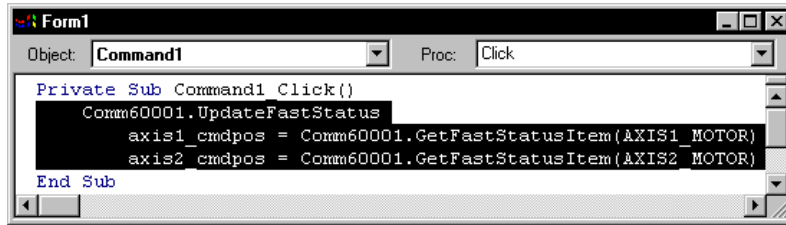
The descriptions and examples provided in this manual are oriented to the Visual Basic 4.0 environment.

### Convention for Code Examples

The code examples presented in this manual represent the code that is placed in a Visual Basic code window. For example, a code example presented as:

```
Comm60001.UpdateFastStatus  
    axis1_cmdpos = Comm60001.GetFastStatusItem(AXIS1_MOTOR)  
    axis2_cmdpos = Comm60001.GetFastStatusItem(AXIS2_MOTOR)
```

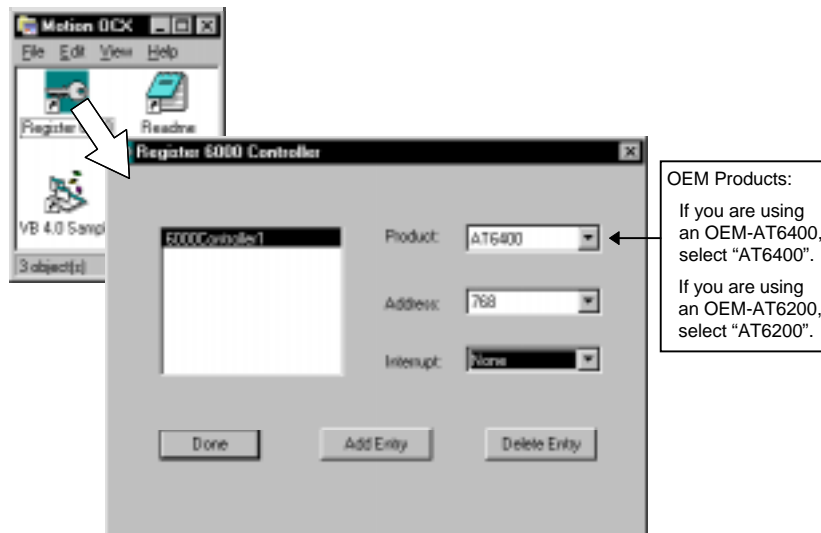
... could be applied to a code window like this (see code in inverse text):



## Basic Operations

### Registering Your 6000 Product

Before using any Motion OCX controls, use the Reg6000 program to register the 6000 Series bus-based product in the Windows Registry. The Reg6000 program is accessible from the Motion OCX program group created during the installation process (see illustration below).



Launch Reg6000, click the “Add Entry” button, and fill in the product type and address, and interrupt settings for the 6000 product you are using (repeat for each product if you have more than one). The address and interrupt settings are established with DIP switches during the product installation process (refer to your product’s *Installation Guide* for instructions). The default settings for 6000 Series controllers are: Address = 768 (decimal); Interrupt = None.

#### NOTES

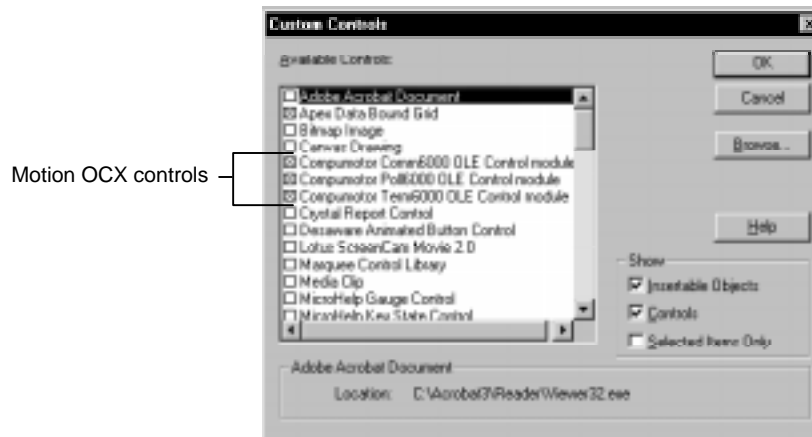
- If you are developing a program that will be distributed, the OCX controls will automatically prompt the user to register this information if it has not yet been entered on the computer.
- If you register Motion OCX as having an interrupt, any attempt to use the same interrupt in another application (e.g., Motion Architect) will cause an interrupt conflict error message. To use the interrupt in another program, run Reg6000 and change the Motion OCX interrupt setting to “none.”
- If you add a 6000 controller or change the address or interrupt of an existing controller, you must redo the registration.

## Using Motion OCX Controls in Visual Basic 4.0

*NOTE: These tasks are also demonstrated in the Tutorial on page 29.*

### **To add the Motion OCX controls to the Toolbox:**

From the *Tools* menu, select *Custom Controls*. Check the box to the left of each Motion OCX control: “Compumotor Comm6000”, “Compumotor Poll6000”, and “Compumotor Term6000”.



The Motion OCX controls will then be added to the control Toolbox:



... Toolbox icon for the Communications Shell OCX (*Comm6000*)



... Toolbox icon for the Terminal OCX (*Term6000*)



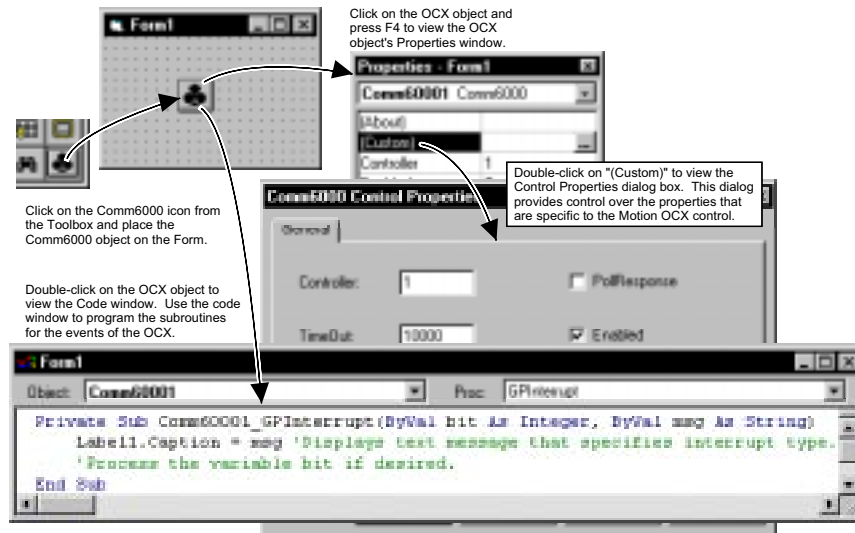
... Toolbox icon for the Fast-Status Polling OCX (*Poll6000*)

### To implement a Motion OCX control:

After you place a Motion OCX control onto the Form, you can implement the Motion OCX features as follows:

- Double-click the “(Custom)” item in the Properties window to view the custom properties dialog. This dialog provides control over all the properties specific to the Motion OCX control (see example below).
- Double-click on an OCX (on the Form) to bring up the Code window you can use to program the subroutines for the events of the OCX (see example below).

Example (using the *Comm6000* OCX):



## Getting Online Help

To get help on the control, press the F1 key when the focus is on the control (i.e., click on the control icon or object and press F1).



## Auto-Loading the 6000 Product Operating System

The 6000 product requires its operating system (.OPS) file to be downloaded before it can function. The operating system files for each product are:

AT6200, AT6400, OEM-AT6200 & OEM-AT6400..... AT6400.OPS  
AT6250..... AT6250.OPS  
AT6450..... AT6450.OPS

The OPS file can be installed as part of the Motion Architect installation process (Motion Architect is provided in your 6000 product's ship kit)—the Motion Architect installer places the selected OPS file in the MA6000 directory (by default). The OPS file can also be installed separately from the "Operating System & DOS Support Software" diskette provided in the product's ship kit, in which case the installer places the OPS file in the directory named for the operating system (i.e., AT6400, AT6250 or AT6450).

When an application using a Motion OCX control is run, each control that is enabled will check if the operating system is loaded. If the operating system is not loaded, it will be loaded automatically if the OPS file is in the working

directory of the application. If the OPS file is not in the working directory, a dialog will appear prompting the user to locate the file.

### **To Prevent Auto-Loading the Operating System**

If you do not wish the Motion OCX controls in your application to automatically download the operating system at run time, set each OCX's initial Enable value to False. Then, after the application is launched, set each OCX's Enable value to True. As soon as an OCX is enabled, it will download the operating system if it is not already downloaded.

### **Distributing Applications** *(with Embedded Motion OCX Controls)*

If you intend to create and distribute applications that use the Motion OCX controls, program your installation utility to install the following files as appropriate for the target operating system. *The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.*

- Windows 95:
  - Install the 6000 product's operating system (.OPS file) in the working directory in which you will install your application.
  - Install these files in WINDOWS\SYSTEM:
    - ISACOMM.DLL
    - MFC40.DLL
    - MSVCRT40.DLL
    - OLEPRO32.DLL
    - WNS27.VXD
- Windows NT:
  - Install the 6000 product's operating system (.OPS file) in the working directory in which you will install your application.
  - Install WNS27.SYS in WINDOWS\SYSTEM32\DRIVERS.
  - Install these files in WINDOWS\SYSTEM32:
    - ISACOMM.DLL
    - MFC40.DLL
    - MSVCRT40.DLL
    - OLEPRO32.DLL

The Motion OCX Toolkit distribution diskette automatically installs most of these files in the appropriate directories on your destination drive.

**NOTE:** You do not have to distribute the Reg6000 application for the end-user to register their 6000 product(s) in the Windows Registry. The first time the user runs your application, the embedded Motion OCX controls will automatically prompt the user to fill in the Registration dialog box (see page 3).



# Communications Shell OCX



---

## Comm6000 Overview

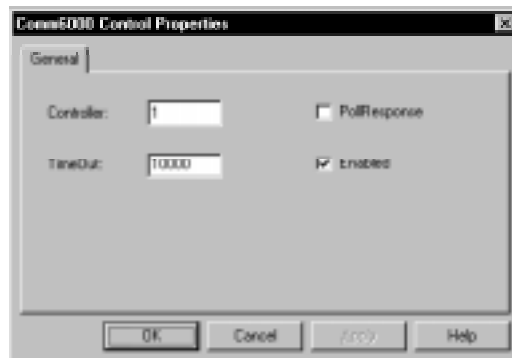
Use the Communications Shell OCX (*Comm6000*) to control basic communication with the 6000 product, including interrupt handling and sending/receiving files.

Page 4 provides general instructions on using this control with Visual Basic. A **tutorial** (starting on page 29) demonstrates how to use all of the Motion OCX controls.

---

## Comm6000 Properties

Visual Basic 4.0: To access the Control Properties dialog box, double-click the “(Custom)” item in the Properties window. This dialog provides control of the properties specific to the Motion OCX control. To customize other properties, use the standard Properties window (accessed by pressing F4 while the OCX object has focus).



## Controller

This is the number of the 6000 Controller that will be used, as registered with the Reg6000 program (see page 3), or with the Register6000 method (see page 12).

## Enabled

The Enabled property is a Boolean, true or false, that enables or disables the control.

## TimeOut

The TimeOut property is the time, in milliseconds, that the Comm6000 OCX control waits for the input buffer to be ready to accept input. This is used in the SendCommand and SendFile methods. The default value is 10 seconds.

## PollResponse

This is a Boolean, true or false, which determines whether or not to poll the 6000 Controller for a response. The PollResponse property should be set to False if using a Term6000 control for the same controller, because if this control is polling for a response from the 6000 Controller, the Term6000 terminal would not get the response.

---

# Comm6000 Methods

## EnableInterrupt

The EnableInterrupt method is necessary to re-enable the general-purpose interrupt when the controller is reset during run-time. If an interrupt is specified in the registry, it is automatically enabled when the Comm6000 OCX is first loaded.

**Example** (in Visual Basic, if the Comm6000 OCX was named Comm60001):

```
Comm60001.EnableInterrupt
```

## GetFastStatusItem

The GetFastStatusItem method returns the value of a specific Fast Status item. The Fast Status offset is the method's parameter and the return value is a long (4 byte) integer.

**Example:** The code in this example first updates the Fast Status register and then places the commanded position of axis 1 and axis 2 ("AXIS1\_MOTOR" and "AXIS2\_MOTOR", respectively) into the variables defined as "axis1\_cmdpos"

and “axis2\_cmdpos”, respectively.

```
Comm60001.UpdateFastStatus
  axis1_cmdpos = Comm60001.GetFastStatusItem(AXIS1_MOTOR)
  axis2_cmdpos = Comm60001.GetFastStatusItem(AXIS2_MOTOR)
```

**NOTE:** The above example assumes that the Fast Status definitions (described below) have been declared in a global .BAS module. (**Tip:** You could use the 6000.BAS module installed in the MOTO CX directory.) If you do not declare the Fast Status definitions, you must enter the Fast Status item’s offset hex value from the table below (e.g., in the above example, you would use “&H0” instead of “AXIS1\_MOTOR” and “&H2” instead of “AXIS2\_MOTOR”).

The table below identifies the offset value and description of each Fast Status item. The syntax of each **Fast Status Item** listed matches the syntax of the definitions in the 6000.BAS module installed in the MOTO CX directory. For more information on the Fast Status register, refer to the Communication chapter in the *6000 Series Programmer’s Guide*. Each Fast Status item returns a 32-bit value.

Fast Status Item	Offset	Description (related status command given - see 6000 Series Software Reference)
AXIS1_MOTOR	&H0	Commanded position, axis 1. See TPM (steppers) or TPC (servos).
AXIS2_MOTOR	&H2	Commanded position, axis 2. See TPM or TPC.
AXIS3_MOTOR	&H4	Commanded position, axis 3. See TPM or TPC.
AXIS4_MOTOR	&H6	Commanded position, axis 4. See TPM or TPC.
AXIS1_ENCODER	&H8	Feedback device position, axis 1. See TPE (steppers) or TFB (servos).
AXIS2_ENCODER	&H0A	Feedback device position, axis 2. See TPE or TFB.
AXIS3_ENCODER	&H0C	Feedback device position, axis 3. See TPE or TFB.
AXIS4_ENCODER	&H0E	Feedback device position, axis 4. See TPE or TFB.
AXIS1_VELOCITY	&H10	Commanded velocity, axis 1. See TVEL.
AXIS2_VELOCITY	&H12	Commanded velocity, axis 2. See TVEL.
AXIS3_VELOCITY	&H14	Commanded velocity, axis 3. See TVEL.
AXIS4_VELOCITY	&H16	Commanded velocity, axis 4. See TVEL.
AXIS1_STATUS	&H18	Axis Status for axis 1. See TAS. *
AXIS2_STATUS	&H1A	Axis Status for axis 2. See TAS. *
AXIS3_STATUS	&H1C	Axis Status for axis 3. See TAS. *
AXIS4_STATUS	&H1E	Axis Status for axis 4. See TAS. *
INPUT_STATUS	&H20	Input Status — general-purpose & trigger inputs. See TIN. *
OUTPUT_STATUS	&H22	Output Status — general-purpose & auxiliary inputs. See TOUT. *
LIMIT_STATUS	&H24	Limit Status — hardware end-of-travel and home limits. *
INO_STATUS	&H25	Other Input Status — joystick, and P-CUT or ENBL. See TINO. *
ANALOG_STATUS	&H26	Status of joystick analog channel voltage. See TANV.
INT_STATUS	&H28	Status of hardware interrupt conditions. See TINT. *
SYSTEM_STATUS	&H2A	System status. See TSS. *
USER_STATUS	&H2C	User status. See TUS. *
TFRM_STATUS	&H2D	Time frame mark status. Updated every 2 ms for steppers, or every “system update” for servos (see SSFR table). Starts on computer power-up. Rolls over.
TIMER_STATUS	&H2E	Status of programmable timer. See TIMST.

\* Bit-wise fast status item — see pages 22-25 for bit descriptions.

## ReceiveFile

This method is used to upload programs from the 6000 controller into a file.

### Example:

```
Comm60001.ReceiveFile
```

## Register6000

The Register6000 method provides a dialog box (see example at right) which has entries for the specific 6000 product, address and optional interrupt. Controller entries can be added, deleted, or changed. This information is stored in the Windows Registry and is necessary for the Motion OCX controls to function.



### Example:

```
Comm60001.Register6000
```

## SendCommand

This method is used to send a command string to the 6000 controller.

### Example (TREV is a 6000 command that displays the product's firmware revision):

```
chars_sent = Comm60001.SendCommand("TREV")
```

### Example (substituting a string variable for "TREV"):

```
Dim cmd As String  
cmd = "TREV"  
chars_sent = Comm60001.SendCommand(cmd)
```

If successful, this method returns an integer value (4 bytes) that represents the number of characters sent (includes an appended command delimiter and is rounded up to the nearest even number). If not successful, one of the following error codes will be returned (and a relevant error message will be displayed):

- 1 ..... operating system not loaded
- 2 ..... timeout (see note below on avoiding a timeout error)

## AVOIDING A TIMEOUT ERROR

These two conditions can cause a Timeout Error\*:

- Not checking for a response causes the 6000 product's output buffer to become full. To avoid the Timeout Error, check for a response by using one (NOT BOTH) of these methods:
  - Place a Term6000 terminal window in your application.
  - Set the Comm6000 PollResponse property (page 10) to True.
- Sending down buffered commands during a long move causes the 6000 product's input buffer to become full.

\* The TimeOut property (see page 10) establishes the time (milliseconds), that the Comm6000 OCX control waits for the input buffer to be ready to accept input. If this wait period is exceeded, a Timeout Error occurs.

## SendFile

Use this method to download a file to the 6000 controller.

**Example:**

```
result = Comm60001.SendFile("prog1.prg")
```

To select the file at run-time (prompt with dialog box), send a null string:

```
result = Comm60001.SendFile(" ")
```

If successful, a positive integer value will be returned. If not successful, one of the following error codes will be returned (and a relevant error message will be displayed):

- 1 ..... operating system not loaded
- 2 ..... timeout (see note above on avoiding a timeout error)
- 3 ..... could not open file

## UpdateFastStatus

The UpdateFastStatus method updates the entire 6000 Fast Status area. See page 10 for more information on the Fast Status register.

**Example:**

```
Comm60001.UpdateFastStatus  
axis1_cmdpos = Comm60001.GetFastStatusItem(Axis1_MOTOR)  
axis2_cmdpos = Comm60001.GetFastStatusItem(Axis2_MOTOR)
```

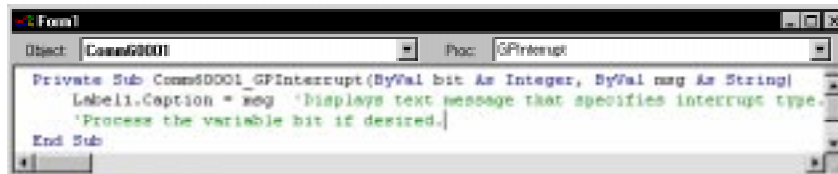
Note that the Fast Status region was first updated by the UpdateFastStatus method, and then the GetFastStatusItem method was used.

# Comm6000 Events

## GPInterrupt

This event is triggered when the 6000 controller experiences a general-purpose interrupt. It essentially provides a user programmable interrupt service routine. The bit corresponding to the TINT command value (see description in the *6000 Series Software Reference*), as well as an explanatory message, are passed. When an interrupt is triggered, the controller is put into ERRLVLO mode (no prompt is returned, no error responses are returned, each transmission is terminated by a line feed only, and status responses are returned without the asterisk or the status command), and the Fast Status area is updated.

Example:



```
Form1
Object: Comm6000
Proc: GPInterrupt

Private Sub Comm6000_GPInterrupt(ByVal bit As Integer, ByVal msg As String)
    Label1.Caption = msg 'Displays text message that specifies interrupt type.
    'Process the variable bit if desired.
End Sub
```

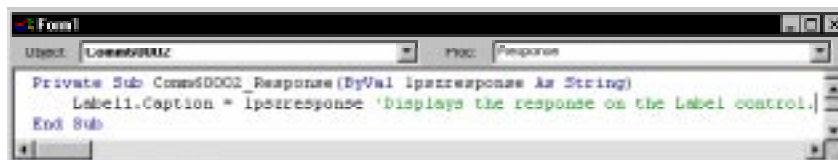
### NOTES ABOUT USING INTERRUPTS

- Motion OCX's interrupt performance is subject to:
  - Machine speed
  - Other programs currently running
  - Frequency of the OCX interrupt
  - Overhead of tasks in the interrupt service routine (ISR)
- After establishing interrupts in Motion OCX, any attempt to set an interrupt in Motion Architect will cause an interrupt conflict error message, even though the Comm6000 OCX is not in use at the same time. To use the interrupt under Motion Architect, run Reg6000 (see page 3) and change the Motion OCX interrupt setting to "none."

## Response

If the PollResponse property (see page 10) is True, the Response event will be triggered when the 6000 Controller has a message.

Example:



```
Form1
Object: Comm6000
Proc: Response

Private Sub Comm6000_Response(ByVal ipzresponse As String)
    Label1.Caption = ipzresponse 'Displays the response on the Label control.
End Sub
```

# Terminal OCX



---

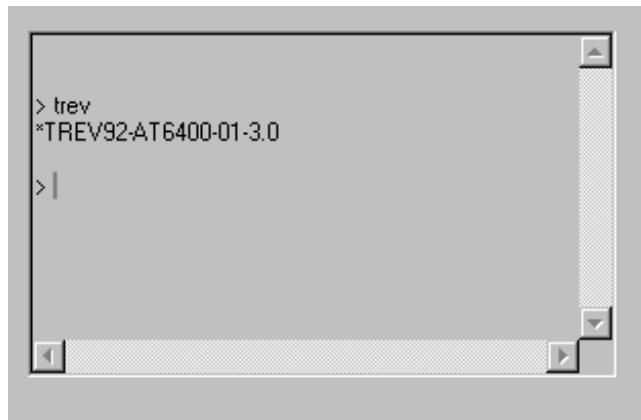
## Term6000 Overview

Use the active communication interface in the Terminal OCX (*Term6000*) to type in commands to send to the 6000 product, check program responses and status reports, and view error messages.

Page 4 provides general instructions on using this control with Visual Basic. A **tutorial** (starting on page 29) demonstrates how to use all of the Motion OCX controls.

NOTE: If you are using Visual C++, terminate the command line with a colon (: ) instead of a carriage return.

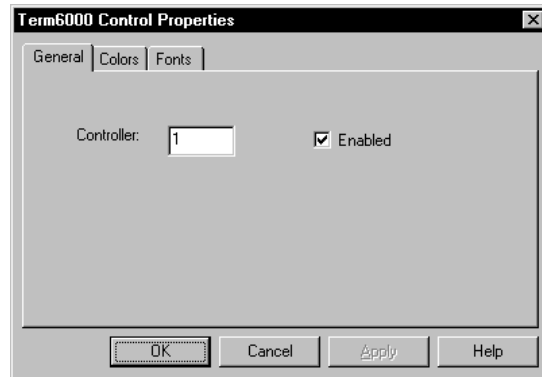
**Example** (Term6000 terminal emulator window with scroll bars):



---

# Term6000 Properties

Visual Basic 4.0: To access the Control Properties dialog (shown below), double-click the “(Custom)” item in the Properties window. This dialog provides control of the properties specific to the Motion OCX control. To customize other properties, use the standard Properties window (accessed by pressing F4 while the OCX object has focus).



## BackColor

Set the background color of the terminal window with this property.

## Controller

This is the number of the 6000 Controller that will be used, as registered with the Reg6000 program (see page 3), or with the Register6000 method in the Comm6000 OCX (see page 12).

## Enabled

The Enabled property is a Boolean, true or false, that enables or disables the control.

## Font

Set this property to the desired terminal window font.

## ForeColor

Set the text color of the terminal window with this property.

---

# Term6000 Methods

## Clear

Use this method to clear all text in the terminal window.

**Example:**

```
Term60001.Clear
```

## Copy

Use this method to copy the contents of the terminal window to the clipboard.

**Example:**

```
Term60001.Copy  
Edit1.Text = Clipboard.GetText
```

This code will transfer the contents of the terminal window to the edit control.



# Fast Status Polling OCX



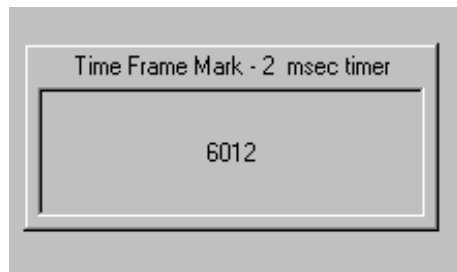
---

## Poll6000 Overview

Use the Fast Status Polling OCX (*Poll6000*) to poll the 6000 product's fast status register and display information such as position, velocity, axis status, system status, etc.

Page 4 provides general instructions on using this control with Visual Basic. A **tutorial** (starting on page 29) demonstrates how to use all of the Motion OCX controls.

**Example** (Poll6000 displays the time frame mark):



---

## Poll6000 Properties

Visual Basic 4.0: To access the Control Properties dialog (shown at right), double-click the “(Custom)” item in the Properties window. This dialog provides control of the properties specific to the Motion OCX control. To customize other properties, use the standard Properties window (accessed by pressing F4 while the OCX object has focus).



### BackColor

Set the background color of the display window with this property.

### Caption

The Caption property is used for putting a title in the status display window.

### CaptionPosition

The Caption (title) can be set to top, bottom, left, or right with the CaptionPosition property.

### Controller

This is the number of the 6000 Controller that will be used, as registered with the Reg6000 program (see page 3), or with the Register6000 method in the Comm6000 OCX (see page 12).

### Enabled

Set the Enabled property True or False to enable or disable the Poll6000 control.

### Font

Set the Font property to the desired terminal window font.

### ForeColor

Set the text color of the terminal window with this property.

## InnerBorder

The InnerBorder property of the status display window can be set to none, raised, or sunken.

## Item

The item property selects the FastStatus information that will be continually polled and updated. It is an integer value (0-32) that corresponds as described in the table below. Refer to the *6000 Series Programmer's Guide* for more information on using the Fast Status register.

Integer	Description (related status command given – see <i>6000 Series Software Reference</i> )
0	Commanded position, axis 1. See <i>TPM</i> (steppers) or <i>TPC</i> (servos).
1	Commanded position, axis 2. See <i>TPM</i> or <i>TPC</i> .
2	Commanded position, axis 3. See <i>TPM</i> or <i>TPC</i> .
3	Commanded position, axis 4. See <i>TPM</i> or <i>TPC</i> .
4	Feedback device position, axis 1. See <i>TPE</i> (steppers) or <i>TFB</i> (servos).
5	Feedback device position, axis 2. See <i>TPE</i> or <i>TFB</i> .
6	Feedback device position, axis 3. See <i>TPE</i> or <i>TFB</i> .
7	Feedback device position, axis 4. See <i>TPE</i> or <i>TFB</i> .
8	Commanded velocity, axis 1. See <i>TVEL</i> .
9	Commanded velocity, axis 2. See <i>TVEL</i> .
10	Commanded velocity, axis 3. See <i>TVEL</i> .
11	Commanded velocity, axis 4. See <i>TVEL</i> .
12	Axis Status for axis 1. See <i>TAS</i> . *
13	Axis Status for axis 2. See <i>TAS</i> . *
14	Axis Status for axis 3. See <i>TAS</i> . *
15	Axis Status for axis 4. See <i>TAS</i> . *
16	Input Status — general-purpose & trigger inputs. See <i>TIN</i> . *
17	Output Status — general-purpose & auxiliary inputs. See <i>TOUT</i> . *
18	Limit Status — hardware end-of-travel and home limits. *
19	Other Input Status — joystick, and P-CUT or ENBL. See <i>TINO</i> . *
20	Voltage of analog channel 1 on joystick connector. See <i>TANV</i> .
21	Voltage of analog channel 2 on joystick connector. See <i>TANV</i> .
22	Voltage of analog channel 3 on joystick connector. See <i>TANV</i> .
23	Voltage of analog channel 4 on joystick connector. See <i>TANV</i> .
24	Status of hardware interrupt conditions. See <i>TINT</i> . *
25	System status. See <i>TSS</i> . *
26	User status. See <i>TUS</i> . *
27	Time frame mark status. Updated every 2 ms for steppers, or every “system update” for servos (see <i>SSFR</i> table). Starts on computer power-up. Rolls over.
28	Status of programmable timer. See <i>TIMST</i> .
29	Variable 11 — value of numeric variable 11 ( <i>VAR11</i> ). **
30	Variable 12 — value of numeric variable 12 ( <i>VAR12</i> ). **
31	Variable 13 — value of numeric variable 13 ( <i>VAR13</i> ). **
32	Variable 14 — value of numeric variable 14 ( <i>VAR14</i> ). **

\* Fast Status item has multiple status bits — see pages 22-25 for descriptions.

\*\* If you reset the 6000 controller, the FastStatus data for the variable will not be updated until you reset the Poll6000 item property to the variable to be polled.

## OuterBorder

Set the display window's OuterBorder property to none, raised, or sunken.

## ScaleFactor

The ScaleFactor property allows scaling of velocity and position. The default value is 1 (no scaling).

## StatusBit

If a bit-wise FastStatus item has been selected, this property indicates which bit to check for updates. Refer to tables below for bit functions.

### Fast Status Bit Descriptions — Axis Status

Bit #	Function ("On" means true, "Off" means false)
1	Axis is moving
2	Direction is negative
3	Accelerating
4	Moving at commanded velocity
5	Homing (HOM) was successful
6	Absolute positioning mode (MA1) is enabled <off means incremental mode>
7	Continuous positioning mode (MC1) is enabled <off means preset mode>
8	Jog mode (JOG) is enabled
9 *	Joystick mode (JOY) is enabled
10 *	Encoder Step mode (ENC) is enabled — stepper products only
11 *	Position Maintenance mode (EPM) is enabled — stepper products only
12 *	Stall Detected (ESTALL) — stepper products only
13 *	Drive is shut down (DRIVE)
14 *	Drive fault occurred (not detected until you enable input functions – INFEN1)
15	Positive-direction hardware end-of-travel limit was encountered
16	Negative-direction hardware end-of-travel limit was encountered
17	Positive-direction software end-of-travel limit (LSPOS) was encountered
18	Negative-direction software end-of-travel limit (LSNEG) was encountered
19	Within Deadband (EPMDB)
20	In Position (COMEXP) — stepper products only
21	Distance Streaming mode (STREAM1) is enabled — stepper products only
22	Velocity Streaming mode (STREAM2) is enabled — stepper products only
23	Position error limit (SMPER) is exceeded — servo products only
24 **	Load is within Target Zone (STRGTD & STRGTV)
25	Target Zone timeout occurred (STRGTT)
26 ***	Motion is suspended, pending a GOWHEN condition
27	<i>reserved</i>
28 ****	A Registration move was initiated by a trigger since the last GO command
29	<i>reserved</i>
30	Profile for an attempted pre-emptive GO or Registration move was impossible
31	<i>reserved</i>
32	<i>reserved</i>

\* Bit is not applicable to the OEM-AT6200 and OEM-AT6400 products.

\*\* Bit is set to "On" only after successful completion of move.

\*\*\* Bit is set to "Off" when the GOWHEN condition is true, or if !K or !S is executed.

\*\*\*\* Bit is set to "Off" when the next GO command is executed.

### Fast Status Bit Descriptions — Input Status

AT6200:

- Bits 1-24 are general-purpose inputs 1-24.
- Bits 25 & 26 are trigger inputs TRG-A and TRG-B.

AT6400 and AT6450:

- Bits 1-24 are general-purpose inputs 1-24.
- Bits 25-28 are trigger inputs TRG-A, TRG-B, TRG-C and TRG-D.

AT6250:

- Bits 1-24 are general-purpose inputs 1-24.
- Bits 25-27 are trigger inputs TRG-A, TRG-B and TRG-C.

OEM-AT6200 and OEM-AT6400:

- Bits 1-6 are general-purpose inputs 1-6.
- Bits 7-10 are trigger inputs TRG-A, TRG-B, TRG-C and TRG-D.

### Fast Status Bit Descriptions — Output Status

AT6200 and AT6400: Bits 1-24 are general-purpose outputs 1-24.

AT6250:

- Bits 1-24 are general-purpose outputs 1-24.
- Bits 25-27 are auxiliary outputs OUT-A, OUT-B and OUT-C.

AT6450:

- Bits 1-24 are general-purpose outputs 1-24.
- Bits 25-28 are auxiliary outputs OUT-A, OUT-B, OUT-C and OUT-D.

OEM-AT6200 and OEM-AT6400: Bits 1-4 are general-purpose outputs 1-4.

### Fast Status Bit Descriptions — Limit Status

Bit #	Function
-------	----------

1	Axis #1 positive-direction hardware end-of-travel limit *
2	Axis #1 negative-direction hardware end-of-travel limit *
3	Axis #2 positive-direction hardware end-of-travel limit *
4	Axis #2 negative-direction hardware end-of-travel limit *
5	Axis #3 positive-direction hardware end-of-travel limit *
6	Axis #3 negative-direction hardware end-of-travel limit *
7	Axis #4 positive-direction hardware end-of-travel limit *
8	Axis #4 negative-direction hardware end-of-travel limit *
9	Axis #1 home limit **
10	Axis #2 home limit **
11	Axis #3 home limit **
12	Axis #4 home limit **

\* If using the default active level and wiring (LHLVLO and n.c. limit switch), an open contact sets the bit status to "Off" and a closed contact sets the bit status to "On".

\*\* If using the default active level and wiring (HOMLVLO and n.o. home switch), an open contact sets the bit status to "On" and a closed contact sets the bit status to "Off".

### Fast Status Bit Descriptions — “Other” Input Status \*

Bit #	Function (“On” means true, “Off” means false)
1	Joystick Auxiliary Input is active (input is on Joystick connector, pin 19)
2	Joystick Trigger Input is active (input is on Joystick connector, pin 18)
3	Joystick Axes Select Input is active (input is on Joystick connector, pin 15)
4	Joystick Velocity Select input is high (input is on Joystick connector, pin 16)
5	Joystick Release Input is active (input is on Joystick connector, pin 17)
6	P-CUT input (steppers) or ENBL input (servos) is connected to ground. This allows motion to occur.

\* This fast status item is not applicable to the OEM-AT6200 and OEM-AT6400 products.

### Fast Status Bit Descriptions — Interrupt Status

Bit #	Function (“On” means true, “Off” means false)
1 *	Software interrupt #1 occurred
2 *	Software interrupt #2 occurred
3 *	Software interrupt #3 occurred
4 *	Software interrupt #4 occurred
5 *	Software interrupt #5 occurred
6 *	Software interrupt #6 occurred
7 *	Software interrupt #7 occurred
8 *	Software interrupt #8 occurred
9 *	Software interrupt #9 occurred
10 *	Software interrupt #10 occurred
11 *	Software interrupt #11 occurred
12 *	Software interrupt #12 occurred
13 *	Software interrupt #13 occurred
14 *	Software interrupt #14 occurred
15 *	Software interrupt #15 occurred
16 *	Software interrupt #16 occurred
17	Command buffer is full
18 **	P-CUT input (steppers) or ENBL input (servos) activated
19	Program complete
20 **	Drive fault occurred, any axis
21	<i>reserved</i>
22	<i>reserved</i>
23	An end-of-travel limit (hardware or software) was encountered, any axis
24 **	Stall detected (steppers) or Excessive position error (servos), any axis
25	Timer (TIMINT)
26 **	Counter (CNTINT) — stepper products only
27	“PC Interrupt” input (a general purpose input defined by INFNCi-I) activated
28	Command error occurred
29	Motion complete on axis #1
30	Motion complete on axis #2
31	Motion complete on axis #3
32	Motion complete on axis #4

\* The software interrupt bit is first enabled with INTHW and executed with INTSW.  
This bit remains set to “On” until you send the TINT command to the 6000 product.

\*\* Bit is not applicable to the OEM-AT6200 and OEM-AT6400 products.

### Fast Status Bit Descriptions — System Status

Bit #	Function ("On" means true, "Off" means false)
1	System is ready
2	<i>reserved</i>
3	Executing a program
4	Last command was an immediate command (prefixed by !)
5	ASCII mode enabled
6	<i>reserved</i>
7	Defining a program
8	Trace mode enabled (TRACE)
9	Single-Step mode enabled (STEP)
10	Translation mode enabled (TRANS)
11	Command error occurred (bit is cleared with TCMDEF command)
12	Break Point (BP) active
13	Pause active
14	Wait (WAIT command) active
15	Monitoring program interrupt conditions (ONCOND)
16	Waiting for data (READ)
17	Loading thumbwheel data (TW)
18	External Program Select mode enabled (INSELP)
19	Dwell (T command) in progress
20	<i>reserved</i>
21	<i>reserved</i>
22	<i>reserved</i>
23	Servo data-gathering transmission in progress — servo products only
24	<i>reserved</i>
25 *	Position was captured with trigger A (TRG-A)
26 *	Position was captured with trigger B (TRG-B)
27 *	Position was captured with trigger C (TRG-C)
28 *	Position was captured with trigger D (TRG-D)
29	Compiled memory is 75% full
30	Compiled memory is 100% full
31	Compile operation (PCOMP command) failed — cleared on power up, RESET, or after successful PCOMP. See PCOMP command for possible causes.
32	<i>reserved</i>

\* Bits 25-28 are cleared when the captured position is read with one of these commands: CA, PCA, PCC, PCE, PCM, TCA, TPCA, TPCC, TPCE, or TPCM.

### Fast Status Bit Descriptions — User Status

The User Status comprises 16 bits (numbered 1-16) that are user-defined with the INDUST command. Depending on the INDUST configuration, the status bit definitions could be any combination of status bits from the Axis Status, the System Status, the Input Status, or the Interrupt Status.

### **StatusBitOffText**

If displaying a bit-wise item, set the StatusBitOffText property to the text that should be displayed when the bit is “Off” (i.e., 0 or false).

### **StatusBitOnText**

If displaying a bit-wise item, set the StatusBitOnText property to the text that should be displayed when the bit is “On” (i.e., 1 or true).

### **TextAlignment**

The status update value can be centered, aligned to the left of the window, or to the right of the window with the TextAlignment property.

### **UpdateRate**

The UpdateRate property sets the polling interval, in milliseconds, for the selected FastStatus item.

### **Value**

This is the value of the fast status item that is polled. It is read only at run time.

---

## Poll6000 Methods

### Update

The Update method forces an immediate poll. It should be used when the polling window Visible property is set to false, because then the Poll6000 internal timer will not work.

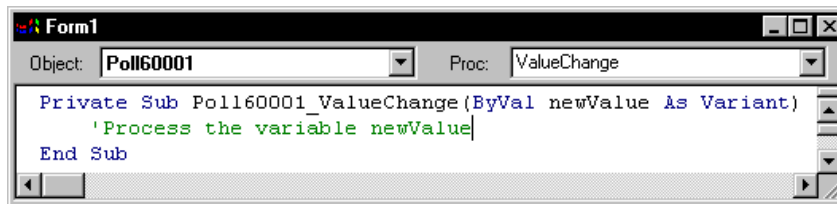
---

## Poll6000 Events

### ValueChanged

When the value of the Fast Status item has changed, the ValueChange event notifies the control container, and passes a *variant* value.

Example:



```
Form1
Object: Poll60001 Proc: ValueChange
Private Sub Poll60001_ValueChanged (ByVal newValue As Variant)
    'Process the variable newValue
End Sub
```

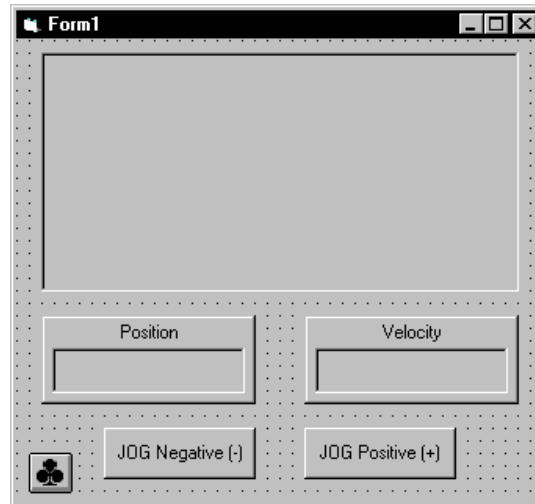


# Appendix A: *Tutorial*

This tutorial leads you through the process of developing a sample application and demonstrates how to implement the basic functionality of each Motion OCX control: Comm6000, Term6000, and Poll6000. [This application is tailored to operate axis #1 of the AT6400 product.](#)

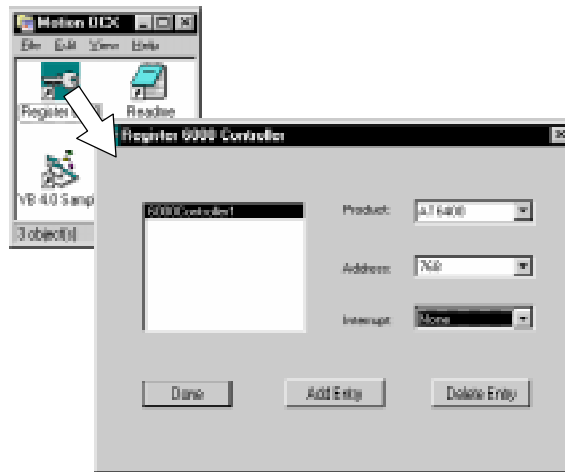
 <b>WARNING</b> 
This sample application disables the hardware end-of-travel limits. Therefore, before using the JOG buttons, you must ensure that moving the motor will not damage equipment and/or injure personnel. To help ensure safety, uncouple the motor from the load before running this application.

When you have completed the procedures below, your project should resemble the one shown here:



## A. Setup

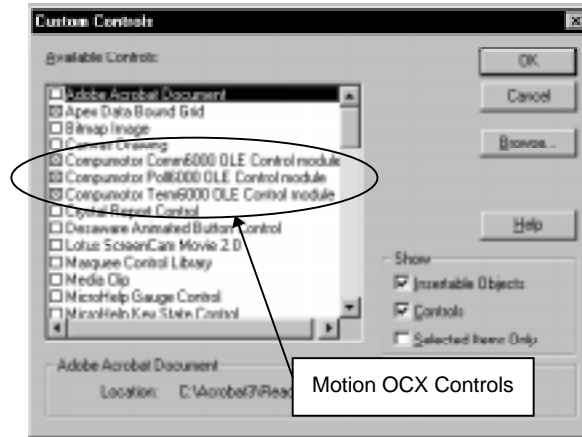
1. Make sure your 6000 product is properly installed and tested according to the product's *Installation Guide*.
2. Install Motion OCX Toolkit on your hard drive (see page 2).
3. Use the Reg6000 program to register your 6000 Series product in the Windows Registry:
  - a. Launch the Reg6000 program from the Motion OCX program group (see example in illustration below).
  - b. Click the **Add Entry** button, and fill in the Product type, Address, and Interrupt settings for your 6000 product. Use the settings you established when you installed your 6000 product (see *Installation Guide*). The default address setting for 6000 products is 768, and the default interrupt setting is None.






- c. Press the **Done** button when you are finished with registration.
4. Launch Visual Basic 4.0. When it launches, it automatically opens a new project.

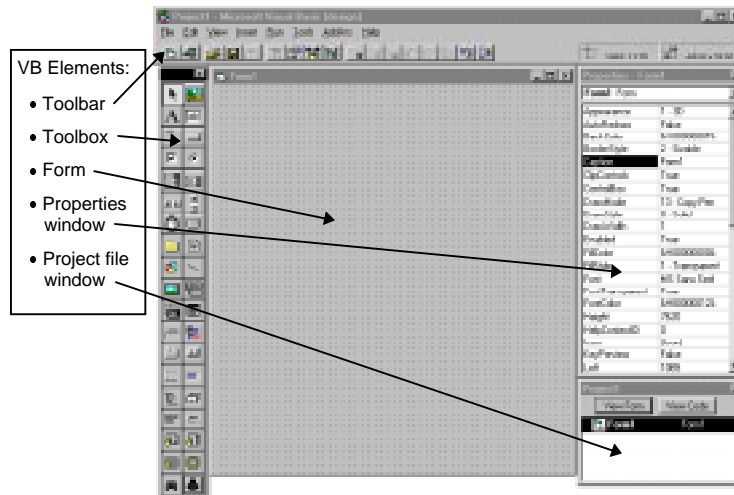
5. Add the Motion OCX controls to the VB Toolbox (see illustration below):

- a. Navigate to the **Tools** menu and select **Custom Controls**.
- b. Select: “Compumotor Comm6000 OLE Control module”  
“Compumotor Poll6000 OLE Control module”  
“Compumotor Term6000 OLE Control module”




- c. Click **OK** when you are finished. You should now see these Motion OCX controls added to the VB Toolbox:

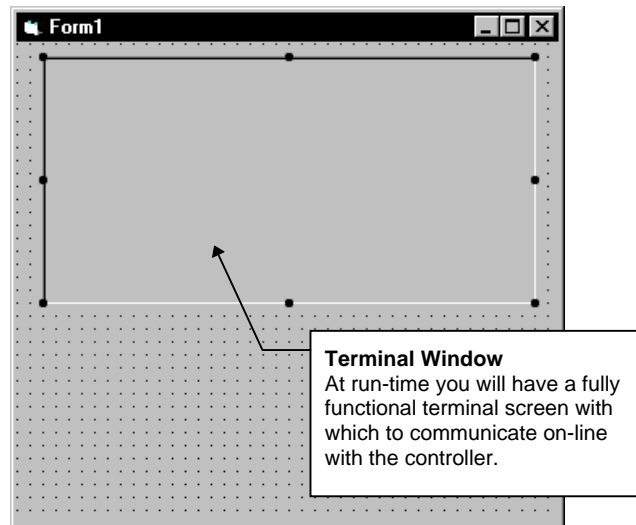
-  .... Communications Shell OCX (*Comm6000*)
-  .... Terminal OCX (*Term6000*)
-  .... Fast-Status Polling OCX (*Poll6000*)



## B. Programming

### **Term6000: Terminal OCX**


To get a working terminal within your program (like the one in Motion Architect), click on the Term6000 control (  ) from the VB Toolbox and drag open the window within the Form (see example below):

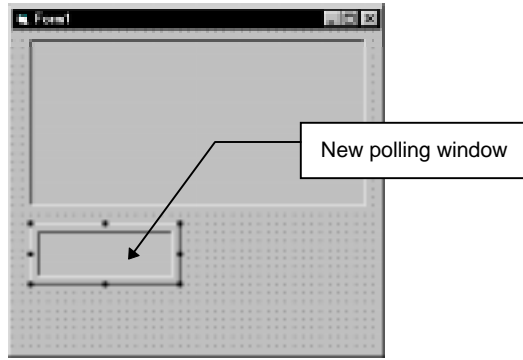


You have now completed placing a terminal window in your application.

### **Poll6000: Fast Status Polling OCX**

Overview: The purpose of the procedure below is to create two fast status polling windows: one to display the commanded position, and the other to display the commanded velocity. (“Commanded” means the position or velocity that the 6000 product is commanding to the drive, not the position or velocity as measured with an attached feedback device.)

1. From the Toolbox, select the Poll6000 control (  ) and drag open a polling window within the Form (place it below the Term6000 terminal window as shown below).



2. Click on the new polling window you just created and press the F4 key to view the control's Properties window. In the Properties window, double-click on the "(Custom)" property to view the Control Properties dialog box. Set the **Item** property to "Axis 1 Position" and set the **Caption** property to "Position". Click **OK** when you're done. See illustration below.

Click on the OCX object and press F4 to view the OCX object's Properties window.

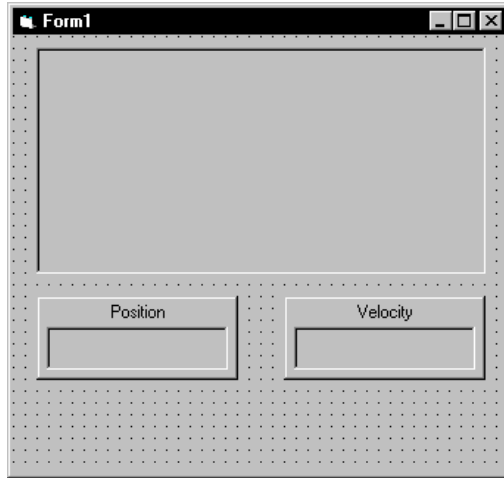
The screenshot shows a 'Properties - Form1' window with a dropdown menu set to 'Poll60001 Poll6000'. Below it are buttons for '(About)' and '(Custom)'. An arrow points from the '(Custom)' button to a 'Poll6000 Control Properties' dialog box. The dialog box has tabs for 'General', 'Display', 'Fonts', and 'Colors'. The 'General' tab is active, showing fields for 'Controller' (set to '1'), 'Update Rate' (set to '110'), 'Caption' (set to 'Position'), and 'Scale Factor' (set to '25000'). There are also fields for 'Status Bit Off Text' (set to 'Off') and 'Status Bit On Text' (set to 'On'). A checkbox labeled 'Enabled' is checked. At the bottom are buttons for 'OK', 'Cancel', 'Apply', and 'Help'. A text box on the right provides instructions on how to set the 'Scale Factor'.

Double-click on "(Custom)" to view the Control Properties dialog box. This dialog provides control over the properties that are specific to the Motion OCX control.

To display in units other than steps or feedback device counts, enter the appropriate scale factor within the **Scale Factor** property field. For example, if using an AT6400 controller and a ZETA drive set for 25000 steps/rev, enter 25000 for Scale Factor to view the position in revs; if using an AT6450, use your encoder resolution for the Scale Factor. This example is set up for 25000 steps/rev.


3. Repeat steps 1 & 2 for the second polling window, but with these exceptions: in the Properties window, set the **Item** property to "Axis 1 Velocity" and set the **Caption** property to "Velocity".

4. On your Form, you should now have a terminal window and two polling status windows for “Position” and “Velocity”. See example below.



### **Comm6000: Communications Shell OCX**

**Overview:** The purpose of the procedure below is to create two JOG buttons, one to move the motor in the negative-counting direction, and the other to move the motor in the positive-counting direction. These two buttons require the presence of the Comm6000 control in the Form.


1. Select the Comm6000 OCX control (  ) from the Toolbox and place it on the Form.
2. Double-click in an empty part of the Form window (not on a control object) to view the code window for the Form. Under the **Proc:** pull-down menu, select **Load**. In the code window (see example shown below), type `chars_sent = Comm60001.SendCommand( "a100:v1:mc1:lh0:drive1" )` so that when you run the application, it automatically sends these 6000 Series commands to the 6000 product to prepare for jogging the motor:

a100 ..... Sets the acceleration to 100 revs/sec/sec  
v1 ..... Sets the velocity to 1 rev/sec  
mc1 ..... Selects continuous positioning mode  
lh0 ..... Disables the hardware end-of-travel limits  
drive1 ..... Enables the drive

(For more information on using the 6000 programming language, refer to the *6000 Series Programming Guide* and the *6000 Series Software Reference* provided in your 6000 product ship kit.)

```
Form1
Object: Form Proc: Load
Private Sub Form_Load()
    chars_sent = Comm60001.SendCommand("a100:v1:mc1:lh0:drive1")
End Sub
```

*Close the code window when you are finished coding.*

3. Select a Command Button (  ) from the Toolbox and place it on the Form (under the Position or the Velocity display window). This button will be used to JOG the motor.
4. Double-click on the button to view its code window. Under the **Proc:** pull-down menu, select **MouseDown**. In the code window type `chars_sent = Comm60001.SendCommand("d-: go1")` so that when the button is pressed, the controller commands the motor to move in the negative-counting direction for as long as the button is pressed down.  
The `d-` command sets the direction to negative and the `go1` command executes motion. As in this case, when multiple 6000 commands are used in the same command line, you must place a colon (`:`) between them.

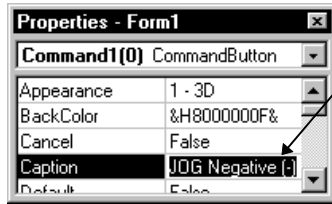
```
Form1
Object: Command1 Proc: MouseDown
Private Sub Command1_MouseDown(Index As Integer, Button As Integer)
    chars_sent = Comm60001.SendCommand("d-: go1")
End Sub
```


5. Remaining in the button's code window, return to the button's **Proc:** menu and select **MouseUp**. In the code window type `chars_sent = Comm60001.SendCommand("!s")` so that when the button is released, the controller commands the motor to stop (`!s` command stops motion):

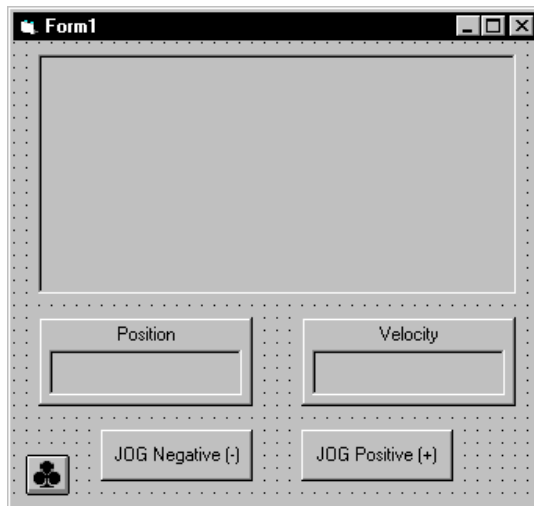
```
Form1
Object: Command1 Proc: MouseUp
Private Sub Command1_MouseUp(Index As Integer, Button As Integer)
    chars_sent = Comm60001.SendCommand("!s")
End Sub
```

*Close the code window when you are finished coding.*

6. In the button's Properties window (press F4 if you can't find it), change the **Caption** property to "JOG Negative (-)":



7. Repeat steps 3-6 for the second JOG button, but with these exceptions:
- Step 4: Within the **MouseDown** procedure, type `chars_sent = Comm60001.SendCommand("d+: go1")` so that the motor moves in the positive-counting direction while the button is pressed down (the difference from the code for the other button is that this button uses `d+` and the other button uses `d-`).
  - Step 6: Set the button Caption property to "JOG Positive (+)".
8. You should now have added two buttons on your Form (in addition to the Comm6000 control icon—): one button labeled "JOG Negative (-)" and one button labeled "JOG Positive (+)". See example below.




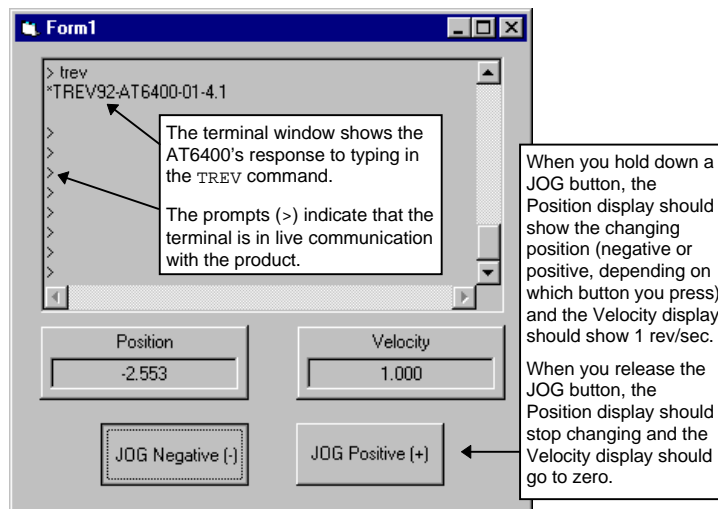
## C. Save Your Project and Test Its Operation

1. To save your application, select **Save Project** from the **File** menu. Note that the default “Save As” target directory is Microsoft Visual Basic.
  - a. You’ll be prompted with a **Save File As** dialog box to save the Form (the graphical interface you just created). Type in “tutorial.frm” in the **File name** field and click the **Save** button.
  - b. Next, you’ll be prompted with the **Save Project As** dialog box. Type in “tutorial.vbp” in the **File name** field and click **Save**.
  - c. Your sample application is complete.
2. You may now test the application, but heed the following warning:

 **WARNING** 

This sample application disables the hardware end-of-travel limits. Therefore, before using the JOG buttons, you must ensure that moving the motor will not damage equipment and/or injure personnel. To help ensure safety, uncouple the motor from the load before running this application.

- a. To test your new application, press the Start button (  ) on the VB toolbar. The illustration below is an example of how it looks at run time .



The terminal window shows the AT6400's response to typing in the TREV command.

The prompts (>) indicate that the terminal is in live communication with the product.

When you hold down a JOG button, the Position display should show the changing position (negative or positive, depending on which button you press) and the Velocity display should show 1 rev/sec.

When you release the JOG button, the Position display should stop changing and the Velocity display should go to zero.

- b. To stop the test, press the Stop button (  ) on the VB toolbar.

If motion does not occur or the display does not respond as expected when you press a JOG button:

- Double-check the coding and OCX property settings performed earlier in the Programming portion of the tutorial (starting on page 32). If the motion is not as expected, make sure the 6000 commands are entered correctly in the code windows (see steps 2-7 starting on page 34).
- Make sure the **P-CUT** input (AT6200 or AT6400) or the **ENBL** input (AT6250 or AT6450) is connected to ground (**GND**). Motion is not allowed if this connection is broken.
- If you are using a servo controller (AT6250 or AT6450), make sure your feedback device is connected properly and that your system is tuned.
- Refer also to the Troubleshooting section in your product's *Installation Guide* and/or your *6000 Series Programmer's Guide*.

# Appendix B: *Using Controls with Visual C++ Non-Dialog Containers*

## (FOR VISUAL C++ USERS ONLY)

In some applications, such as an SDI or MDI application, you will want to embed a control in a window of the application. The `Create` member function of the wrapper class, inserted by Component Gallery, can create an instance of the control dynamically, without the need of a dialog box.

The `Create` member function has the following parameters:

- `lpzWindowName` ...A pointer to the text to be displayed in the control's `Text` or `Caption` property (if any).
- `dwStyle` .....Windows styles. For a complete list, see `CWnd::CreateControl`.
- `rect` .....Specifies the control's size and position.
- `pParentWnd` .....Specifies the control's parent window, usually a `CDialog`. It must not be `NULL`.
- `nID` .....Specifies the control ID and can be used by the container to refer to the control.

### Example

One example of using this function to dynamically create an OLE control would be in a form view of an SDI application. You could then create an instance of the control in the `WM_CREATE` handler of the application.

For this example, `CMyView` is the main view class, `CComm6000` is the wrapper class, and `Comm6000.H` is the header (.H) file of the wrapper class.

**Example** *(continued)*

Implementing this feature is a four-step process:

1. Insert `Comm6000.H` in `CMYVIEW.H`, just before the `CMyView` class definition:

```
#include "Comm6000.h"
```

2. Add a member variable (of type `CComm6000`) to the protected section of the `CMyView` class definition located in `CMYVIEW.H`:

```
class CMyView : public CView
{
...
protected:

CComm6000 comm6000;
};
```

3. Add a `WM_CREATE` message handler to class `CMyView`.
4. In the handler function, `CMyView::OnCreate`, make a call to the control's `Create` function using this pointer as the parent window:

```
int CMyView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (MyView::OnCreate(lpCreateStruct) == -1)
        return -1;

    // ***** Add your code below this line ***** //
    comm6000.Create (NULL, 0, CRect(0,0,0,0), this, 0);
    // ***** Add your code above this line ***** //

    return 0;
}
```

Rebuild the project. A `Comm6000` control will be created dynamically whenever the application's view is created.