

Eggert Electronics Engineering

Daniel N. Eggert • 125 Pasa Por Aqui Ln. • Alamogordo, NM 88310 • Telephone (575) 437-0698

Using the SSC3 ActiveX Control

The SSC3 ActiveX control performs all of the serial port communications between your application and the SSC3 stepper motor control system.

Registering the SSC3 ActiveX Control

Before you can use the SSC3 control you must first register it on your computer. Copy the file named SSC3.ocx from the programming examples disk to the C:\Windows\System32 directory on your computer. Create a new subdirectory and copy the remaining project files for the Visual Basic or Visual C++ stepper motor exerciser programs onto your computer.

The control can be registered in Visual Basic 6.0 by choosing **Components** from the **Project** menu. Choose the **Browse** button and locate the SSC3.ocx file in the C:\Windows\System32 directory. Select the SSC3.ocx file and Click on **Open**. The SSC3 control is now added to the list of registered controls in the **Components** dialog box. Select the check box to the left of the control name and Click **OK** to close the **Components** dialog box. The SSC3 control icon will now appear in the toolbox. Finally, the control is added to the Visual Basic project form just like any of the other common controls like command buttons, text boxes etc.

The control can be registered in Visual C++ 6.0 by choosing **ActiveX Control Test Container** from the **Tools** menu. Select **Register Controls** from the **File** menu and then select **Register** to locate the SSC3.ocx file in the C:\Windows\System32 directory. Select the SSC3.ocx file and click on **Open**. The SSC3 control is now added to the list of registered controls in the **Register Controls** dialog box. To add the control to your Visual C++ project select **Add to Project\Components & Controls** from the **Project** menu. Double Click on **Registered ActiveX Controls** to display the **Registered ActiveX Controls Gallery**. Select **SSC3 Control** from the list and Click on **Insert**. Click **OK** on the **Confirm Classes** dialog when it appears and exit the **Registered ActiveX Controls Gallery** dialog box. The SSC3 control is now added to your project.

Using the controls with Visual Basic and Visual C++

Establishing the Serial Connection

Opening the Serial Port with the SetSerialPort Method

Visual Basic: SSC31.SetSerialPort PortSelected

Visual C++: m_SSC3.SetSerialPort(m_PortSelected);

PortSelected (value = 1 thru 12) This argument contains the serial port number that you will be using for communications with the SSC3 stepper motor control system. The valid range of serial port numbers is from 1 to 12.

The serial port automatically closes when your program ends?

Resetting the Interface with the ResetInterface Method

Visual Basic: SSC31.ResetInterface, ResetResponse

Visual C++: m_SSC3.ResetInterface(&ResetResponse);

ResetResponse (value = 5) This argument contains the returned response from the SSC3 interface. The interface will send a decimal 5 back to the host PC to indicate that a reset has been performed.

Testing Communications with the TestInterface Method

Visual Basic: SSC31.TestInterface, SyncByte, TestResponse

Visual C++: m_SSC3.TestInterface(m_SyncByte, &TestResponse);

SyncByte (value = 0 thru 255) This argument contains the sync byte that is used to test the communications integrity of the SSC3 interface. The value of the upper nibble in this byte is sent back to the host PC in the lower nibble of the response to this command.

TestResponse (value = upper nibble of sync byte sent) This argument contains the returned response from the SSC3 interface. The value in the lower nibble of the response that the interface sends back to the host PC is the value of the sync bytes upper nibble that was sent to the SSC3 interface.

Initializing the Interface with the Initialize Method

Visual Basic: SSC31.Initialize, S1HalfStep, S2HalfStep, S3HalfStep, S1Limits, S2Limits, S3Limits, S1Phases, S2Phases, S3Phases, S3DOption

Visual C++: m_SSC3.Initialize(m_S1HalfStep, m_S2HalfStep, m_S3HalfStep, m_S1Limits, m_S2Limits, m_S3Limits, m_S1Phases, m_S2Phases, m_S3Phases, m_S3DOption);

S1HalfStep, S2HalfStep, S3HalfStep (value = 0 or 1)

0 = full step mode. 1 = half step mode.

S1Limits, S2Limits, S3Limits (value = 0 or 1)

0 = Limit sense is not selected, 1 = Limit sense is selected.

S1Phases, S2Phases, S3Phases (value = 0 or 1)

0 = 1 phase mode selected. 1 = 2 phase mode selected.

S3DOption (value = 0 or 1)

0 = Stepper motor 3 control selected. 1 = Digital outputs 8 through 11 selected.

Setting the Step Rate Clock with the SetStepRate Method

This method sends command 4 to the SSC3 interface.

Visual Basic: SSC31.SetStepRate, StepRateSetting

Visual C++: m_SSC3.SetStepRate(m_StepRateSetting);

StepRateSetting (value = 1 thru 240) This argument contains the step rate clock setting which is common to all 3 stepper motors. The actual speed of a stepper motor is determined by the step rate clock value and the individual stepper motors speed divisor setting. The maximum step rate that can be achieved by the SSC3 interface is 1,748 steps-per-second (step rate = 240, speed divisor = 1). The slowest step rate that can be achieved is 1 step every 2.26 seconds (step rate clock = 1, speed divisor = 255).

Setting the Speed Divisor(s) with the SetDivisor Method

This method sends command 5 to the SSC3 interface. Depending upon the states of the stepper flags S1Flag, S2Flag and S3Flag, the speed divisor will be set simultaneously for any combination of the 3 stepper motors.

Visual Basic: `SSC31.SetDivisor S1Flag, S2Flag, S3Flag, DivisorSetting`

Visual C++: `m_SSC3.SetDivisor(m_S1Flag, m_S2Flag, m_S3Flag, m_DivisorSetting);`

DivisorSetting (value = 1 thru 255) This argument contains the stepper motor speed divisor setting for the SSC3 interface. The speed divisor is used to reduce the stepper speed by a division of 1 to 255.

Setting the Step Count(s) with the SetStepCount Method

This method sends command 6 to the SSC3 interface. Depending upon the states of the stepper flags S1Flag, S2Flag and S3Flag, the step count will be set simultaneously for any combination of the 3 stepper motors.

Visual Basic: `SSC31.SetStepCount S1Flag, S2Flag, S3Flag, StepCount`

Visual C++: `m_SSC3.SetStepCount(m_S1Flag, m_S2Flag, m_S3Flag, m_StepCount);`

StepCount (value = 1 to 65535) This argument contains the step count(s) for the SSC3 interface. If a stepper flag is set to 0 then the step count will not be loaded for that stepper. If a stepper flag is set to 1 then the step count will be loaded for that stepper.

Setting the Stepper Motor(s) Direction with the SetDirection Method

This method sends command 7 to the SSC3 interface. The direction of travel will be set simultaneously for all stepper motors.

Visual Basic: `SSC31.SetDirection S1Direction, S2Direction, S3Direction`

Visual C++: `m_SSC3.SetDirection(m_S1Direction, m_S2Direction, m_S3Direction);`

S1Direction, S2Direction, S3Direction (value = 0 or 1) These arguments contain the direction flag states for the stepper motors. If a direction flag = 0 then the direction will be set to clockwise. If a direction flag = 1 then the direction will be set to counter-clockwise.

Starting the Stepper Motor(s) into Motion with the Start Method

This method sends command 8 with bit 7 cleared to the SSC3 interface. Depending upon the states of the stepper flags S1Flag, S2Flag and S3Flag, any combination of the 3 stepper motors will be sent into motion simultaneously.

Visual Basic: `SSC31.Start S1Flag, S2Flag, S3Flag`

Visual C++: `m_SSC3.Start(m_S1Flag, m_S2Flag, m_S3Flag);`

The stepper motor(s) will be sent into motion executing the number of steps that has been loaded into its step count register. The motor(s) will stop automatically when the step count has finished execution. The motor(s) will stop before the step count has finished execution if a mechanical limit is detected or a stop command is issued. The number of steps remaining can be read with the `ReadStepsRemaining` method.

Starting the Stepper Motor(s) into Motion with the Run Method

This method sends command 8 with bit 7 set to the SSC3 interface. Depending upon the states of the stepper select flags S1Flag, S2Flag and S3Flag, any combination of the 3 stepper motors will be sent into motion simultaneously.

Visual Basic: `SSC31.Run S1Flag, S2Flag, S3Flag`

Visual C++: `m_SSC3.Run(m_S1Flag, m_S2Flag, m_S3Flag);`

The motor(s) will run continuously until a mechanical limit is detected or a stop command is issued.

Stopping the Stepper Motor(s) with the Stop Method

This method sends command 9 to the SSC3 interface. Depending upon the states of the stepper flags S1Flag, S2Flag and S3Flag, any combination of the 3 stepper motors will be stopped simultaneously.

Visual Basic: `SSC31.Stop S1Flag, S2Flag, S3Flag`

Visual C++: `m_SSC3.Stop(m_S1Flag, m_S2Flag, m_S3Flag);`

Reading a Steppers Status Word with the GetStatus Method

This method sends command 10 with bit 7 cleared to the SSC3 interface. The StepperNumber argument determines which stepper motors status is read. The PosCompFlag and AtLimitFlag arguments are the returned states of the Position Complete and At limit Flags.

Visual Basic: `SSC31.GetStatus StepperNumber, PosCompFLag, AtLimitFlag`

Visual C++: `m_SSC3.GetStatus(m_StepperNumber, &PosCompFlag, &AtLimitFlag);`

StepperNumber (value = 1, 2 or 3) This argument contains the stepper number.

PosCompFlag (value = 0 or 1) This argument is a returned response from the SSC3 interface. It contains the current state of the position complete flag for the stepper motor that was addressed by the value of the StepperNumber argument. If PosCompFlag = 0 then stepper positioning has not been completed. If PosCompFlag = 1 then stepper positioning has been completed.

AtLimitFlag (value = 0 or 1) This argument is a returned response from the SSC3 interface. It contains the current state of the at-limit flag for the stepper motor that was addressed by the StepperNumber argument. If AtLimitFlag = 0 then the stepper motor is not at a mechanical limit. If AtLimitFlag = 1 then the stepper motor is at a mechanical limit.

Reading the Number of Steps Remaining with the GetRemaining Method

This method sends command 10 with bit 7 set to the SSC3 interface. The StepperNumber argument determines which steppers remaining step count is read.

Visual Basic: `SSC31.GetRemaining StepperNumber, StepsRemaining`

Visual C++: `m_SSC3.GetRemaining(m_StepperNumber, &StepsRemaining);`

StepperNumber (value = 1, 2 or 3) This argument contains the stepper number.

StepsRemaining (value = 0 thru 65535) This argument contains the returned response from the SSC3 interface. It is equal to the number of steps that is remaining until positioning is completed after a step count is loaded and while a stepper motor is in motion from the execution of a start command. It is always equal to 1 while a motor is in motion from the execution of a Run command.

Setting the Stepper Motor(s) Ramp Rate with the SetRate Method

This method sends command 11 to the SSC3 interface. Depending upon the states of the stepper flags S1Flag, S2Flag and S3Flag, the ramp rate will be set simultaneously for any combination of the 3 stepper motors.

Visual Basic: `SSC31.SetRate S1Flag, S2Flag, S3Flag, RampRate`

Visual C++: `m_SSC3.SetRate(m_S1Flag, m_S2Flag, m_S3Flag, m_RampRate);`

RampRate (value = 0 thru 255) This argument contains the ramp rate setting for the SSC3 interface. The ramp value modifies the stepper(s) speed divisor and provides a means of ramping the acceleration and deceleration of the steppers speed.

Reading the State of a Digital Input with GetInput Method

This method sends command 12 to the SSC3 interface.

Visual Basic: `SSC31.GetInput InputNumber, InputState`

Visual C++: `m_SSC3.GetInput(m_InputNumber, &InputState);`

InputNumber (value = 1 thru 11) This argument contains the digital input number that is to be read.

InputState (value = 0 or 1) This argument is a returned response from the Get Input command that is sent back to the host PC by the SSC3 interface.

Changing the State of a Digital Output with the ChangeOutput Method

This method sends command 13 to the SSC3 interface.

Visual Basic: `SSC31.ChangeOutput OutputNumber, OutputState`

Visual C++: `m_SSC3.ChangeOutput(m_OutputNumber, m_OutputState);`

OutputNumber (value = 1 thru 11) This argument contains the digital output number whose output state is to be changed.

OutputState (value = 0 or 1) This argument contains the output state that the digital output is changed to.

Turning On and Off the Stepper(s) Auto Drive Option with the SetAuto Method

This method sends command 14 to the SSC3 interface. Depending upon the states of the stepper flags S1Auto, S2Auto, S3Auto, the auto drive option will be set simultaneously for any combination of the 3 stepper motors.

Visual Basic: `SSC31.SetAuto S1Auto, S2Auto, S3Auto`

Visual C++: `m_SSC3.SetAuto(m_S1Auto, m_S2Auto, m_S3Auto);`

S1Auto, S2Auto, S3Auto (value = 0 or 1) These arguments contain the auto drive option flag states for the stepper motors. If an auto flag = 0 then the auto drive option will be turned off for the stepper motor. If an auto flag = 1 then the auto drive option will be turned on for the stepper motor.