

# Eggert Electronics Engineering

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

## Using The SSC1 ActiveX Control

The SSC1 ActiveX control performs all of the serial port communications between your application and the SSC1 stepper motor control system.

## Registering the SSC1 ActiveX Control

Before you can use the SSC1 control you must first register it on your computer. Copy the file named SSC1.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 SSC1.ocx file in the C:\Windows\System32 directory. Select the SSC1.ocx file and Click on **Open**. The SSC1 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 SSC1 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 SSC1.ocx file in the C:\Windows\System32 directory. Select the SSC1.ocx file and click on **Open**. The SSC1 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 **SSC1 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 SSC1 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: `SSC11.SetSerialPort PortSelected`

Visual C++: `m_SSC1.SetSerialPort(m_PortSelected);`

**PortSelected (value = 1 thru 12)** This argument contains the serial port number that you will be using for communications with the SSC1 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 a Node with the ResetNode Method

Visual Basic: `SSC11.ResetNode Node, ResetResponse`

Visual C++: `m_SSC1.ResetNode(m_Node, &ResetResponse);`

**ResetResponse (value = node address)** This argument contains the returned response from the SSC1 interface that is addressed by the **Node** argument. The interface that is reset will send its node address back to the host PC to indicate that a reset has been performed.

### Testing Communications with a Node with the TestNode Method

Visual Basic: `SSC11.TestNode Node, TestResponse`

Visual C++: `m_SSC1.TestNode(m_Node, &TestResponse);`

**TestResponse (value = node address + 128)** This argument contains the returned response from the SSC1 interface that is addressed by the **Node** argument. The interface that is tested will send its node address (with bit 7 set) back to the host PC.

## Initializing a Node with the Initialize Method

Visual Basic: `SSC11.Initialize Node, StepMode, LimitLogic`

Visual C++: `m_SSC1.Initialize(m_Node, m_StepMode, m_LimitLogic);`

**StepMode (value = 0 thru 3)** This argument contains the stepper motor mode setting for the SSC1 interface that is addressed by the **Node** argument.

### Stepper Motor Mode Settings

<u>Mode</u>	<u>SSC1U Interface</u>	<u>SSC1B Interface</u>
0	Full step, 1 phase	Full step, 2 phase
1	Half step	Half step
2	Full step, 2 phase	Quarter step
3	Half step	Eighth step

**LimitLogic (value = 0 or 1)** This argument contains the mechanical limit logic setting for the SSC1 interface that is addressed by the **Node** argument. If **LimitLogic** equals 0 then the SSC1 interface will indicate an at-limit status when a limit input is high. If **LimitLogic** equals 1 then the interface will indicate an at-limit status when a limit input is low.

When an SSC1 interface is first powered up or it has been reset, the motor drive outputs will be disabled. The motor drive outputs are enabled after initialization has been performed. The SSC1U interface will ignore initialization commands if it is already in the initialized state. The SSC1B interface can be re-initialized when the stepper motor is stopped. It will ignore initialization commands if the stepper motor is running.

## Changing the Global Flag State of a Node with the ChangeGlobal Method

Visual Basic: `SSC11.ChangeGlobal Node, AllNodes, GlobalState`

Visual C++: `m_SSC1.ChangeGlobal(m_Node, m_AllNodes, m_GlobalState);`

Most of the stepper motor motion control commands can be sent simultaneously to any select number of nodes on the network. They are designated as global commands. If a nodes global flag is set then it will execute a global command without regard to the address contained in the **Node** argument. A node that is addressed by the **Node** argument will always execute the command. You can change the state of all node flags simultaneously if the **AllNodes** argument = 1.

**AllNodes (value = 0 or 1)** This argument contains a flag that determines whether the addressed node or all nodes have their global flags changed. If **AllNodes = 0** then the node that is address by the **Node** argument will have its global flag state changed. If **AllNodes = 1** then all nodes will have their global flags changed regardless of the address that is contained in the **Node** argument.

**GlobalState (value = 0 or 1)** This argument contains the global flag state for the SSC1 interface that is addressed by the **Node** argument. If **GlobalState = 0** and **AllNodes = 0** then the global flag for the SSC1 interface addressed by the **Node** argument will be cleared. If **GlobalState = 1** and **AllNodes = 0** then the global flag for the interface addressed will be set. If **AllNodes = 1** then the **Node** argument has no significance and all nodes will have their global flags changed to the state determined by the **GlobalState** argument.

### Setting the Speed of a Stepper Motor with the SetSpeed Method

Visual Basic: `SSC11.SetSpeed Node, SpeedSetting`

Visual C++: `m_SSC1.SetSpeed(m_Node, m_SpeedSetting);`

**SpeedSetting (value = 1 thru 255)** This argument contains the stepper motor speed setting for the SSC1 interface that is addressed by the **Node** argument. The actual speed in steps-per-second will be 10 times faster than this value. If the speed divisor value is set to 1 and the speed value is set to 1 then the actual step rate will be 10 steps-per-second. If the speed divisor value is set to 10 and the speed value is set to 1 then the actual step rate will be 1. The maximum step rate for the SSC1 interface is 2,550 steps-per-second with a speed divisor setting of 1. The slowest step rate for the SSC1 interface is 1 step every 102.4 seconds (10 steps-per-second / 1024).

This is an SSC1 system global command. If any node on a network has its global flag set it will set its speed with the value contained in the **SpeedSetting** argument.

### Setting the Speed Divisor of a Stepper Motor with the SetDivisor Method

Visual Basic: `SSC11.SetDivisor Node, DivisorSetting`

Visual C++: `m_SSC1.SetDivisor(m_Node, m_DivisorSetting);`

**DivisorSetting (value = 1 thru 1023)** This argument contains the stepper motor speed divisor setting for the SSC1 interface that is addressed by the **Node** argument. The speed divisor is used to reduce the stepper speed by a division of 1 to 1024.

This is an SSC1 system global command. If any node on a network has its global flag set it will set its speed divisor with the value contained in the **DivisorSetting** argument.

### **Setting a Step Count for a Stepper Motor with the SetStepCount Method**

Visual Basic: SSC11.SetStepCount Node, StepCount

Visual C++: m\_SSC1.SetStepCount(m\_Node, m\_StepCount);

**StepCount (value = 1 to 262143)** This argument contains the step count for the SSC1 interface that is addressed by the **Node** argument. The step count is used when a Start or Auto Start command is executed.

This is an SSC1 system global command. If any node on a network has its global flag set it will set its step count with the value contained in the **StepCount** argument.

### **Setting the Direction of a Stepper Motor with the SetDirection Method**

Visual Basic: SSC11.SetDirection Node, Direction

Visual C++: m\_SSC1.SetDirection(m\_Node, m\_Direction);

**Direction (value = 0 or 1)** This argument contains the direction flag state for a stepper motor that is addressed by the **Node** argument. If **Direction** = 0 then the direction will be set to clockwise. If **Direction** = 1 then the direction will be set to counter-clockwise.

This is an SSC1 system global command. If any node on a network has its global flag set it will set its direction to the value contained in the **Direction** argument.

### **Starting a Stepper Motor into Motion with the Run Method**

Visual Basic: SSC11.Run Node

Visual C++: m\_SSC1.Run(m\_Node);

This command starts a stepper motor that is addressed by the **Node** argument. The motor will run continuously until a mechanical limit is detected, a sense-activated input is detected, or a stop command is issued.

This command will be ignored if it is received by the SSC1 interface while the stepper motor is in motion.

This is an SSC1 system global command. If any node on a network has its global flag set it will start motion in run mode.

### **Starting a Stepper Motor into Motion with the AutoRun Method**

Visual Basic: `SSC11.AutoRun Node`

Visual C++: `m_SSC1.AutoRun(m_Node);`

This command starts a stepper motor that is addressed by the **Node** argument. The motor will run continuously until a mechanical limit is detected, a sense-activated input is detected, or a stop command is issued.

The **AutoRun** command is identical to the **Run** command with the exception that it can be sent to the SSC1 interface while the stepper motor is in motion. This command will be executed if the stepper is stopped when the command is issued, or immediately after the stepper stops motion from the previous motion command.

This is an SSC1 system global command. If any node on a network has its global flag set it will start motion in run mode.

### **Starting a Stepper Motor into Motion with the Start Method**

Visual Basic: `SSC11.Start Node`

Visual C++: `m_SSC1.Start(m_Node);`

This command starts the stepper motor that is addressed by the **Node** argument into motion executing the number of steps set by the **SetStepCount** command. The position complete flag in stepper status word 1 will be set if the number of designated steps has been completed successfully. The position complete flag will not be set if step execution is interrupted by the detection of a mechanical limit, a sense-activated input, or a stop command is issued.

This command will be ignored if the command is received while the stepper motor is in motion.

This is an SSC1 system global command. If any node on a network has its global flag set it will start motion executing the number of steps in its step count register.

## Starting a Stepper Motor into Motion with the AutoStart Method

Visual Basic: SSC11.AutoStart Node

Visual C++: m\_SSC1.AutoStart(m\_Node);

This command starts the stepper motor that is addressed by the **Node** argument into motion executing the number of steps set by the **SetStepCount** command. The position complete flag in stepper status word 1 will be set if the number of designated steps has been completed successfully. The position complete flag will not be set if step execution is interrupted by the detection of a mechanical limit, a sense-activated input, or a stop command is issued.

The **AutoStart** command is identical to the **Start** command with the exception that it can be sent to the SSC1 interface while the stepper motor is in motion. This command will be executed if the stepper is stopped when the command is issued, or immediately after the stepper stops motion from the previous motion command.

This is an SSC1 system global command. If any node on a network has its global flag set it will start motion executing the number of steps in its step count register.

## Stopping a Stepper Motor with the Stop Method

Visual Basic: SSC11.Stop Node

Visual C++: m\_SSC1.Stop(m\_Node);

This command immediately stops the stepper motor that is addressed by the **Node** argument.

This is an SSC1 system global command. If any node on a network has its global flag set it will stop its stepper motor immediately.

## Reading Stepper Status Word 1 with the GetStatus1 Method

Visual Basic: SSC11.GetStatus1 Node, StatusWord1

Visual C++: m\_SSC1.GetStatus1(m\_Node, &StatusWord1);

**StatusWord1 (value = 0 thru 255)** This argument contains the returned response from the SSC1 interface that is addressed by the **Node** argument.

Each bit in **StatusWord1** contains a status flag state.

## StatusWord1 Flag Bit Functions

<u>Bit Number</u>	<u>Flag Function</u>
0	Position complete flag
1	Motor run flag
2	Counter-clockwise limit flag
3	Clockwise limit flag
4	Sense flag
5	Serial port overrun error flag
6	Serial port framing error flag
7	Global flag.

**Note:** The SSC1 system technical manuals contain a detailed functional description of the status flags.

## Reading Stepper Status Word 2 with the GetStatus2 Method

Visual Basic: `SSC11.GetStatus2 Node, StatusWord2`

Visual C++: `m_SSC1.GetStatus2(m_Node, &StatusWord2);`

**StatusWord2 (value = 0 thru 255)** This argument contains the returned response from the SSC1 interface that is addressed by the **Node** argument.

Each bit in **StatusWord2** contains a status flag state.

## StatusWord2 Flag Bit Functions

<u>Bit Number</u>	<u>Flag Function</u>
0	Parameters load flag
1	Direction flag
2	Initialization flag
3	Continuous run flag
4	Auto drive enable flag
5	Power down flag (not used with SSC1U)
6	Acceleration on flag
7	Deceleration on flag

### Reading the Number of Steps Remaining with the GetRemaining Method

Visual Basic: SSC11.GetRemaining Node, StepsRemaining

Visual C++: m\_SSC1.GetRemaining(m\_Node, &StepsRemaining);

**StepsRemaining (value = 0 thru 262143)** This argument contains the returned response from the SSC1 interface that is addressed by the **Node** argument. It is equal to the number of steps that is remaining until positioning is completed during a **Start** or **AutoStart** command execution. It is always equal to 1 while a motor is in motion in Run mode.

### Reading the Number of Steps Taken with the GetTaken Method

Visual Basic: SSC11.GetTaken Node, StepsTaken

Visual C++: m\_SSC1.GetTaken(m\_Node, &StepsTaken);

**StepsTaken (value = 0 thru 262143)** This argument contains the returned response from the SSC1 interface that is addressed by the **Node** argument. It is equal to the number of steps that has been taken since motion has been started by a **Run**, **AutoRun**, **Start** or **AutoStart** command.

### Setting the Ramp Rate for a Stepper Motor with the SetRate Method

Visual Basic: SSC11.SetRate Node, RampRate

Visual C++: m\_SSC1.SetRate(m\_Node, m\_RampRate);

**RampRate (value = 1 thru 255)** This argument contains the ramp rate setting for the SSC1 interface that is addressed by the **Node** argument. A ramp rate setting of 1 produces the longest ramp. The ramp rate setting limits the stepper motors speed for the first 255 steps taken during acceleration, and the last 255 steps remaining during deceleration. The longest possible ramp, which is 255 steps, is obtainable by setting the stepper speed to 255 and the ramp rate to 1. The number of steps in the ramp is determined by dividing the stepper speed value by the ramp rate value. See the system technical manual for a detailed description of this command.

This is an SSC1 system global command. If any node on a network has its global flag set it will set its ramp rate to the value contained in the **RampRate** argument.

## Setting the Ramp Accel/Decel Flags with the SetAccDec Method

Visual Basic: `SSC11.SetAccDec Node, RampFlags`

Visual C++: `m_SSC1.SetAccDec(m_Node, m_RampFlags);`

**RampFlags (value = 0 thru 3)** This argument contains the ramp acceleration and deceleration select flag settings for the SSC1 interface that is addressed by the **Node** argument. Bit 0 in this argument contains the acceleration select flag state and bit 1 contains the deceleration select flag state. You can use separate variables for each flag state and combine them into the **RampFlags** argument, or simply set the **RampFlags** argument to a value of 0, 1, 2 or 3 representing one of the 4 possible flag settings.

### Stepper Motor Ramp Flag Settings

0 = No Ramping

1 = Acceleration ramping only

2 = Deceleration ramping only

3 = Acceleration and deceleration ramping

### Combining Separate Ramp Flag Variables

**RampFlags** = (DecelFlag \* 2) + AccelFlag

This is an SSC1 system global command. If any node on a network has its global flag set it will set its ramp flags to the value contained in the **RampFlags** argument.

## Setting the Ramp Offset with the SetOffset Method

Visual Basic: `SSC11.SetOffset Node, RampOffset`

Visual C++: `m_SSC1.SetOffset(m_Node, m_RampOffset);`

**RampOffset (value = 0 thru 255)** This argument contains the ramp offset setting for the SSC1 interface that is addressed by the **Node** argument. The SSC1 interface adds the ramp offset value to the ramp rate value to set a minimum starting speed for ramping.

This is an SSC1 system global command. If any node on a network has its global flag set it will set its ramp offset to the value contained in the **RampOffset** argument.

## Setting the Ramp Stretch Flags with the SetStretch Method

Visual Basic: SSC11.SetStretch Node, StretchFlags

Visual C++: m\_SSC1.SetStretch(m\_Node, m\_StretchFlags);

**StretchFlags (value = 0 thru 2)** This argument contains the ramp stretch flag settings for the SSC1 interface that is addressed by the **Node** argument. Bit 0 in this argument contains the ramp times 2 select flag state and bit 1 contains the ramp times 4 select flag state.

### Ramp Stretch Flag Settings

0 = Normal ramp (maximum of 255 steps)

1 = 2 times normal ramp (maximum of 511 steps)

2 = 4 times normal ramp (maximum of 1023 steps)

### Combining Separate Ramp Stretch Flag Variables

**StretchFlags** = (StretchX4Flag \*2) + StretchX2Flag

## Turning On and Off the Auto Drive Enable Option with the SetEnable Method

Visual Basic: SSC11.SetEnable Node, EnableFlag

Visual C++: m\_SSC1.SetEnable(m\_Node, m\_EnableFlag);

**EnableFlag (Value = 0 or 1)** This argument contains the auto drive enable option flag state for a stepper motor that is addressed by the **Node** argument. If **EnableFlag** = 0 then the auto drive enable option will be disabled. If **EnableFlag** = 1 then the auto drive enable option will be enabled.

This is an SSC1 system global command. If any node on a network has its global flag set it will set its auto drive enable option to the value contained in the **EnableFlag** argument.

## Turning On and Off the Auto Power Down Option with the SetPower Method

Visual Basic: `SSC11.SetPower Node, PowerFlag`

Visual C++: `m_SSC1.SetPower(m_Node, m_PowerFlag);`

**PowerFlag (Value = 0 or 1)** This argument contains the auto power down option flag state for the stepper motor that is addressed by the **Node** argument. If **PowerFlag = 0** then the auto power down option will be disabled. If **PowerFlag = 1** then the auto power down option will be enabled.

This is an SSC1 system global command. If any node on a network has its global flag set it will set its auto power down option to the value contained in the **PowerFlag** argument.

## Reading the State of a Digital Input with GetInput Method

Visual Basic: `SSC11.GetInput InputNumber, NowValue, LatchValue`

Visual C++: `m_SSC1.GetInput(m_InputNumber, &NowValue, &LatchValue);`

**InputNumber (value = 1 thru 64)** This argument contains the digital input number that is to be read. There can be up to 64 digital inputs in an SSC1 system network. Digital inputs 1 thru 4 are located on the SSC1 interface with the address of 1. Digital inputs 5 thru 8 are located on the SSC1 interface with the address of 2 etc.

**NowValue (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 SSC1 interface. **NowValue** is the logic state of the digital input when the Get Input command is interpreted by the SSC1 interface.

**LatchValue (value = 0 or 1)** This argument is also a returned response from the Get Input command that is sent back to the host PC by the SSC1 interface. The SSC1 interface reads the states of the digital inputs a minimum of 10 times per second. If **LatchValue = 1** then the digital input was read as a 1 at least once since the last Get Input command was interpreted. The SSC1 interface resets the latched value to 0 after the Get Input command sends the response to the host PC.

## Changing the State of a Digital Output with the ChangeOutput Method

Visual Basic: `SSC11.ChangeOutput OutputNumber, State`

Visual C++: `m_SSC1.ChangeOutput( m_OutputNumber, m_State);`

**OutputNumber (value = 1 thru 64)** This argument contains the digital output number whose output state is to be changed. There can be up to 64 digital outputs in an SSC1 system network. Digital outputs 1 thru 4 are located on the SSC1 interface with the address of 1. Digital outputs 5 thru 8 are located on the SSC1 interface with the address of 2 etc.

**State (value = 0 or 1)** This argument contains the output state that the digital output addressed by the **OutputNumber** argument is changed to.

## Inverting the Logic State of a Digital Input with the ChangeLogic Method

Visual Basic: `SSC11.ChangeLogic InputNumber, LogicState`

Visual C++: `m_SSC1.ChangeLogic(m_InputNumber, m_LogicState);`

**InputNumber (value = 1 thru 64)** This argument contains the digital input number that will have its logic state changed.

**LogicState (value = 0 or 1)** This argument contains the logic state that the digital input addressed by the **InputNumber** argument will be changed to. If **LogicState** = 0 then the digital inputs logic will be normal with an input state that is read by the interface as a 1 being reported as a 1. If **LogicState** = 1 then the digital inputs logic will be reversed with an input state that is read by the interface as a 1 being reported as a 0.

## Turning On and Off Input Sense for a Digital Input with the ChangeSense Method

Visual Basic: `SSC11.ChangeSense InputNumber, SenseState`

Visual C++: `m_SSC1.ChangeSense(m_InputNumber, m_SenseState);`

**InputNumber (value = 1 thru 64)** This argument contains the digital input number that will have its input sense turned on or off.

**SenseState (value = 0 or 1)** This argument contains the input sense state that the digital input addressed by the **InputNumber** argument will be changed to. If **SenseState** = 0 then input sense will be turned off. If **SenseState** = 1 then input sense will be turned on.