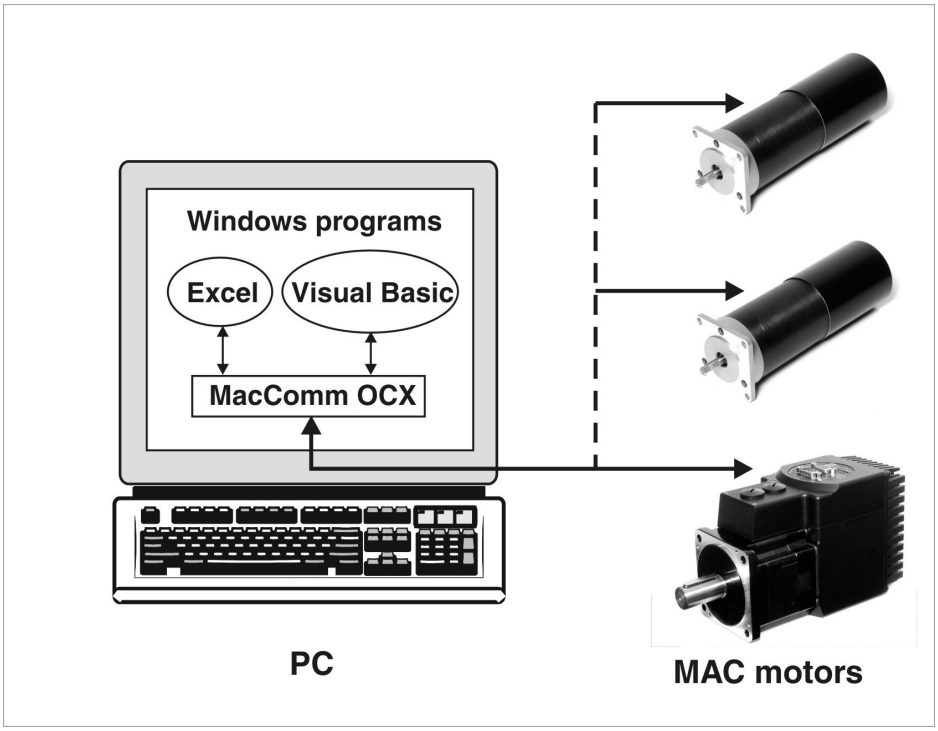
MacComm OCX Control 1.01

User Manual



JVL Industri Elektronik A/S

Contents:

Foreword: 2
Interface overview: 3
Methods: 3
Properties: 3
Method Descriptions: 4
OpenPort 4
ClosePort 4
ReadParameter 5
ReadParameterAlternate 6
WriteParameter 7
WriteParameterAlternate 8
GetParamNumFromName 9
GetLastError 10
GetLastErrorStr 10
GetParameterType 11
Reset 12
ResetWait 12
WriteToFlash 13
WriteToFlashWait 13
SetFactors 14
SetFactorsBig 15
SetMACType 16
GetModuleType 16
NanoGet 17
NanoPut 17
NanoProgram 18
NanoWrite 18
NanoStop 18
NanoReset 19
NanoResetWait 19
NanoDebug 20
Installation 21
Adding MacComm OCX to the program 22
Visual Basic 6 22
Visual C++ 6 22
Visual .NET 22
Borland C++ Builder 6.0 22
LabVIEW 7.0 22
Custom Errors: 23

Foreword:

MacComm OCX has been developed to provide an easy way of interfacing to the MAC motors from various Windows applications.

To use the MacComm OCX you only need a development environment that supports ActiveX components (previously called OLE controls)

This includes: MS Excel, MS Visual Studio, Borland C++ Builder, Borland Delphi, LabView, and many more.

By using this OCX you do not need to worry about setting up the COM port settings such as: baud rate, start bits, stop bits, databits, parity, CTS, RTS etc.
These settings are handled automatically by the OCX. All you need is to tell what COM port is used and what addresses the MAC motors are connected.

The MAC motor address is only required when more than one MAC motor is connected to the same serial cable. Otherwise address 255 is a broadcast address, where the MAC motor will react regardless of configured address.

First you need to add the OCX to your project (See Installation instructions later in this manual)
To initialize the OCX the following are required:

- Setting “ComPort” to wanted COM port number
- Calling SetMACType to set MAC type (only required for MAC 400/800)
- Calling “OpenPort”

Now it is possible to use the various commands for setting/retrieving values:

- Call ReadParameterAlternate to read a value
- Call WriteParameterAlternate to set a value
- Etc.

To close the communications do the following:

- Call ClosePort

Interface overview:

Methods:

OpenPort()
ClosePort()
ReadParameter([in] Address, [in] ParamNum, [out] Value)
ReadParameterAlternate([in] Address, [in] ParamNum, [out] Value)
WriteParameter([in] Address, [in] ParamNum, [in] Value)
WriteParameterAlternate([in] Address, [in] ParamNum, [in] Value)
GetParamNumFromName([in] Address, [in] ParamNum)
GetLastError()
GetLastErrorStr([in] long ErrorCode)
GetParameterType([in] Address, [in] ParamNum)
Reset([in] Address)
ResetWait([in] Address)
WriteToFlash([in] Address)
WriteToFlashWait([in] Address)
SetFactors([in] PositionFactor, [in] AccelerationFactor, [in] VelocityFactor)
SetFactorsBig([in] PositionFactor, [in] AccelerationFactor, [in] VelocityFactor)
SetMACType([in] Address, [in] Type)
GetModuleType([in] Address, [out] ModuleType, [out] FirmWareVersion)
NanoGet([in] Address, [in] Page, [out] Data)
NanoPut([in] Address, [in] Page, [in] Data)
NanoProgram([in] Address, [in] FileName)
NanoWrite([in] Address)
NanoStop([in] Address)
NanoReset([in] Address)
NanoResetWait([in] Address)
NanoDebug([in] Address, [out] State, [out] Inputs, [out] Outputs, [out] Time, [out] Count, [out] ModeCmd)
AboutBox()

Properties:

ComPort
Retries

Method Descriptions:

In the examples MacComm is an instance of the MacComm OCX.
NOTE: All methods will block the calling thread until completed.

OpenPort()
Return type: Boolean Returns true if open was successful
Description: Use this method to open the port
Example(s): **C++:** Opening the port bool Result=MacComm.OpenPort(); **BASIC:** Opening the port Dim Result As Boolean Result = MacComm.OpenPort

ClosePort()
Description: Use this method to close the port
Example(s): **C++:** Closing the port MacComm.ClosePort(); **BASIC:** Closing the port MacComm.ClosePort

ReadParameter([in] Address, [in] ParamNum, [out] Value)

Parameters:

Type	Name	Description
16 bit signed integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit signed integer	ParamNum	Parameter number
32 bit signed integer (pointer)	Value	Value to be read (Pointer)

Return type: Boolean

Returns true if read was successful

It will try the amount of times the property "Retries" has been set to before returning false.

Description:

Use this method to read a parameter from a Macmotor register

Value is one of the following types cast to a long integer:

- Word: 16 bit unsigned integer
- Integer: 16 bit signed integer
- LongInt: 32 bit signed integer
- Fixed4: 16 bit signed fixed point (Unit: 1/4096)
- Fixed8: 16 bit signed fixed point (Unit: 1/256)
- Fixed16 32 bit signed fixed point (Unit: 1/65536)
- Fixed24 32 bit signed fixed point (Unit: 1/256)

Example(s):

C++:

Getting operation mode (Parameter number 2)

```
long Value;
bool Result=MacComm.ReadParameter(255,2,&Value);
```

Getting position (Parameter 10: P_IST)

```
long Value;
bool Result=MacComm.ReadParameter(255,3,&Value);
```

BASIC:

Common dim statements:

```
Dim LocalValue As Long
Dim Result As Boolean
```

Getting operation mode (Parameter number 2)

```
Result = MacComm.ReadParameter(255, 2, LocalValue)
```

Getting position (Parameter 10: P_IST)

```
Result = MacComm.ReadParameter(255, 10, LocalValue)
```

ReadParameterAlternate([in] Address, [in] ParamNum, [out] Value)

Parameters:

Type	Name	Description
16 bit signed integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit signed integer	ParamNum	Parameter number
32 bit floating point (pointer)	Value	Value to be read (Pointer)

Return type: Boolean
Returns true if read was successful
It will try the amount of times the property "Retries" has been set to before returning false.

Description:
Use this method to read a parameter from a Macmotor register
This method uses the factors for Acceleration, Position and Velocity-registers, which can be set by calling SetFactors.
Some of the other registers are converted using predefined factors (See SetFactors)
The rest just pass through
Types are handled automatically by this method

Example(s):

C++:

Getting operation mode (Parameter number 2)

```
float Value;
bool Result=MacComm.ReadParameterAlternate(255,2,&Value);
```

Getting position (Parameter 10: P_IST) multiplied with Positionfactor

```
float Value;
bool Result=MacComm.ReadParameterAlternate(255,3,&Value);
```

BASIC:

Common dim statements:

```
Dim LocalValue As Single
Dim Result As Boolean
```

Getting operation mode (Parameter number 2)

```
Result = MacComm.ReadParameterAlternate(255, 2, LocalValue)
```

Getting position (Parameter 10: P_IST) multiplied with Positionfactor

```
Result = MacComm.ReadParameterAlternate(255, 10, LocalValue)
```

WriteParameter([in] Address, [in] ParamNum, [in] Value)

Parameters:

Type	Name	Description
16 bit signed integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit signed integer	ParamNum	Parameter number
32 bit signed integer	Value	Value to be written

Return type: Boolean

Returns true if write was successful

It will try the amount of times the property "Retries" has been set to before returning false.

Description:

Use this method to write a parameter to a Macmotor register

Value is one of the following types cast to a long integer:

- Word: 16 bit unsigned integer
- Integer: 16 bit signed integer
- LongInt: 32 bit signed integer
- Fixed4: 16 bit signed fixed point (Unit: 1/4096)
- Fixed8: 16 bit signed fixed point (Unit: 1/256)
- Fixed16 32 bit signed fixed point (Unit: 1/65536)
- Fixed24 32 bit signed fixed point (Unit: 1/256)

Example(s):

C++:

Setting operation mode (Parameter number 2) to Position mode (Value 2)

```
bool Result=MacComm.WriteParameter(255,2,2);
```

Setting Position (Parameter 3: P_SOLL) to 4096 (Value 4096)

```
bool Result=MacComm.WriteParameter(255,3,4096);
```

BASIC:

Setting operation mode (Parameter number 2) to Position mode (Value 2)

```
Dim Result As Boolean
Result = MacComm1.WriteParameter(255, 2, 2)
```

Setting Position (Parameter 3: P_SOLL) to 4096 (Value 4096)

```
Dim Result As Boolean
Result = MacComm1.WriteParameter(255, 3, 4096)
```

WriteParameterAlternate([in] Address, [in] ParamNum, [in] Value)

Parameters:

Type	Name	Description
16 bit signed integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit signed integer	ParamNum	Parameter number
32 bit floating point	Value	Value to be written

Return type: Boolean
Returns true if write was successful
It will try the amount of times the property "Retries" has been set to before returning false.

Description:
Use this method to write a parameter to a Macmotor register
This method uses the factors for Acceleration, Position and Velocity-registers, which can be set by calling SetFactors.
Some of the other registers are converted using predefined factors (See SetFactors)
The rest just pass through
Types are handled automatically by this method

Example(s):

C++:

Setting operation mode (Parameter number 2) to Position mode (Value 2)

```
bool Result=MacComm.WriteParameterAlternate(255,2,2);
```

Setting Position (Parameter 3: P_SOLL) to 4000 divided by Positionfactor (Value 4000)

```
bool Result=MacComm.WriteParameterAlternate(255,3,4000);
```

BASIC:

Setting operation mode (Parameter number 2) to Position mode (Value 2)

```
Dim Result As Boolean
Result = MacComm1.WriteParameterAlternate(255, 2, 2)
```

Setting Position (Parameter 3: P_SOLL) to 4000 divided by Positionfactor (Value 4000)

```
Dim Result As Boolean
Result = MacComm1.WriteParameterAlternate(255, 3, 4000)
```

GetParamNumFromName([in] Address, [in] ParamName)

Parameters:

Type	Name	Description
16 bit signed integer	Address	Address of the MAC motor
String	ParamName	Parameter name

Return type: 16 bit signed integer
Returns parameter number or 0 if not found.

Description:
Use this method to retrieve the parameter number from the name

Example(s):

C++:

Getting last error code:

```
unsigned short ParamNum=MacComm.GetParamNumFromName(255, "P_IST");
```

BASIC:

Getting last error code:

```
Dim ParamNum As Integer
ParamNum=MacComm.GetParamNumFromName(255, "P_IST")
```

AboutBox()

Description:

Shows a dialog about the program

Example(s):

C++:

Show the about box

```
MacComm.AboutBox();
```

BASIC:

Show the about box

```
MacComm.AboutBox
```

GetLastError()

Return type: 32 bit signed integer
Returns an error code like the Windows GetLastError(), but with some additions

Description:
Use this method to retrieve the error code for the last error

Example(s):
C++:
Getting last error code:

```
unsigned short ErrorCode=MacComm.GetLastError();
```

BASIC:
Getting last error code:

```
Dim ErrorCode As Integer
ErrorCode = MacComm.GetLastError
```

GetLastErrorStr([in] ErrorCode)

Parameters:

Type	Name	Description
32 bit signed integer	ErrorCode	Errorcode to be converted to a string

Return type: String
Returns an error code description like the Windows GetLastError(), but with the same additions as GetLastError()

Description:
Use this method to retrieve a description of an error code

Example(s):
C++:
Get description of passed error code

```
CString Text=MacComm.GetLastErrorStr(MacComm.GetLastError());
```

BASIC:
Getting last error code:

```
Dim Description As String
Description = MacComm.GetLastErrorStr(MacComm.GetLastError)
```

GetParameterType([in] Address, [in] ParamNum)

Parameters:

Type	Name	Description
16 bit signed integer	Address	Address of the MAC motor
16 bit signed integer	ParamNum	Parameter number

Return type: 16 bit signed integer

Return value indicates what type the parameter is stored as internally in the MAC Motor

-1	Invalid	Invalid parameter!
0	Word:	16 bit unsigned integer
1	Integer:	16 bit signed integer
2	LongInt:	32 bit signed integer
3	Fixed4:	16 bit signed fixed point (Unit: 1/4096)
4	Fixed8:	16 bit signed fixed point (Unit: 1/256)
5	Fixed16	32 bit signed fixed point (Unit: 1/65536)
6	Fixed24	32 bit signed fixed point (Unit: 1/256)

Description:

Use this method to determine how a parameter should be sent.
The integer types should just be used as parameters.
The Fixed4 type should be converted to an integer by multiplying with 4096
The Fixed8 type should be converted to an integer by multiplying with 256
The Fixed16 type should be converted to an integer by multiplying with 65536
The Fixed24 type should be converted to an integer by multiplying with 256

Example(s):

C++:

Get Parameter 100s type

```
short Type=MacComm.GetParameterType(100);
```

BASIC:

Get Parameter 100s type

```
Dim ParameterType As Integer
ParameterType = MacComm.GetParameterType(100)
```

Reset([in] Address)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)

Return type: Boolean
Returns true if reset was successful
It will try the amount of times the property "Retries" has been set to before returning false.

Description:
Resets MAC motor to last flashed values.
Returns as soon as the Reset command has been sent to the MAC motor.

Example(s):
C++:

```
Reset MAC motor
    bool Result=MacComm.Reset(255);

BASIC:
Reset MAC motor
    Dim Result As Boolean
    Result=MacComm.Reset(255)
```

ResetWait([in] Address)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)

Return type: Boolean
Returns true if reset was successful
It will try the amount of times the property "Retries" has been set to before returning false.

Description:
Resets MAC motor to last flashed values
Returns when MAC motor is ready.

Example(s):
C++:

```
Reset MAC motor
    bool Result=MacComm.Reset(255);

BASIC:
Reset MAC motor
    Dim Result As Boolean
    Result=MacComm.Reset(255)
```

WriteToFlash([in] Address)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)

Return type: Boolean

Returns true if flashing was successful
It will try the amount of times the property "Retries" has been set to before returning false.

Description:

Writes MAC registers to Flash memory
Returns as soon as the Flash command has been sent to the MAC motor.

Example(s):

C++:

Write registers to flash

```
bool Result=MacComm.WriteToFlash(255);
```

BASIC:

Write registers to flash

```
Dim Result As Boolean
Result=MacComm.WriteToFlash(255)
```

WriteToFlashWait([in] Address)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)

Return type: Boolean

Returns true if flashing was successful
It will try the amount of times the property "Retries" has been set to before returning false.

Description:

Writes MAC registers to Flash memory
Returns when MAC motor is ready.

Example(s):

C++:

Write registers to flash

```
bool Result=MacComm.WriteToFlashWait(255);
```

BASIC:

Write registers to flash

```
Dim Result As Boolean
Result=MacComm.WriteToFlashWait (255)
```

SetFactors([in] PositionFactor, [in] AccelerationFactor, [in] VelocityFactor)

Parameters:

Type	Name	Description
16 bit floating point	Pos	Position Factor
16 bit floating point	Acc	Acceleration Factor
16 bit floating point	Vel	Velocity Factor

Description:

Sets factors used by ReadParameterAlternate and WriteParameterAlternate
If a value of 0 is passed the previous factor is retained
The defaults are

Name	Factor	Resulting unit
PositionFactor	1	Pulses
AccelerationFactor	~248.3	RPM/s
VelocityFactor	~0.4768	RPM

The following registers are also converted, but these factors are fixed:

7 (T_SOLL)	100/1023	Percent
8 (P_SIM)	1/16	Encoder counts
16 (I2T)	1/22	Percent (assuming I2TLIM is 2200)
18 (UIT)	1/6	Percent (assuming UITLIM is 600)
41 (T_HOME)	100/1023	Percent
77-80 (T1-4)	100/1023	Percent
121 (VF_OUT)	100/1023	Percent
122 (ANINP)	10/1023	Volts
123 (ANINP_OFFSET)	10/1023	Volts
124 (ELDEGN_OFFSET)	360/2048	Degrees
125 (ELDEGP_OFFSET)	360/2048	Degrees
143 (ELDEG_IST)	360/2048	Degrees
151 (U_SUPPLY)	0.0537	Volts

Example(s):

C++:

Set Position factor to 1/4096 (Converts Pulses to revolutions), and disable the other factors

```
MacComm.SetFactors((float)1/4096,1,1);
```

BASIC:

Set Position factor to 1/4096 (Converts Pulses to revolutions), and disable the other factors

```
MacComm.SetFactors 1/4096,1,1
```

SetFactorsBig([in] PositionFactor, [in] AccelerationFactor, [in] VelocityFactor)

Parameters:

Type	Name	Description
16 bit floating point	Pos	Position Factor
16 bit floating point	Acc	Acceleration Factor
16 bit floating point	Vel	Velocity Factor

Description:

Sets factors used by ReadParameterAlternate and WriteParameterAlternate
If a value of 0 is passed the previous factor is retained
The defaults are

Name	Factor	Resulting unit
PositionFactor	1	Pulses
AccelerationFactor	~277.922	RPM/s
VelocityFactor	~0.360938	RPM

The following registers are also converted, but these factors are fixed:

7 (T_SOLL)	100/1023	Percent
8 (P_SIM)	1/16	Encoder counts
21 (U_24V)	~74.47	Volts
29 (DEGC)	0.05/4096	Degrees
31 (DEGCMAX)	0.05/4096	Degrees
41 (T_HOME)	100/1023	Percent
46 (T_REG_P)	100/1023	Percent
77 (TQ0)	100/1023	Percent
78 (TQ1)	100/1023	Percent
79 (TQ2)	100/1023	Percent
80 (TQ3)	100/1023	Percent
169 (VF_OUT)	100/1023	Percent
170 (ANINP)	10/2047	Percent
171 (ANINP_OFFSET)	10/2047	Percent
172 (ELDEG_OFFSET)	360/1143	Electrical degrees
176 (MAN_ALPHA)	360/1143	Electrical degrees
190 (ELDEG_IST)	360/1143	Electrical degrees
191 (V_ELDEG)	360/1143	Electrical degrees
198 (U_BUS)	325.3/400	Volts
199 (U_BUS_OFFSET)	325.3/400	Volts

Example(s):

C++:

Set Position factor to 1/8000 (Converts Pulses to revolutions), and disable the other factors

```
MacComm.SetFactorsBig ((float)1/8000,1,1);
```

BASIC:

Set Position factor to 1/8000 (Converts Pulses to revolutions), and disable the other factors

```
MacComm.SetFactorsBig 1/8000,1,1
```

SetMACType([in] Address, [in] Type)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
BOOL	Type	true if it is a MAC 400 or MAC 800

Description:

Sets MAC motor type.
Default is 0
Type can have the following values:

0 MAC50/95/140/141
1 MAC400/800

Example(s):

C++:

Set MAC400 on address 4:

```
MacComm.SetMACType (4,TRUE);
```

BASIC:

Set MAC80 on address 4:

```
MacComm.SetMACType 4,FALSE
```

GetModuleType([in] Address, [out] ModuleType, [out] FirmWareVersion)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit unsigned short integer	ModuleType	What type of module is it
16 bit unsigned short integer	FWVersion	FirmWare version

Description:

Reads module information from the MAC motor
The values of ModuleType are:

1 MAC00-Rx module (NanoPLC)
2 MAC00-FPx module (Profibus)
3 MAC00-FCx module (CAN-Open)

Example(s):

C++:

Read page 0:

```
VARIANT Data;
MacComm.NanoGet (255,0,Data);
```

BASIC:

Read Page 0:

```
dim Data as VARIANT
MacComm.NanoGet 255,0,Data
```

NanoGet([in] Address, [in] Page, [out] Data)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit unsigned short integer	Page	What page is to be read
VARIANT	Data	Array of Bytes (128 elements of type VT_UI1)

Description:
Reads a page from the RX module

Example(s):

C++:

Read page 0:

```
VARIANT Data;
MacComm.NanoGet (255,0,Data);
```

BASIC:

Read Page 0:

```
dim Data as VARIANT
MacComm.NanoGet 255,0,Data
```

NanoPut([in] Address, [in] Page, [in] Data)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit unsigned short integer	Page	What page is to be read (0-3)
VARIANT	Data	Array of Bytes (128 elements of type VT_UI1)

Description:
Sends a page to the RX module

Example(s):

C++:

Send page 0:

```
VARIANT Data;
MacComm.NanoPut (255,0,Data);
```

BASIC:

Send Page 0:

```
dim Data as VARIANT
MacComm.NanoPut 255,0,Data
```

NanoProgram([in] Address, [in] FileName)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
BSTR	FileName	HEX-file to be sent to RX module

Description:

Reads the HEX-file, and writes it to the RX module

Example(s):

C++:

Send "C:\File.hex":

```
MacComm.NanoProgram(255,"C:\\File.hex");
```

BASIC:

Send "C:\File.hex":

```
MacComm.NanoProgram 255,"C:\File.hex"
```

NanoWrite([in] Address)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)

Description:

Write pages to flash in RX module
NOTE: Waits for module to become ready before returning

Example(s):

C++:

Write Page

```
MacComm.NanoWrite(255);
```

BASIC:

Write Page

```
MacComm.NanoWrite 255
```

NanoStop([in] Address)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)

Description:

Stops the program

Example(s):

C++:

Stop progam:

```
MacComm.NanoStop(255);
```

BASIC:

Stop progam:

```
MacComm.NanoStop 255
```

NanoReset([in] Address)		
Parameters:		
Type	**Name**	**Description**
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
Description: Resets RX module (which also starts the program) Returns immediately after the command has been sent		
Example(s): **C++:** Reset module/start progam: MacComm.NanoReset(255); **BASIC:** Reset module/start progam: MacComm.NanoReset 255		

NanoResetWait([in] Address)		
Parameters:		
Type	**Name**	**Description**
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
Description: Resets RX module (which also starts the program) Returns when the module is ready		
Example(s): **C++:** Reset module/start progam: MacComm.NanoResetWait(255); **BASIC:** Reset module/start progam: MacComm.NanoResetWait 255		

NanoDebug
([in]Address,[out]State,[out]Inputs,[out]Outputs,[out]Time,[out]Count,[out]ModeCmd)

Parameters:

Type	Name	Description
16 bit unsigned short integer	Address	Address of the MAC motor (Use 255 to broadcast)
16 bit unsigned short integer	State	
16 bit unsigned short integer	Inputs	Inputs in binary format
16 bit unsigned short integer	Outputs	Outputs in binary format
32 bit unsigned long integer	Time	Time module has been running
32 bit unsigned long integer	Count	
16 bit unsigned short integer	ModeCmd	

Description:
Reads information for debugging

Example(s):

C++:

Reset module/start progam:

```
unsigned short State,Inputs,Outputs,ModeCmd;
unsigned long Time,Count;
MacComm.NanoDebug(255,State,Inputs,Outputs,Time,Count,ModeCmd);
```

BASIC:

Reset module/start progam:

```
dim State, Inputs, Outputs, ModeCmd as Integer
dim Time, Count as Long;
MacComm.NanoReset 255, State, Inputs, Outputs, Time, Count, ModeCmd
```

Installation

The MacComm OCX and required DLLs are installed automatically by running Setup.exe and following the onscreen prompts.
You have the option to install a Visual Basic sample and a LabVIEW sample along with the OCX.

It can also be done manually by copying the following Microsoft redistributable DLLs to the Windows\System folder:

- OLEAUT32.DLL
- OLEPRO32.DLL

MacComm.OCX should be placed in a directory called MacComm in the Windows folder, and registered with RegSvr32 i.e. “Regsvr32 C:\Windows\MacComm\MacComm.ocx”

Adding MacComm OCX to the program

Visual Basic 6

1. In the menu Projects click Components.
2. Make sure the "Selected Items Only" checkbox is NOT selected
3. Find "MacComm OCX Control module", and put a checkmark besides it, and click OK

The MacComm OCX is now available in the controls bar
When put on a form the properties page of the object can be used to set the startup values for the 2 properties (Retries and ComPort)

Visual C++ 6

1. In the menu "Projects" choose "Add To Project" and click "Components and Controls…"
2. Go into the folder "Registered ActiveX Controls" and click "MacComm Control"
3. Click Insert, and two times OK followed by a Close

The MacComm OCX is now available in the controls bar
When put on a dialog the properties page of the object can be used to set the startup values for the 2 properties (Retries and ComPort)

Visual .NET

1. In the menu "Tools" click "Customize Toolbox…"
2. Find "MacComm OCX Control module", and put a checkmark besides it, and click OK

The MacComm OCX is now available in the Toolbox
When put on a form the properties page of the object can be used to set the startup values for the 2 properties (Retries and ComPort)

Borland C++ Builder 6.0

1. In the menu "Component" click "Import ActiveX Control…"
2. Select "MacComm ActiveX Control module…" in the lists of components.
3. Press the "install…" button.
4. On the page "Into existing package" select the dclusr.bpk file (This should be default) and click "OK".
5. Select "yes" to rebuild the package.
6. The ActiveX should now be available in the tool palette on the ActiveX page.

LabVIEW 7.0

1. Place an ActiveX container on your Front Panel.
2. Right click it and select "Insert ActiveX object…"
3. Select MacComm Control from the list.
4. Connect it to a "Property node" and use this to setup the properties.
5. Connect it to an "Invoke node" and use this to call the methods.

Custom Errors:

Hex value:	Description
2000 0000	Serial port could not be initialized
2000 0001	Serial port is not open
2000 0002	Could not write required Bytes to serial port
2000 0003	Answer is not of expected length
2000 0004	Invalid accept from mac motor
2000 0005	Startsync error in reply
2000 0006	Address mismatch in reply
2000 0007	Parameter number mismatch in reply
2000 0008	Parameter length mismatch in reply
2000 0009	Inversion check failed on value
2000 000A	Endsync error in reply
2000 000B	Unspecified error
2000 000C	Error entering safe mode
2000 000D	Timeout