Wp2Comm.dll version 1.0.18.x, 09.03.2004
Beta version

Wp2comm.zip is a set of Wp2comm.dll, a library to simplify the communication with a controller using the serial port under Win95/98/Me/NT/2000/XP and several demo files including source code showing how to access Wp2comm.dll in C++, Delphi or VisualBasic. JavaScript-, VisualBasic-Script and HTML-code using an OCX built with VisualBasic is also included.

Containing files

- WP2COMM.DLL: DLL to simplify the communication with various types of SMC controllers. Wp2Comm.dll can use Wp2Comm.ini.
- WP2COMM.INI: Ini file to set some parameters (time-out, diagnostic protocol, commands used during initialization of controller, setting the mode wp2comm.dll uses to process commands). If default values are used, WP2COMM.INI is not necessary. Wp2Comm.ini must be in the same directory as Wp2Comm.dll.
- WP2DEMOB.EXE: Demo program, created with Borland C++-Builder Prof. version 5. Wp2demoB.exe uses Wp2comm.dll.
- WP2DEMOD.EXE: Demo program, created with Borland Delphi Prof. version 5. Wp2demoD.exe uses Wp2comm.dll.
- WP2DEMOV.EXE: Demo program, created with Microsoft Visual C++ version 6.0 standard. Wp2demoV.exe uses Wp2comm.dll.
- CONSOLE.EXE: A simple demo program that runs in a DOS box, created with Borland C++-Builder Prof. version 5. Console.exe uses Wp2comm.dll.
- Demo ActiveX, created with Microsoft Visual Basic version 5 Control Creating Edition. Wp2demo.ocx uses Wp2comm.dll.
- WP2DEMOA.HTM, WP2DEMOC.HTM , WP2DEMOD.HTM and WP2DEMOM.HTM are HTML files, built with Softquad HotMetal Pro 4. They use Wp2demo.ocx.
- WP2DEMOA.JS, WP2DEMOC.JS, WP2DEMOD.JS and WP2DEMOM.JS are JScript files, built using Microsoft Editor. They use Wp2demo.ocx.
- WP2DEMOA.VBS, WP2DEMOC.VBS, WP2DEMOD.VBS and WP2DEMOM.VBS are Visual Basic Script files, built using Microsoft Editor. They use Wp2demo.ocx.
- REGSVR32, a Microsoft tool to register/unregister ActiveX controls as Wp2comm.ocx
- MFC42D.DLL, MSVCRT.DLL, two runtime libraries form the Microsoft Visual C++ environment, necessary to run Wp2demoV.exe.
- SOURCECODE to build Wp2demoB.exe, Wp2demoD.exe, Wp2demoV.exe, Console.exe and Wp2demo.ocx.

Requirements

General: Microsoft Windows 95/98/Me/NT/2000/XP with Intel 486 processor or better with a controller connected to a serial port.

Special requirements:

- WP2COMM.OCX has to be registered before it can be used. To register Wp2comm.ocx the tool RegSvr32.exe can be used. The command line is *Regsvr32 Wp2comm.ocx*. If you are using an older version of Wp2Comm.ocx you have to unregister it before you can use the new one. To unregister Wp2comm.ocx call *Regsvr32 -u Wp2comm.ocx*. Rebuilding Wp2comm.ocx with Visual Basic will register it automatically. Wp2comm.dll (and Wp2Comm.ini) must be copied to the system directory (usually c:\windows\system). Also Wp2comm.ini must be in the search path for Wp2comm.ocx to find it.
- The HTML files require a browser capable of executing ActiveX controls (i.e. Microsoft Internet Explorer 3 or better). The security settings have to be adjusted to allow scripting and the use of ActiveX controls. Wp2DemoA.htm, Wp2DemoC.htm, Wp2DemoD.htm and Wp2DemoM.htm refer to the Wp2Demo.ocx demo file located in the directory C:\Wp2Comm. If the demo files are located in a different directory, the html files have to be edited.
- The JavaScript/JScript (*.JS) and Visual Basic script (*.VBS) files require the Microsoft Windows Scripting Host to be installed. The Windows Scripting Host is part of a standard Windows 98/Me installation und can be installed separately under Windows 95 and Windows NT. The windows scripting host requires Microsoft Internet Explorer.
- WP2DEMOM.HTM, WP2DEMOM.JS and WP2DEMOM.VBS are designed to run with a MCX or MC2000 with 3 axes at a Baudrate of 19200. WP2DEMOC.HTM, WP2DEMOC.JS and WP2DEMOC.VBS are designed to run with a Corvus with 3 axes at a Baudrate of 57600. WP2DEMOA.HTM, WP2DEMOA.JS and WP2DEMOA.VBS expect an Alpha controller with 4 axes and 19200 Baud. WP2DEMOD.HTM, WP2DEMOD.JS and WP2DEMOD.VBS expect a DeltaPC with 4 axes and 9600 Baud.
- The HTML-, JS- and VBS-files expect the controller at COM2. Otherwise the script files have to be edited.

How to execute the demo files

1. Unzip WP2COMM.ZIP to a drive/directory of your choice using the stored directory names, preferably C:\. A directory WP2COMM will be created, containing the executables and a directory containing the source code.

2. To use Wp2Comm.ocx, Wp2CommA.htm, Wp2CommC.htm, Wp2CommD.htm, Wp2CommM.htm, Wp2DemoA.js, Wp2DemoC.js, Wp2DemoD.js, Wp2DemoM.js, Wp2DemoA.vbs, Wp2DemoC.vbs, Wp2DemoD.vbs and Wp2DemoM.vbs see requirements.

3. Connect the controller to a serial port an switch it on.

4. Start Wp2demoB.exe, Wp2demoD.exe, Wp2demoV.exe. These files are built using different environments and provide almost similar functionality when used with MCX, MC2000, Corvus or DeltaPC. Connected to an Alpha, Taurus, Pegasus, Orion or Pollux controller the Test routine (Test button), provided by Wp2DemoB.exe is not yet available under Wp2DemoD.exe and Wp2DemoV.exe. Wp2demo.ocx is invisible at runtime. Rebuilding Wp2demo.ocx with the supplied source code will create a visible version. Executing the visible version of Wp2comm.ocx i.e. in an Internet Browser will provide almost similar functions as the Wp2demoX.exe files.
4.1. Select **Port**, **Baudrate** and **Axes**
4.2. **Init**ialize Wp2comm.dll
4.3. Connect to the controller (**Open**)
4.4. Run the **test** routine (if available). This routine shows the use of various functions provided by Wp2comm.dll. The status of the test routine, the commands and the replies of the controller are shown in the textfield.
4.5. **Execute** a Venus command (in Wp2demoB.exe and Wp2demoD.exe by pressing enter). The **lines** parameter has to be set to the appropriate number of lines the controller will reply processing this command. Lines are separated by a CR/LF pair.
Examples for a controller with 3 axes:
Move 10 mm in each direction: Enter the Venus command *10 10 10 m*, lines = *0*
Get the version of the firmware: Enter *version*, lines =*1*
Get the limits of the stage: Enter *getlimit*, lines = *3*

How to use the source code

After unzipping Wp2comm.zip using the stored directory names a wp2comm directory will be created. This wp2comm directory contains a source directory. The source directory contains the four directories CBuildr5, Delphi5, VBas5CCE, VisualC6. They contain the Borland C++Builder, Borland Delphi, Microsoft Visual Basic 5 CCE and Microsoft Visual C++ source code. To rebuild the demo files open the desired project file, if necessary adapt to your compiler version, and recompile it.

Simple sample

This sample sets controller type to MCX/MC2000, the number of axes to 3, the Baudrate to 19200, opens COM2 and connects to the controller, executes a move command, that lets the axes move to position 0, then closes the controller and returns.
Console.exe is a small program that runs in a DOS box and is based on this code with some additional command line parameters (try *console ?*) and error processing. Wp2Comm.dll is linked statically by invoking Wp2Comm.lib in the project.

```
int main(int argc, char **argv)
{
InitController(1,3,2,19200,0,0, 1050884);
OpenController();
MoveAbsolute(”0”,”0”,”0”,NULL);
CloseController();
return(0);
```

```
}
```

Function summary

A: Used with Alpha, Taurus, Pegasus or Orion controller
D: Used with DeltaPC
M: Used with MCX, MC2000 or Corvus controller
P: Used with Pollux controller

Function	See also	A	D	M	P	Venus command
InitController		X	X	X	X	-
OpenController		X	X	X	X	-
CloseController		X	X	X	X	-
ResetComm		X	X	X	X	-
SetDecimalSeparator		X	X	X	X	
PreprocessReply		X	X	X	X	
ExecuteCommand		X	X	X	X	-
GetReply		X	X	X	X	-
AbortCommand		X	X	X	X	[Ctrl]+[C]
Calibrate	RangeMeasure		X	X		calibrate, cal
CalibrateA	RangeMeasureA	X			X	ncalibrate, ncal
ClearParameterStack	GetParamsOnStack		X	X		clear
ClearParameterStackA	GetParamsOnStackA	X			X	nclear
GetAcceleration	SetAcceleration		X	X		getaccel, ga
GetAccelerationA	SetAccelerationA	X			X	getnaccel, gna
GetAxisMode	SetAxisMode		X	X		getaxis
GetAxisModeA	SetAxisModeA	X				getnaxis
GetError			X	X		geterror, ge
GetErrorA		X			X	getnerror, gne
GetJoystickVelocity	SetJoystickVelocity			X		getjoyspeed
GetJoystickVelocityA	SetJoystickVelocityA	X				getnjoyspeed
GetLimits			X	X		getlimit
GetLimitsA		X			X	getnlimit
GetParamsOnStack	ClearParameterStack			X		gsp
GetParamsOnStackA	ClearParameterStackA	X			X	gnsp/ngsp
GetPos			X	X		pos, p
GetPosA		X			X	npos, np
GetStatus			X	X		status, st
GetStatusA		X			X	nstatus, nst
GetVelocity	SetVelocity		X	X		getvel, gv
GetVelocityA	SetVelocityA	X			X	getnvel, gnv
Identify			X	X		identify
IdentifyA		X				nidentify
JoystickDisable	JoystickEnable			X		joystick, j
JoystickDisableA	JoystickEnableA	X				njoystick, nj
JoystickEnable	JoystickDisable			X		joystick, j
JoystickEnableA	JoystickDisableA	X				njoystick, nj
MoveAbsolute			X	X		move, m
MoveAbsoluteA		X			X	nmove, nm
MoveRelative			X	X		rmove, r
MoveRelativeA		X			X	nrmove, nr
MoveAbsoluteAutoReply			X	X		move, m
MoveAbsoluteAutoReplyA		X			X	nmove, nm
MoveRelativeAutoReply			X	X		rmove, r
MoveRelativeAutoReplyA		X			X	nrmove, nr
RangeMeasure			X	X		rangemeasure, rm

RangeMeasureA		X			X	nrangemeasure, nrm
SetAcceleration	GetAcceleration		X	X		setaccel, sa
SetAccelerationA	GetAccelerationA	X			X	setnaccel, sna
SetAxisMode	GetAxisMode		X	X		getaxis
SetAxisModeA	GetAxisModeA	X				getnaxis
SetJoystickVelocity	GetJoystickVelocity			X		joyspeed, js
SetJoystickVelocityA	GetJoystickVelocityA	X				njoyspeed, njs
SetOrigin			X	X		setpos
SetOriginA		X			X	setnpos
SetVelocity	GetVelocity		X	X		setvel, sv
SetVelocityA	GetVelocityA	X			X	setnvel, snv

WP2COMM.DLL functions/procedures in C/Pascal notation

General:

- Return values of all functions: 0 if OK, otherwise errorcode
- LPSTR: Buffer size should be >255 Bytes

InitController

DWORD InitController(DWORD ControllerMode, DWORD Axes, DWORD ComPort, DWORD Baudrate, HWND UserWin, UINT AsyncMsg, DWORD Mode);
function InitController(ControllerMode: DWORD; Axes: DWORD; ComPort: DWORD; Baudrate: DWORD; UserWin: HWND; AsyncMsg: UINT; Mode: DWORD):DWORD;

Descr.: Sets the main parameters, Wp2Comm.dll uses to communicate with the controller. InitController must be called first.

Params: *ControllerMode*: Type of controller, MCX or MC2000 = 1, DeltaPC = 2, Alpha = 4, Taurus = 8, Pegasus = 16, Orion = 32, Corvus = 64, Pollux = 128. If this parameter is set to 0 Wp2comm.dll will try to read the value from Wp2comm.ini, section *Startup*, item *CMode*. *CMode* can be *Default*, *Delta* (or *Delta-PC*), *Alpha*, *Taurus*, *Pegasus*, *Orion*, *Corvus, Pollux*.

Axes: Number of axes (usually 3 or 4)

ComPort: No. 1 .. 8 (e.g. 1 = COM1)

Baudrate: 300 ... 115200. For DeltaPC the Baudrate must be 9600, for Pollux the Baudrate must be 19200

UserWin: Handle of the window to post AsyncMsg to. Can be 0 if no asynchronous mode is used (default)

AsyncMsg: Message posted in asynchronous mode to signal that the controller replied and the data can be processed. Can be 0 if no asynchronous mode is used, otherwise a WM_USER value.

Mode: Used to set the mode, the DLL uses to process Venus-commands internally, default value for synchronous mode is 1050884 (or 2308), for asynchronous mode 1051140 (or 2564).

OpenController

DWORD OpenController(void);
function OpenController: DWORD;

Descr.: Opens the COM-port and connects to the controller. Make sure to have called InitController() before executing OpenController().

Params: None

CloseController

DWORD CloseController(void);
function CloseController: DWORD;

Descr.: Closes the port, clears buffers etc., includes ResetComm()

Params: None

ResetComm

DWORD ResetComm(void);
function ResetComm: DWORD;

Descr.: Should be called in case of communication errors. Clears COM-buffers, program buffers, resets errors

Params: None

SetDecimalSeparator

DWORD SetDecimalSeparator(BYTE DecimalSep);
function SetDecimalSeparator(DecimalSep: BYTE): DWORD;

Descr.: By default Wp2comm.dll preprocesses data returned by the controller and replaces the '.' by the decimalseparator as set in Windows. SetDecimalSeparator() allows to change this behaviour.

Params: *DecimalSep*: decimal separator

0	Use decimal separator of windows (default)
0xFF	Leave decimal separator unchanged
Every other value	Use this byte as decimal separator

PreprocessReply

DWORD PreprocessReply(BYTE Active);
function PreprocessReply(Active: BYTE): DWORD;

Descr.: By default Wp2comm.dll preprocesses the data returned by the controller, eliminates leading and trailing spaces, CR/LF and replaces the '.' by the decimalseparator as set in Windows(default) or by SetDecimalSeparator().

Params: *Active*:

0	Don't preprocess the data
<>0	Preprocess data (default)

ExecuteCommand

DWORD ExecuteCommand(LPSTR Command, DWORD LinesExpected, LPSTR Reply);
function ExecuteCommand(Command: LPSTR; LinesExpected: DWORD; Reply: LPSTR):DWORD;

Descr.: Executes a Venus command

Params: *Command*: The command itself

LinesExpected: Number of lines, the controller will return processing the command (a line is terminated by CR\LF)

Reply: Data returned by the controller

GetReply

DWORD GetReply(LPSTR Reply);
function GetReply(Reply: LPSTR):DWORD;

Descr.: Retrieves data sent by the controller; used in asynchronous mode

Params: *Reply*: Buffer to receive the data

AbortCommand

DWORD AbortCommand(void);
function AbortCommand: DWORD;

Descr.: Sends a Break ([Ctrl]+[C]) to the controller, aborting the command (e.g. terminating move operation) and clearing the buffers

Params: None

See Venus command: [Ctrl]+[C]

Calibrate

DWORD Calibrate(void);
function Calibrate: DWORD;

Descr.: Calibrates the controller; see RangeMeasure

Params: None

See Venus command: calibrate/cal

CalibrateA

DWORD CalibrateA(DWORD Axis);
function CalibrateA(Axis:DWORD): DWORD;

Descr.: Calibrates the axis; see RangeMeasure

Params: *Axis*: 1 ... no. of axes

See Venus command: ncalibrate/ncal

ClearParameterStack

DWORD ClearParameterStack(void);
function ClearParameterStack: DWORD;

Descr.: Clears the controller stack

Params: None

See Venus command: clear

ClearParameterStackA

DWORD ClearParameterStackA(DWORD Axis);
function ClearParameterStackA(Axis: DWORD): DWORD;

Descr.: Clears the parameter stack of an axis

Params: *Axis*: 1 ... no. of axes

See Venus command: nclear

GetAcceleration

DWORD GetAcceleration(LPSTR Accel);
function GetAcceleration(Accel: LPSTR):DWORD;

Descr.: Gets the acceleration of all axes

Params: *Accel*: acceleration in current unit

See Venus command: getaccel/ga

GetAccelerationA

DWORD GetAccelerationA(LPSTR Accel, DWORD Axis);
function GetAccelerationA(Accel: LPSTR, Axis: DWORD):DWORD;

Descr.: Gets the acceleration of an axis

Params: *Accel*: acceleration in current unit. For Pollux in mm/s^2.

Axis: 1 ... no. of axes

See Venus command: getnaccel/gna

GetAxisMode

DWORD GetAxisMode(LPDWORD Axis1Mode, LPDWORD Axis2Mode, LPDWORD Axis3Mode, LPDWORD Axis4Mode);
function GetAxisMode(Axis1Mode, Axis2Mode, Axis3Mode, Axis4Mode: LPDWORD):DWORD;

Descr.: Gets the mode of the axes

Params: *Axis1Mode, Axis2Mode, Axis3Mode, Axis4Mode*: values defining the mode of each axis. Can be NULL/nil if the axis doesn't exist.

See Venus command: getaxis

GetAxisModeA

DWORD GetAxisModeA(LPDWORD Mode, DWORD Axis);
function GetAxisModeA(Mode:LPDWORD, Axis:DWORD):DWORD;

Descr.: Gets the mode of an axis (active, inactive ...)

Params: *Mode*: 0 ... 2, defining the mode of the axis

Axis: 1 ... no. of axes

See Venus command: getnaxis

GetError

DWORD GetError(LPDWORD Error);

function GetError(Error: LPDWORD):DWORD;

Descr.: Gets the errorcode of the controller

Params: *Error*: Error code

See Venus command: geterror/ge

GetErrorA

DWORD GetErrorA(LPDWORD Error, DWORD Axis);
function GetErrorA(Error: LPDWORD; Axis: DWORD):DWORD;

Descr.: Gets the errorcode of an axis

Params: *Error*: Venus error code

Axis: 1 ... no. of axes

See Venus command: getnerror/gne

GetJoystickVelocity

DWORD GetJoystickVelocity(LPSTR JoyVel);
function GetJoystickVelocity(JoyVel: LPSTR):DWORD;

Descr.: Gets the joystick velocity

Params: *JoyVel*: joystick velocity

See Venus command: getjoyspeed

GetJoystickVelocityA

DWORD GetJoystickVelocityA(LPSTR JoyVel, DWORD Axis);
function GetJoystickVelocityA(JoyVel: LPSTR; Axis: DWORD):DWORD;

Descr.: Gets the joystick velocity of an axis

Params: *JoyVel*: joystick velocity

Axis: 1 ... no. of axes

See Venus command: getnjoyspeed

GetLimits

DWORD GetLimits(LPSTR A1Min, LPSTR A1Max, LPSTR A2Min, LPSTR A2Max, LPSTR A3Min, LPSTR A3Max, LPSTR A4Min, LPSTR A4Max);
function GetLimits(A1Min, A1Max, A2Min, A2Max, A3Min, A3Max, A4Min, A4Max: LPSTR):DWORD;

Descr.: Gets the limits of the axes

Params: *A1Min, A1Max, A2Min, A2Max, A3Min, A3Max, A4Min, A4Max*: Values, defining the min/max values of the stage. Can be NULL/nil if the axis doesn't exist.

See Venus command: getlimits

GetLimitsA

DWORD GetLimitsA(LPSTR Min, LPSTR Max, DWORD Axis);
function GetLimitsA(Min, Max: LPSTR; Axis: DWORD):DWORD;

Descr.: Gets the limits of an axis

Params: *Min, Max*: Values, defining the min/max values of the axis

Axis: 1 ... no. of axes

See Venus command: getnlimits

GetParamsOnStack

DWORD GetParamsOnStack(LPDWORD Value);
function GetParamsOnStack(Value: LPDWORD):DWORD;

Descr.: Gets the number of parameters on the stack of the controller

Params: *Value*: Number of parameters on the stack

See Venus command: gsp

GetParamsOnStackA

DWORD GetParamsOnStackA(LPDWORD Value, DWORD Axis);
function GetParamsOnStackA(Value: LPDWORD; Axis: DWORD):DWORD;

Descr.: Gets the number of parameters on the stack of an axis

Params: *Value*: Number of parameters on the stack

Axis: 1 ... no. of axes

See Venus command: gnsp

GetPos

DWORD GetPos(LPSTR Axis1Pos, LPSTR Axis2Pos, LPSTR Axis3Pos, LPSTR Axis4Pos);
function GetPos(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

Descr.: Gets the position of the axes

Params: *Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos*: coordinates in current units. Can be NULL/nil if the axis doesn't exist.

See Venus command: pos/p

GetPosA

DWORD GetPosA(LPSTR Pos, DWORD Axis);
function GetPosA(Pos:LPSTR; Axis:DWORD):DWORD;

Descr.: Gets the position of an axis

Params: *Pos*: coordinate in current units. For Pollux in mm.

See Venus command: npos/np

GetStatus

DWORD GetStatus(LPDWORD Status);
function GetStatus(Status: LPDWORD):DWORD;

Descr.: Gets the controller status

Params: *Status*: Status of the controller

See Venus command: status/st

GetStatusA

DWORD GetStatusA(LPDWORD Status, DWORD Axis);
function GetStatusA(Status: LPDWORD; Axis: DWORD):DWORD;

Descr.: Gets the status of an axis

Params: *Status*: Status of the axis

Axis: 1 ... no. of axes

See Venus command: nstatus/nst

GetVelocity

DWORD GetVelocity(LPSTR Vel);
function GetVelocity(Vel: LPSTR):DWORD;

Descr.: Gets the velocity of all axes

Params: *Vel*: velocity in current unit

See Venus command: getvel/gv

GetVelocityA

DWORD GetVelocityA(LPSTR Vel, DWORD Axis);
function GetVelocityA(Vel: LPSTR; Axis: DWORD):DWORD;

Descr.: Gets the velocity of an axis

Params: *Vel*: velocity in current unit. For Pollux in mm/s.

Axis: 1 ... no. of axes

See Venus command: getnvel/gnv

Identify

DWORD Identify(LPSTR Id);
function Identify(Id: LPSTR):DWORD;

Descr.: Gets the identify string of the controller

Params: *Id*: identify string

See Venus command: identify

IdentifyA

DWORD IdentifyA(LPSTR Id, DWORD Axis);

function IdentifyA(Id: LPSTR; Axis: DWORD):DWORD;

Descr.: Gets the identify string of an axis

Params: *Id*: identify string

Axis: 1 ... no. of axes

See Venus command: nidentify

JoystickDisable

DWORD JoystickDisable(void);
function JoystickDisable: DWORD;

Descr.: Turns joystick off

Params: None

See Venus command: 0 joystick/0 j

JoystickDisableA

DWORD JoystickDisableA(DWORD Axis);
function JoystickDisableA(Axis: DWORD): DWORD;

Descr.: Turns joystick off

Params: *Axis*: 1 ... no. of axes

See Venus command: 0 njoystick/0 nj

JoystickEnable

DWORD JoystickEnable(void);
function JoystickEnable: DWORD;

Descr.: Turns joystick on

Params: None

See Venus command: 1 joystick/1 j

JoystickEnableA

DWORD JoystickEnableA(DWORD Axis);
function JoystickEnableA(Axis: DWORD): DWORD;

Descr.: Turns joystick on

Params: *Axis*: 1 ... no. of axes

See Venus command: 1 njoystick/1 nj

MoveAbsolute

DWORD MoveAbsolute(LPSTR Axis1Pos,LPSTR Axis2Pos,LPSTR Axis3Pos, LPSTR Axis4Pos);
function MoveAbsolute(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

Descr.: Moves to the desired position

Params: *Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos*: coordinates in current units. Can be NULL/nil if the axis doesn't exist.

See Venus command: move/m

Remarks: While a move operation is performed it is possible to determine the position with GetPos() and the status with GetStatus()

MoveAbsoluteA

DWORD MoveAbsoluteA(LPSTR Pos, DWORD Axis);
function MoveAbsoluteA(Pos:LPSTR; Axis: DWORD):DWORD;

Descr.: Moves to the desired position

Params: *Pos*: coordinate in current units. For Pollux in mm or nm.

Axis: 1 ... no. of axes. For Pollux synchronous positioning is supported.

See Venus command: nmove/nm

Remarks: While a move operation is performed it is possible to determine the position with GetPosA() and the status with GetStatusA()

MoveAbsoluteAutoReply

DWORD MoveAbsoluteAutoReply(LPSTR Axis1Pos, LPSTR Axis2Pos, LPSTR Axis3Pos, LPSTR Axis4Pos);
function MoveAbsoluteAutoReply(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

Descr.: Moves to the desired position; after reaching the target, returns a signal

Params: *Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos*: coordinates in current units

See Venus command: move/m, status/st

Remarks: The AutoReply-functions return only, when a desired target is reached. To respond to user input while waiting a PeekMessage - TranslateMessage - DispatchMessage routine is performed. GetPos() and Status() cannot be used.

MoveAbsoluteAutoReplyA

DWORD MoveAbsoluteAutoReplyA(LPSTR Pos, DWORD Axis);
function MoveAbsoluteAutoReplyA(Pos: LPSTR; Axis: DWORD):DWORD;

Descr.: Moves to the desired position; after reaching the target, returns a signal

Params: *Pos*: coordinate in current units. For Pollux in mm or nm.

Axis: 1 ... no. of axes.For Pollux synchronous positioning is not supported.

See Venus command: nmove/nm, nstatus/nst

Remarks: The AutoReply-functions return only, when a desired target is reached. To respond to user input while waiting a PeekMessage - TranslateMessage - DispatchMessage routine is performed. GetPos() and Status() cannot be used.

MoveRelative

DWORD MoveRelative(LPSTR Axis1Pos,LPSTR Axis2Pos,LPSTR Axis3Pos, LPSTR Axis4Pos);
function MoveRelative(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

Descr.: Moves relative starting from the current position

Params: *Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos*: coordinates in current units. Can be NULL/nil if the axis doesn't exist.

See Venus command: rmove/rm

Remarks: While a move operation is performed it is possible to determine the position with GetPos() and the status with GetStatus()

MoveRelativeA

DWORD MoveRelativeA(LPSTR Pos, DWORD Axis);
function MoveRelativeA(Pos: LPSTR; Axis: DWORD):DWORD;

Descr.: Moves relative starting from the current position

Params: *Pos*: coordinate in current units. For Pollux in mm or nm.

Axis: 1 ... no. of axes. For Pollux synchronous positioning is supported.

See Venus command: nrmove/nrm

Remarks: While a move operation is performed it is possible to determine the position with GetPos() and the status with GetStatus()

MoveRelativeAutoReply

DWORD MoveRelativeAutoReply(LPSTR Axis1Pos,LPSTR Axis2Pos,LPSTR Axis3Pos, LPSTR Axis4Pos);
function MoveRelativeAutoReply(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

Descr.: Moves relative, starting from the current position; after reaching the target, return a signal

Params: *Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos*: coordinates in current units. Can be NULL/nil if the axis doesn't exist.

See Venus command: rmove/rm

Remarks: The AutoReply-functions return only, when a desired target is reached. To respond to user input while waiting a PeekMessage - TranslateMessage - DispatchMessage routine is performed. Position and Status cannot be used.

MoveRelativeAutoReplyA

DWORD MoveRelativeAutoReplyA(LPSTR Pos, DWORD Axis);
function MoveRelativeAutoReplyA(Pos: LPSTR; Axis: DWORD):DWORD;

Descr.: Moves relative, starting from the current position; after reaching the target, return a signal

Params: *Pos*: coordinate in current units

Axis: 1 ... no. of axes. For Pollux synchronous positioning is not supported.

See Venus command: nrmove/nrm

Remarks: The AutoReply-functions return only, when a desired target is reached. To respond to user input while waiting a PeekMessage - TranslateMessage - DispatchMessage routine is performed. Position and Status cannot be used.

RangeMeasure

DWORD RangeMeasure(void);
function RangeMeasure: DWORD;

Descr.: Measures the stage to determine the limits

Params: None

See Venus command: rangemeasure/rm

RangeMeasureA

DWORD RangeMeasureA(DWORD Axis);
function RangeMeasureA(Axis: DWORD): DWORD;

Descr.: Measures the axis to determine the limit

Params: *Axis*: 1 ... no. of axes

See Venus command: nrangemeasure/nrm

SetAcceleration

DWORD SetAcceleration(LPSTR Accel);
function SetAcceleration(Accel: LPSTR):DWORD;

Descr.: Sets the acceleration of all axes

Params: *Accel*: acceleration in current unit

See Venus command: setaccel/sa

SetAccelerationA

DWORD SetAccelerationA(LPSTR Accel, DWORD Axis);
function SetAccelerationA(Accel: LPSTR; Axis: DWORD):DWORD;

Descr.: Sets the acceleration of an axis

Params: *Accel*: acceleration in current unit. For Pollux in mm/s^2 or μm/s^2.

Axis: 1 ... no. of axes

See Venus command: setnaccel/sna

SetAxisMode

DWORD SetAxisMode(LPSTR Axis1Mode, LPSTR Axis2Mode, LPSTR Axis3Mode, LPSTR Axis4Mode);
function SetAxisMode(Axis1Mode, Axis2Mode, Axis3Mode, Axis4Mode: LPSTR):DWORD;

Descr.: Sets the mode of the axes

Params: *Axis1Mode, Axis2Mode, Axis3Mode, Axis4Mode*: 0 ... 2, defining the mode of each axis. Can be NULL/nil if the axis doesn't exist.

See Venus command: setaxis

SetAxisModeA

DWORD SetAxisModeA(LPSTR Mode, DWORD Axis);
function SetAxisModeA(Mode: LPSTR; Axis: DWORD):DWORD;

Descr.: Sets the mode of the axes

Params: *Mode*: 0 ... 2, defining the mode of the axis

Axis: 1 ... no. of axes

See Venus command: setnaxis

SetJoystickVelocity

DWORD SetJoystickVelocity(LPSTR JoyVel);
function SetJoystickVelocity(JoyVel: LPSTR):DWORD;

Descr.: Sets joystick velocity

Params: *JoyVel*: joystick velocity

See Venus command: joyspeed/js

SetJoystickVelocityA

DWORD SetJoystickVelocityA(LPSTR JoyVel, DWORD Axis);
function SetJoystickVelocityA(JoyVel: LPSTR; Axis: DWORD):DWORD;

Descr.: Sets joystick velocity of an axis

Params: *JoyVel*: joystick velocity

Axis: 1 ... no. of axes

See Venus command: njoyspeed/njs

SetOrigin

DWORD SetOrigin(void);
function SetOrigin: DWORD;

Descr.: Sets the current position as zero

Params: None

See Venus command: setpos

SetOriginA

DWORD SetOriginA(DWORD Axis);
function SetOriginA(Axis: DWORD): DWORD;

Descr.: Sets the current position of an axis as zero

Params: *Axis*: 1 ... no. of axes

See Venus command: setnpos

SetVelocity

DWORD SetVelocity(LPSTR Vel);
function SetVelocity(Vel: LPSTR):DWORD;

Descr.: Set the velocity of all axes

Params: *Vel*: Velocity in current unit

See Venus command: setvel/sv

SetVelocityA

DWORD SetVelocityA(LPSTR Vel, DWORD Axis);
function SetVelocityA(Vel: LPSTR; Axis: DWORD):DWORD;

Descr.: Sets the velocity of an axis

Params: V*el*: Velocity in current unit. For Pollux in mm/s or nm/s

Axis: 1 ... no. of axes

See Venus command: setnvel/snv

Some considerations:

Executing functions that retrieve data from the controller:

During synchronous processing of a command that expects a reply from the controller, WP2COMM.DLL will wait a fixed time for this reply. The time-out is by default 1000ms and can be changed by setting the value *TimeOut* in WP2COMM.INI section Startup. Some commands take longer than others to be processed (up to 200ms, e.g. getlimit, getjoyspeed) others take just 20ms or less. During this WaitTime the program will generally not respond to user input, resulting in a program lock. Normally this WaitTime is short enough, not to be recognized
by the user.

If this lock is not acceptable, there are two ways to avoid this behavior.

1. Use of asynchronous mode.
In this mode the function will return immediately after sending the Venus-command to the controller, not waiting for the reply. When the reply is received by WP2COMM.DLL, a WM_USER message will be sent to an application window. Then GetReply() can be used to get the replied data. GetReply() will return the data as received from the controller without preprocessing it. Extracting the values has to be done separately.
Example:
Processing GetPos() in synchronous mode will return up to four strings in the parameters of GetPos(), e.g. '11.23400', '20.00000', '3.50000' and '0.00000'.
GetPos() executed in asynchronous will not return valid data. After posting AsyncMsg, GetReply() will return a single string like '11.23400 20.00000 3.5000 0.00000', where the values are still to be extracted.

2. Change of the Mode parameter in InitController(), which is not recommended (and not documented), because it takes too much processor time.

Functions that are treated different are Calibrate(), RangeMeasure(), MoveAbsoluteAutoReply() and MoveRelativeAutoReply(), for which a time-out cannot be specified. WP2COMM.DLL will wait infinitely, processing a PeekMessage - TranslateMessage - DispatchMessage loop. If an error occurs while processing these functions, AbortCommand must be used.
A way to force WP2COMM.DLL to wait infinitely is to append a '0 relative move' command (that does nothing) and a *status* command at the end of the command line processed by ExecuteCommand().
Example for a MC2000/MCX/DeltaPC controller with 3 axes: To wait for a move to be finished, you could add *0 0 0 r st* to the command line, resulting in *100 200 300 m 0 0 0 r st* and waiting for 1 line to be replied. The function will not return until the move is done and the status is returned by the controller. Instead if you try to process *100 200 300 400 m st* usually a timeout will occur.

function OpenController:

While establishing a communication with a controller, WP2COMM.DLL is executing a few Venus-commands to set the controller to a defined mode. For the MCX and MC2000 these commands are: *0 mode*, *0 setioport*, *ge*, *clear*, *identify*, *ico*.
For a DeltaPC the *setdim* command is used, to set the controller to the number of axes you are using.

*ico*can be set by the item IcoAtConnect=0/1 in WP2COMM.INI, Section Startup.

0 mode, *0 setioport*, *ge*, *clear* can be changed by setting the Values in WP2COMM.INI, section startup to
UseDefaultConnectString=0
ConnectString=<desired commands, can be left blank>
ConnectReply=<expected reply, can be left blank>

This initialization routine can be skipped completely by setting
UseDefaultConnectString=0
ConnectString=Skip

The are no initialization commands used when connecting to a Pollux controller.

Error codes

NOT ENOUGH MEMORY	2
NOT CONNECTED	3
INVALID PORT	4
INVALID COMMAND	5
INVALID CONTROLLER	6
INVALID AXES	7
INVALID AXIS	8
INVALID BAUDRATE	9
WAITING FOR REPLY	10
INVALID POSDATA	11
INVALID DATA	12
INVALID REPLY	13
INVALID POINTER	14
NO REPLY FROM CONTROLLER	15
INVALID IDENTIFY	16
INVALID INTERPRETER	17
ERROR SENDING BREAK CHAR	18
WAIT TIMEOUT	19
WAIT FAILED	20
BREAK DETECTED	21
VALIDATE NOT READY	22
FUNCTION NOT AVAILABLE	23
EVERR	24