Messen – Steuern
Positionieren

Dr. Kassen GmbH

Beim Eberacker 3 · D-35633 Lahnau
Tel.: +49 64 41-6 50 05-0
Fax: +49 64 41-6 50 05-29
email: info@itknet.de
Internet: http://www.itknet.de

Documentation

Dynamik Link Library

WP2COMM.DLL

Version 1.0.13

Contents

Overview

WP2COMM.DLL, is a function library to simplify the communication with the ITK controllers Corvus, Pegasus, MCX-2 and DeltaPC.
The dll supports the serial ports under Win95/98/Me/NT/2000/XP.
Several demo files including the source code are showing how to access WP2COMM.DLL in C++, Delphi or VisualBasic.

In Folder “Special” the procedure is described to manage two controllers with WP2COMM.DLL at different Com Ports in the same application.

Included also a demonstration to use JavaScript-, Visual Basic-Script and HTML-code with an OCX that is built with VisualBasic.
The OCX is only a example. It is not fully developed like WP2COMM.DLL

Containing files

- WP2COMM.DLL:
  DLL to simplify the communication with controllers. WP2COMM.DLL uses Wp2Comm.ini.

- WP2COMM.INI:
  Init file to set parameters (time-out, diagnostic protocol, commands to initialise the 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 3. Wp2demoD.exe uses WP2COMM.DLL.

- WP2DEMOV.EXE:
  Demo program, created with Microsoft Visual C++ version 4.0 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.

- WP2comm.OCX:
  Demo ActiveX, created with Microsoft Visual Basic version 5 Control Creating Edition. Wp2comm.ocx uses WP2COMM.DLL.

- WP2DEMOA.HTM, WP2DEMOC.HTM , WP2DEMOD.HTM and WP2DEMOM.HTM
  are HTML files, built with Softquad HotMetal Pro 4. They use Wp2comm.ocx.

- WP2DEMOA.JS, WP2DEMOC.JS, WP2DEMOD.JS and WP2DEMOM.JS
  are JScript files, built using Microsoft Editor. They use Wp2comm.ocx.

- WP2DEMOA.VBS, WP2DEMOC.VBS, WP2DEMOD.VBS and WP2DEMOM.VBS
  are Visual Basic Script files, built using Microsoft Editor. They use Wp2comm.ocx.

- REGSVR32,
  a Microsoft tool to register/unregister ActiveX controls as Wp2comm.ocx

- MFC40D.DLL, MSVCR40D.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 Wp2comm.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 Wp2comm.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 and 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 or Orion controller the Test routine (Test button), provided by Wp2DemoB.exe is not yet available under Wp2DemoD.exe and Wp2DemoV.exe. Wp2comm.ocx is invisible at runtime. Rebuilding Wp2comm.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.

a. Select **Port**, **Baudrate** and **Axes**
b. **Init**ialize **WP2COMM.DLL**
c. Connect to the controller (**Open**)
d. 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 text field.
e. **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.

Examples for a controller with 3 axes, moving 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, Delphi3, VBas5CCE, VisualC4. 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/Corvus, 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);
}
```

What are Venus commands

Venus-1 / Venus-2 is an interpreter language to program all functions of the controllers.
Venus commands consist of ASCII-signs which are interpreted in the controller and immediately executed.
A software development surrounding to produce the control programs is not needed.
The commands can be produced by any Host and whatever programming language you are using, on condition that there is an access to the RS-232.
In the simplest way the commands are directly transmitted to the controller via an ASCII terminal.
The WP2COMM.DLL library is using a subset of this commands.

Venus-1 / Venus-2 differences

Venus-2 has been developed on the basis of the interpreter language Venus-1. The fundamental command construction is identical.
The expansion was necessary because the fundamental structures of Venus-1 are designed for controller with at most three linear interpolated axes; but the controller **Pegasus** supports any number of independent axes (n). Venus-1 commands which structure is designed for operating n-axes are taken over in Venus-2 without any syntax alteration.
The other commands are expanded with the addition "n".
For example: The Venus-1 command ***cal*** which has at the same time an effect on three axes has become the axis specific command ***ncal*** in Venus-2, ***move*** has become ***nmove*** etc.

Function summary

In the following table the provided functions of the WP2COMM.DLL and the therein used Venus commands are shown.
Only the basic Controller functions are supported. More features are given if the Venus commands directly are used to program the controller, or use WP2COMM.DLL function “ExecuteCommand”

P: **Pegasus / Taurus / Orion**
D : **Delta PC**
C : **Corvus, MCX-2, MCX-2eco**

Function	Related Function	P	D	C	Used Venus-[X] commands
InitController		X	X	X	-
OpenController		X	X	X	-
CloseController		X	X	X	-
ResetComm		X	X	X	-
SetDecimalSeparator		X	X	X	
PreprocessReply		X	X	X	
ExecuteCommand		X	X	X	-
GetReply		X	X	X	-
AbortCommand		X	X	X	[Ctrl]+[C]
Calibrate	RangeMeasure		X	X	calibrate, cal
CalibrateA	RangeMeasureA	X			ncalibrate, ncal
ClearParameterStack	GetParamsOnStack		X	X	clear
ClearParameterStackA	GetParamsOnStackA	X			nclear
GetAcceleration	SetAcceleration		X	X	getaccel, ga
GetAccelerationA	SetAccelerationA	X			getnaccel, gna
GetAxisMode	SetAxisMode		X	X	getaxis
GetAxisModeA	SetAxisModeA	X			getnaxis
GetError			X	X	geterror, ge
GetErrorA		X			getnerror, gne
GetJoystickVelocity	SetJoystickVelocity			X	getjoyspeed
GetJoystickVelocityA	SetJoystickVelocityA	X			getnjoyspeed
GetLimits			X	X	getlimit
GetLimitsA		X			getnlimit
GetParamsOnStack	ClearParameterStack			X	gsp
GetParamsOnStackA	ClearParameterStackA	X			gnsp
GetPos			X	X	pos, p
GetPosA		X			npos, np
GetStatus			X	X	status, st
GetStatusA		X			nstatus, nst
GetVelocity	SetVelocity		X	X	getvel, gv
GetVelocityA	SetVelocityA	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			nmove, nm
MoveRelative			X	X	rmove, r
MoveRelativeA		X			nrmove, nr
MoveAbsoluteAutoReply			X	X	move, m
MoveAbsoluteAutoReply A		X			nmove, nm
MoveRelativeAutoReply			X	X	rmove, r
MoveRelativeAutoReplyA		X			nrmove, nr
RangeMeasure			X	X	rangemeasure, rm
RangeMeasureA		X			nrangemeasure, nrm
SetAcceleration	GetAcceleration		X	X	setaccel, sa
SetAccelerationA	GetAccelerationA	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			setnpos
SetVelocity	GetVelocity		X	X	setvel, sv
SetVelocityA	GetVelocityA	X			setnvel, snv

Venus-1 / Venus-2 differences

Venus-2 has been developed on the basis of the interpreter language Venus-1. The fundamental command construction is identical.
The expansion was necessary as the fundamental structures of Venus-1 are designed for controller with at most three linear interpolated axes; but the controller Pegasus supports any number of independent axes (n). Venus-1 commands which structure is designed for operating n-axes are taken over in Venus-2 without any syntax alteration.
The other commands are expanded with the addition "n".
For example: The Venus-1 command ***cal*** which has at the same time an effect on three axes has become the axis specific command ***ncal*** in Venus-2, ***move*** has become ***nmove*** etc.

WP2COMM.DLL functions

Procedures in C/Pascal notation

General:

- Returns values of all functions: 0 if OK, otherwise error code
- LPSTR: Buffer size should be >255 Bytes

InitController

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

functionInitController (ControllerMode: DWORD; Axes: DWORD; ComPort: DWORD; Baudrate: DWORD; UserWin: HWND; AsyncMsg: UINT; Mode: DWORD):DWORD;

Description:
Sets the main parameters, WP2COMM.DLL uses to communicate with the controller. InitController must be called first.

Parameters:

ControllerMode:
Type of controller, MCX or MC2000 = 1, DeltaPC = 2, Alpha = 4, Taurus = 8, Pegasus = 16, Orion = 32, Corvus = 64.
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*.
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

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);
functionOpenController: DWORD;

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

Parameters:
None

CloseController

DWORD CloseController(void);
functionCloseController: DWORD;

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

Parameters:
None

ResetComm

DWORD ResetComm(void);
functionResetComm: DWORD;

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

Parameters:
None

SetDecimalSeparator

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

Description:
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.

Parameters:
DecimalSep: decimal separator

Values:	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);
functionPreprocessReply(Active: BYTE): DWORD;

Description:
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().

Parameters:
Active:

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

ExecuteCommand

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

Description:
Executes any Venus command

Parameters:
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);
functionGetReply(Reply: LPSTR):DWORD;

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

Parameters:
Reply: Buffer to receive the data

AbortCommand

DWORD AbortCommand(void);
functionAbortCommand: DWORD;

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

Parameters: None
See Venus-1 / Venus-2 command: [Ctrl]+[C]

Calibrate

DWORD Calibrate(void);
functionCalibrate: DWORD;

Description:
Calibrates the controller; see RangeMeasure

Parameters:
None
See Venus-1 command: calibrate/cal

CalibrateA

DWORD CalibrateA(DWORD Axis);
functionCalibrateA(Axis:DWORD): DWORD;

Description:
Calibrates the axis; see RangeMeasure

Parameters:
Axis: 1 ... no. of axes
See Venus-2 command: ncalibrate/ncal

ClearParameterStack

DWORD ClearParameterStack(void);
functionClearParameterStack: DWORD;

Description:
Clears the controller stack

Parameters: None
See Venus-1 command: clear

ClearParameterStackA

DWORD ClearParameterStackA(DWORD Axis);
functionClearParameterStackA(Axis: DWORD): DWORD;

Description:
Clears the parameter stack of an axis

Parameters:
Axis: 1 ... no. of axes
See Venus-2 command: nclear

GetAcceleration

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

Description:
Gets the acceleration of all axes

Parameters:
Accel: acceleration in current unit
See Venus-1 command: getaccel/ga

GetAccelerationA

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

Description:
Gets the acceleration of an axis

Parameters:
Accel: acceleration in current unit
Axis: 1 ... no. of axes
See Venus-2 command: getnaccel/gna

GetAxisMode

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

Description:
Gets the mode of the axes

Parameters:
Axis1Mode, Axis2Mode, Axis3Mode, Axis4Mode: values defining the mode of each axis. Can be NULL/nil if the axis doesn't exist.
See Venus-1 command: getaxis

GetAxisModeA

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

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

Parameters:
Mode: 0 ... 2, defining the mode of the axis
Axis: 1 ... no. of axes
See Venus-2 command: getnaxis

GetError

DWORD GetError(LPDWORD Error);
functionGetError(Error: LPDWORD):DWORD;

Description:
Gets the errorcode of the controller

Parameters:
Error: Error code
See Venus-1 command: geterror/ge

GetErrorA

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

Description:
Gets the errorcode of an axis

Parameters:
Error: Venus error code
Axis: 1 ... no. of axes
See Venus-2 command: getnerror/gne

GetJoystickVelocity

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

Description:
Gets the joystick velocity

Parameters:
JoyVel: joystick velocity
See Venus-1 command: getjoyspeed

GetJoystickVelocityA

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

Description:
Gets the joystick velocity of an axis

Parameters:
JoyVel: joystick velocity
Axis: 1 ... no. of axes
See Venus-2 command: getnjoyspeed

GetLimits

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

Description:
Gets the limits of the axes

Parameters:
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-1 command: getlimits

GetLimitsA

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

Description:
Gets the limits of an axis

Parameters:
Min, Max: Values, defining the min/max values of the axis
Axis: 1 ... no. of axes
See Venus-2 command: getnlimits

GetParamsOnStack

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

Description:
Gets the number of parameters on the stack of the controller

Parameters:
Value: Number of parameters on the stack
See Venus-1 command: gsp

GetParamsOnStackA

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

Description:
Gets the number of parameters on the stack of an axis

Parameters:
Value: Number of parameters on the stack
Axis: 1 ... no. of axes
See Venus-2 command: gnsp

GetPos

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

Description:
Gets the position of the axes

Parameters:
Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: coordinates in current units. Can be NULL/nil if the axis doesn't exist.
See Venus-1 command: pos/p

GetPosA

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

Description:
Gets the position of an axis

Parameters:
Pos: coordinate in current units
See Venus-2 command: npos/np

GetStatus

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

Description:
Gets the controller status

Parameters:
Status: Status of the controller
See Venus-1 command: status/st

GetStatusA

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

Description:
Gets the status of an axis

Parameters:
Status: Status of the axis
Axis: 1 ... no. of axes
See Venus-2 command: nstatus/nst

GetVelocity

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

Description:
Gets the velocity of all axes

Parameters:
Vel: velocity in current unit
See Venus-1 command: getvel/gv

GetVelocityA

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

Description:
Gets the velocity of an axis

Parameters:
Vel: velocity in current unit
Axis: 1 ... no. of axes
See Venus-2 command: getnvel/gnv

Identify

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

Description:
Gets the identify string of the controller

Parameters:
Id: identify string
See Venus-1 command: identify

IdentifyA

DWORD IdentifyA(LPSTR Id, DWORD Axis);
functionIdentifyA(Id: LPSTR; Axis: DWORD):DWORD;

Description:
Gets the identify string of an axis

Parameters:
Id: identify string
Axis: 1 ... no. of axes
See Venus-2 command: nidentify

JoystickDisable

DWORD JoystickDisable(void);
functionJoystickDisable: DWORD;

Description:
Turns joystick off

Parameters: None
See Venus-1 command: 0 joystick/0 j

JoystickDisableA

DWORD JoystickDisableA(DWORD Axis);
functionJoystickDisableA(Axis: DWORD): DWORD;

Description:
Turns joystick off

Parameters:
Axis: 1 ... no. of axes
See Venus-2 command: 0 njoystick/0 nj

JoystickEnable

DWORD JoystickEnable(void);
functionJoystickEnable: DWORD;

Description:
Turns joystick on

Parameters: None
See Venus-1 command: 1 joystick/1 j

JoystickEnableA

DWORD JoystickEnableA(DWORD Axis);
functionJoystickEnableA(Axis: DWORD): DWORD;

Description:
Turns joystick on

Parameters:
Axis: 1 ... no. of axes
See Venus-2 command: 1 njoystick/1 nj

MoveAbsolute

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

Description:
Moves to the desired position

Parameters:
Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: coordinates in current units. Can be NULL/nil if the axis doesn't exist.
See Venus-1 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);
functionMoveAbsoluteA(Pos:LPSTR; Axis: DWORD):DWORD;

Description:
Moves to the desired position

Parameters:
Pos: coordinate in current units
Axis: 1 ... no. of axes
See Venus-2 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);
functionMoveAbsoluteAutoReply(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

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

Parameters:
Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: coordinates in current units
See Venus-1 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);
functionMoveAbsoluteAutoReplyA(Pos: LPSTR; Axis: DWORD):DWORD;

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

Parameters:
Pos: coordinate in current units
Axis: 1 ... no. of axes
See Venus-2 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);
functionMoveRelative(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

Description:
Moves relative starting from the current position

Parameters:
Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: coordinates in current units. Can be NULL/nil if the axis doesn't exist.
See Venus-1 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);
functionMoveRelativeA(Pos: LPSTR; Axis: DWORD):DWORD;

Description:
Moves relative starting from the current position

Parameters:
Pos: coordinate in current units
Axis: 1 ... no. of axes
See Venus-2 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);
functionMoveRelativeAutoReply(Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: LPSTR):DWORD;

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

Parameters:
Axis1Pos, Axis2Pos, Axis3Pos, Axis4Pos: coordinates in current units. Can be NULL/nil if the axis doesn't exist.
See Venus-1 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);
functionMoveRelativeAutoReplyA(Pos: LPSTR; Axis: DWORD):DWORD;

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

Parameters:
Pos: coordinate in current units
Axis: 1 ... no. of axes
See Venus-2 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);
functionRangeMeasure: DWORD;

Description:
Measures the stage to determine the limits

Parameters: None
See Venus-1 command: rangemeasure/rm

RangeMeasureA

DWORD RangeMeasureA(DWORD Axis);
functionRangeMeasureA(Axis: DWORD): DWORD;

Description:
Measures the axis to determine the limit

Parameters:
Axis: 1 ... no. of axes
See Venus-2 command: nrangemeasure/nrm

SetAcceleration

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

Description:
Sets the acceleration of all axes

Parameters:
Accel: acceleration in current unit
See Venus-1 command: setaccel/sa

SetAccelerationA

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

Description:
Sets the acceleration of an axis

Parameters:
Accel: acceleration in current unit
Axis: 1 ... no. of axes
See Venus-2 command: setnaccel/sna

SetAxisMode

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

Description:
Sets the mode of the axes

Parameters:
Axis1Mode, Axis2Mode, Axis3Mode, Axis4Mode: 0 ... 2, defining the mode of each axis. Can be NULL/nil if the axis doesn't exist.
See Venus-1 command: setaxis

SetAxisModeA

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

Description:
Sets the mode of the axes

Parameters:
Mode: 0 ... 2, defining the mode of the axis
Axis: 1 ... no. of axes
See Venus-2 command: setnaxis

SetJoystickVelocity

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

Description:
Sets joystick velocity

Parameters:
JoyVel: joystick velocity
See Venus-1 command: joyspeed/js

SetJoystickVelocityA

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

Description:
Sets joystick velocity of an axis

Parameters:
JoyVel: joystick velocity
Axis: 1 ... no. of axes
See Venus-2 command: njoyspeed/njs

SetOrigin

DWORD SetOrigin(void);
functionSetOrigin: DWORD;

Description:
Sets the current position as zero

Parameters: None
See Venus-1 command: setpos

SetOriginA

DWORD SetOriginA(DWORD Axis);
functionSetOriginA(Axis: DWORD): DWORD;

Description:
Sets the current position of an axis as zero

Parameters: None
Axis: 1 ... no. of axes
See Venus-2 command: setnpos

SetVelocity

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

Description:
Set the velocity of all axes

Parameters:
Vel: Velocity in current unit
See Venus-1 command: setvel/sv

SetVelocityA

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

Description:
Set the velocity of all axes

Parameters:

Vel: Velocity in current unit
Axis: 1 ... no. of axes
See Venus-2 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.

- 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 Corvus/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 Corvus these commands are: *0 mode*, *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, *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

Communication with two ITK controllers in one application

See folder “Special”

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