PATHLIB4.DLL: Three Axis Path Control Library
for the *PIC-SERVO* SC and *PIC-SERVO CMC*

*This document assumes that the reader has a basic familiarity with the **PIC-SERVO SC or CMC** Coordinated Motion Controller. Please refer to the **PIC-SERVO SC** or **PIC-SERVO CMC** data sheet for additional information.*

1.0 Overview

PATHLIB4.DLL is a library for creating 2 or 3 axis coordinated motion control applications using the **PIC-SERVO SC** or **PIC-SERVO CMC** motion controllers. The library is designed for controlling Cartesian (X-Y-Z) mechanisms, and allows users to create and execute multi-segment linear and circular arc motions in any plane or combination of planes. It relieves the user of most of the mathematical calculation required for generating complex paths, although a basic understanding of three dimensional geometry is still required.

A complete path for coordinated motion is built up of line segments and arc segments which are added to a segment list. Adjoining segments should be tangent. Functions for adding line segments or circular arc segments to the internal segment list, as well as functions for setting other path parameters, are included in this library. Once the list of segments has been defined, the function AddPathPoints is called to break the segments down into finely spaced path points and download them into the **PIC-SERVO**s' path point buffers for execution. Because most paths will be longer than can be held in the path point buffers, AddPathPoints can be called repeatedly to add more path points as the path executes. AddPathPoints must be called often enough to make sure that more points are added before path point buffers are depleted. While the path is executing, the application program can dynamically alter the feed rate, as well as monitor the path progress using standard **PIC-SERVO** functions found in the library NMCLIB04.DLL.

PATHLIB4.DLL only includes functions for executing coordinated paths. Complete applications will need to call additional **PIC-SERVO** related functions found in NMCLIB04.DLL. Source code for a complete example program, PATHDEMO, is provided in both C (for C++ Builder) and Visual Basic, and can be found at www.jrkerr.com.

PATHLIB4.DLL can be used under Windows 95, Windows 98, Windows NT, Windows XP or Windows 2000 and is intended for use with the standard **PIC-SERVO** library, NMCLIB04.DLL.

2.0 Architecture

The PATHLIB.DLL architecture consists of three parts: (1) basic geometric functions (dot product, cross product, etc.) which are used internally and not exported, (2) functions relating to the creation of the *segment list*, and (3) functions used in the actual execution the path.

Segment List Operations

The segment list functions all deal with path segments described with floating point numbers, typically in units of inches or millimeters. The units used do not actually matter, because the user specifies a conversion factor for each axis in terms of encoder counts per unit. All segment dimensions should then be in terms of the same units.

For example, if the X axis has a 500 line encoder giving 2000 counts per revolution, and it is connected to a lead screw with 10 threads per inch, the X scale factor would be 20000 counts per inch, and all dimensions would be specified in inches. If you prefer working in millimeters, the X scale factor should be specified as 787.4 counts per mm, and all dimensions would be specified in millimeters. (Scale factors are set with the SetPathParams function described further on.)

The segments are also described relative to an origin. The SetOrigin(x, y, z) function is used to specify the location of the origin (in your preferred units) relative to machine's home position. (The home position is the location where the encoder positions all read zero.) For example, a homing routine for the X axis may set the encoder zero position at one end of a 10 inch travel, but you want to use the middle of the travel as your position for specifying motions. In this case, you would set the X origin to 5.0 inches. A move to a position of -1.0 would then put the X axis 4.0 inches from the end of travel.

A path is initialized with a call to the function ClearSegList(x, y, z) which clears the segment list, and sets the starting point (x, y, z) for the path. You must make sure that your machine is actually at this starting point before starting to execute the motion!

Also prior to adding line and arc segments, you must call the function SetTanTolerance(theta). When you add a line or arc segment, ideally it should be exactly tangent to the previous segment for smooth motion. However, you may wish to move along a series of *nearly* tangent segments in a continuous path, event though it will be a little bit lumpy. SetTanTolerance allows you to set the maximum allowable angle between segments for them to be considered tangent. (The functions to add line and arc segments will return an error if you attempt to add a segment which is not tangent to the previous segment.)

With the tangent tolerance set and the starting point specified, you can then begin to add line and arc segments. The function AddLineSeg(x, y, z) simply specifies the final end point of a line segment to be added to the segment list. x, y and z are relative to the specified origin. The starting point for the segment will automatically be set to the end point of the previous segment. This function will return an error if the line segment is not tangent (within the specified tolerance)

to the previous segment, or if the segment list is full. The segment list can hold up to 1000 segments to accommodate very complex paths.

The function AddArcSeg(x, y, z, cx, cy, cz, nx, ny, nz) is used to add an arc segment to the segment list. x, y, and z specify end point of the segment, and cx, cy and cz specify the center point of the arc. Both the points (x, y, z) and (cx, cy, cz) are specified relative to the origin. The parameters nx, ny and nz are used to specify a vector normal (perpendicular) to the plane containing the arc. Whether this vector points in the positive direction or the negative direction will determine whether the arc is clockwise or counterclockwise.

For example, if we are starting at a point x=1.0, y=0.0, z=0.0 and we wish to move to the point x=2.0, y=1.0, z=0.0, rotating in a clockwise direction about the center point cx = 2.0, cy = 0.0, cz = 0.0 (an arc of 90 degrees) , we would use a normal vector of nx = 0.0, ny = 0.0, nz = -1.0 . This is essentially a vector perpendicular to the X-Y plane and pointing in the -Z direction. If we wanted to move to the same end point and about the same center point but in the counterclockwise direction (to give us an arc of 270 degrees), the normal vector would be nx = 0.0, ny = 0.0, nz = +1.0. Complete circles can be added by specifying an end point which is same as the starting point. The table below shows the normal vectors to use for clockwise and counterclockwise arcs lying in the X-Y, Y-Z or Z-X planes.

Type of Arc	Normal vector nx, ny, nz
X-Y plane, clockwise	0.0, 0.0, -1.0
X-Y plane, counterclockwise	0.0, 0.0, 1.0
Y-Z plane, clockwise	-1.0, 0.0, 0.0
Y-Z plane, counterclockwise	1.0, 0.0, 0.0
Z-X plane, clockwise	0.0, -1.0, 0.0
Z-X plane, counterclockwise	0.0, 1.0, 0.0

In general arc segments can lie in planes at any angle, although determining the arc parameters is somewhat more complicated.

The AddArcSeg function will return an error if the arc is not tangent to the previous segment, the data specified does not form a valid arc, or if the segment list is full.

Path Execution Operations
Initialization
The first step in path execution is to initialize a host of variables relating to the execution of the path using the function SetPathParams. This one function is used to set the addresses of the motor controller axes, the axis scale factors (counts per unit as described above), the acceleration/deceleration value, the frequency of the path points (30, 60 or 120 Hz) and the size of the path point buffer that you want to use.

The addresses include the individual module addresses used for the X, Y and Z axes, as well as the group address assigned for all three axes, and the individual address of the module assigned to be the group leader. For applications where there are no other NMC modules besides the three motor controllers, the group addresses can be left at the default of 0xFF, and it is not necessary to make any of the modules a group leader. (A leader address of 0 can be used if there is no leader.) For systems which only use 2 axes, the X and Y axes must be used, and the address for the Z axis should be set to zero.

The acceleration value is used for the acceleration at the beginning of the path and the deceleration at the end of the path. It is specified in units per second per second. For example, if we wanted to reach a maximum feed rate of 2.0 inches per second in 0.5 seconds, the acceleration would be 4.0 inches/sec/sec.

The path point frequency determines how finely each segment will be broken up into individual path points (with straight line segments in-between). Defined constants P_30HZ, P_60HZ or P_120HZ may be used, corresponding to 30, 60 or 120 Hz. A 60 Hz path will typically be more than accurate enough for most applications. A 30 Hz path, however, will require less computation and communication, and the path point buffer will last twice as long as for a 60 Hz path. A 120 Hz path should only be needed for controlling very high performance mechanisms.

Lastly, the path point buffer size specifies how much data should be loaded into the path point buffer at one time. A maximum of 87 can be used which will last about 3 seconds at 30 Hz or 1.5 seconds at 60 Hz. A longer path point buffer size means that as the path executes, the function AddPathPoints does not have to be called as often to refill the buffer. However, a longer path point buffer will introduce longer delays if you decide to change the feed rate, because the path points already loaded into the buffers will continue executing at the previous feed rate.

One last thing the SetPathParams function does is to make sure that you have set the default status packet for each axis to include the position data, the position error data, the number points left in the path point buffer, and the auxiliary status byte. (You must call the function NmcDefineStatus(addr, SEND_POS | SEND_PERROR | SEND_NPOINTS) for each axis.) You may wish to have additional data included, but this data should be specified at a minimum.

Typically, the function SetPathParams can be called just once at the beginning of your program, although the parameters can also be changed between (*but never during*) motions.

Also prior to beginning the execution of your path, you should set the feed rate with the function SetFeedrate(fr). The feed rate fr is given in units per second. This will be the speed that the business end of your machine travels along the path, except during the acceleration and deceleration phases.

Finally, just before starting the path execution, the function InitPath should be called to indicate that the segment list is complete, and that your mechanism is at the starting point you specified with the ClearSegList command.

<u>Path Execution</u>
Execution of the path is started by a call to the function AddPathPoints. This function starts at
the beginning of your segment list and breaks it down into individual path points spaced at either
30, 60 or 120 Hz. The individual path points will form a smooth path with an acceleration ramp
at the beginning of the path and a deceleration ramp at the end.

AddPathPoints is recalled repeatedly until the entire path is executed. When AddPathPoints is
first called, it will fill up the path point buffers and begin execution of the path. If AddPathPoints
has not downloaded the all of the path points for the entire segment list, it will return a value of 0.
If this is the case, AddPathPoints should be called again before the path point buffers are emptied.
For example, if you specify a path frequency of 30 Hz and a path point buffer size of 45 points,
you should make sure to call AddPathPoints at least every 1.5 seconds. If you do not, the motion
will jerk to a stop and then jerk to a start again as you add more points.

When AddPathPoints has downloaded the last of the points for the path, it will return a value of
-1. Subsequent calls to AddPathPoints will continue to return a value of -1 until a new path is
initialized.

Even after AddPathPoints returns a value of -1, the portion of the path still loaded into the path
point buffers will continue to execute until the buffers have been emptied. You should monitor
the PATH_MODE bit of the auxiliary status byte to determine when the tail end of the path
motion is complete.

Calls to AddPathPoints will automatically update the module data for the X, Y and Z axis
controllers as long as AddPathPoints is still actively downloading path points and returning a
value of 0. In this case, your application does not need to do further polling of the controllers to
retrieve the current status data. In general, it is good to avoid unnecessary polling of the NMC
controllers to ensure maximum performance. If, however, AddPathPoints returns a value of -1,
your application will have to actively poll the modules (using NmcNoOp or NmcReadStatus) to
get the current module data.

During path execution, the function SetFeedrate can be used dynamically change the feed rate,
and the new feed rate will take effect for the section of the path yet to be downloaded into the
path point buffers. Setting a feed rate of exactly 0.0 will put the path in a "feed hold" condition,
decelerating the machine to a stop along the path, and then pausing until the feed rate is set to a
positive value again.

3.0 Function Definitions

Segment List Functions

void SetOrigin(float xoffset, float yoffset, float zoffset)
Sets the origin, defined relative to the home position. (The home position is where all encoder
counters read zero.) Coordinate data for AddLineSeg or AddArcSeg commands will be relative

to this origin. xoffset, yoffset and zoffset should be in whatever units you choose to use for the scale parameters set with SetPathParams.

`void SetTangentTolerance(float theta)`
Sets the angle tolerance, theta (in degrees), between two consecutive segments which are considered to be tangent. Values between 3 and 10 degrees are typical.

`void ClearSegList(float x, float y, float z)`
Clears the internal segment list and sets the starting point for a new set of segments. The new starting point (x, y, z) should be in whatever units you choose to use for the scale parameters set with SetPathParams.

`int AddLineSeg(float x, float y, float z)`
Adds a line segment to the internal segment list. Coordinates (x, y, z) are the endpoint of the line segment and should be in whatever units you choose to use for the scale parameters set with SetPathParams.

Returns the index of the segment in the segment list on success, -1 if the line segment is not tangent to the previous segment, or -2 if the segment list is full. If the return value is less than 0, the segment will not be added to the segment list.

`int AddArcSeg( float x,  float y,  float z,`
`            float cx, float cy, float cz,`
`            float nx, float ny, float nz )`
Adds an arc segment to the internal segment list. Coordinates (x, y, z) are the endpoint of the arc, coordinates (cx, cy, cz) are the center point of the arc, and coordinates (nx, ny, nz) form a vector normal to the plane of the arc. The direction of the normal vector (positive or negative) dictates the direction of the arc, moving from the start point to the endpoint in a right-handed sense. The end point and center point coordinates should be relative to the origin, and in whatever units you choose to use for the scale parameters set with SetPathParams.   To add a complete circle, you should make the endpoint coordinates of the arc equal to the endpoint coordinates of the previous path segment.

Returns the index of the segment in the segment list on success, -1 if the line segment is not tangent to the previous segment, -2 if the segment list is full, or -3 if the arc data is invalid. If the return value is less than 0, the segment will not be added to the segment list.

J E F F R E Y   K E R R ,   L L C • *www.jrkerr.com*

Path Execution Functions

```
int SetPathParams(int freq, int nbuf,
                  int xaxis, int yaxis, int zaxis,
                  int groupaddr, int leaderaddr,
                  float xscale, float yscale, float zscale,
                  float accel )
```

*--- For **PIC-SERVO CMC** only - use SetPathParams2() for **PIC-SERVO SC** ---*

Sets variables relating to the execution of the path defined by the internal segment list, and verifies that the proper status data for each module has been specified.. The following parameters are set:

freq	use defined constants P_30HZ, P_60HZ or P_120HZ
nbuf	number of points to store in the **PIC-SERVO's** path point buffer (maximum of 87)
xaxis, yaxis, zaxis	Individual module addresses for the X, Y and Z axes (set the zaxis to 0 if only X and Y are used)
groupaddr	Group address for the X, Y and Z axis modules
leaderaddr	Individual address for whichever module (X, Y or Z) has been defined as the group leader. (Use 0 if there is no group leader.)
xscale, yscale, zscale	Scale factor, in encoder counts per unit, for the X, Y and Z axes. The units can be anything (typically inches or millimeters) as long as all coordinates are specified in the same units.
accel	Value used for acceleration and deceleration along the path in units/second/second.

Returns 0 on success, or -1 if the status data for each module does not at least include the motor position, position error, auxiliary status byte, and number of path points.

```
int SetPathParams2(int freq, int nbuf,
                   int xaxis, int yaxis, int zaxis,
                   int groupaddr, int leaderaddr,
                   float xscale, float yscale, float zscale,
                   float accel )
```

*--- May be used with either **PIC-SERVO SC** or **PIC-SERVO CMC** ---*

Sets variables relating to the execution of the path defined by the internal segment list, and verifies that the proper status data for each module has been specified.. The following parameters are set:

freq	use defined constants P_30HZ, P_60HZ or P_120HZ
nbuf	number of points to store in the **PIC-SERVO's** path point buffer (maximum of 87)
xaxis, yaxis, zaxis	Individual module addresses for the X, Y and Z axes (set the zaxis to 0 if only X and Y are used)
groupaddr	Group address for the X, Y and Z axis modules

J E F F R E Y   K E R R ,   L L C • *www.jrkerr.com*

leaderaddr	Individual address for whichever module (X, Y or Z) has been defined as the group leader. (Use 0 if there is no group leader.)
xscale, yscale, zscale	Scale factor, in encoder counts per unit, for the X, Y and Z axes. The units can be anything (typically inches or millimeters) as long as all coordinates are specified in the same units.
accel	Value used for acceleration and deceleration along the path in units/second/second.

Returns 0 on success, or -1 if the status data for each module does not at least include the motor position, position error, auxiliary status byte, and number of path points. If using the **PIC-SERVO SC** with the 3-Phase commutation or Antiphase PWM output modes, or if you have limit switch protection enabled, you must call ServoSetIoCtrl() (in NMCLIB04.DLL) prior to calling SetPathParams2().

float InitPath()
Initializes the path before execution, and declares that all axes are at the starting point of the path. Returns the overall path length on success, or 0.0 on a communications error.

int AddPathPoints()
Adds path points to the **PIC-SERVO CMC**s' path point buffers, and begins path execution automatically after nbuf points have been added. If the path is longer that can be held in the buffers, additional calls to AddPathPoints should be made frequently enough to make sure the buffers are never emptied prematurely. Returns 0 if the path is not complete (ie, more points need to be added) or -1 if all path points for the current segment list have been downloaded. Returns -2 on a communications error.

void SetFeedrate(float fr)
Sets the feed rate, fr (in units/second) for the path execution. This is essentially the velocity of the tool along the specified path. Setting a feed rate of 0.0 will decelerate along the path to a stop, and the position will be held, even with subsequent calls to AddPathPoints, until the feedrate is returned to a positive value.

Example Software

Two example programs are available which demonstrate the use of PATHLIB4.DLL. PATHDEMO is a simple program which executes a path of line and arc segments for a 2 axis mechanism. PSCNC is a 3-axis milling machine controller which includes a simple G-Code interpreter. Source code for both of these programs is available in C, written for use with Borland's C++ Builder. Source code for PATHDEMO is also provided in Visual Basic.