```
//
//	Test.cpp		sample console application that uses USMCDLL.dll
//

#pragma warning(disable : 4996) // Disable warnings about some functions in VS 2005
#define WIN32_LEAN_AND_MEAN			// Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>
#include <conio.h>
#include <process.h>
#include "USMCDLL.h"

USMC_Devices DVS;
DWORD Dev;

// Function that prints information about connected devices to console
void PrintDevices(USMC_Devices &DVS)
{
	for(DWORD i = 0; i < DVS.NOD; i++)
	{
		printf("Device - %d,\tSerial Number - %.16s,\tVersion -
%.4s\n",i+1,DVS.Serial[i],DVS.Version[i]);
	}
}

// Function that prints information about device state to console
```

```
void PrintDState(USMC_State &State)

{

        printf( "The state is:\n" );

        printf( "- Current Position in microsteps - %d\n", State.CurPos );

        printf( "- Temperature - %.2f\xf8\x43\n", State.Temp );

        printf( "- Step Divisor - %d\n", State.SDivisor);

        printf( "- Loft State - %s\n", State.Loft?"Indefinite":"Fixed" );

        printf( "- Power - %s\n", State.Power?(State.FullPower?"Full":"Half"):"Off" );

        if(State.RUN)

                printf( "- Step Motor is Running in %s Direction %s\n",

                                State.CW_CCW?"CCW":"CW", ((State.SDivisor==1) &&
State.FullSpeed)?"at Full Speed":"" );

        else

                printf( "- Step Motor is Not Running\n" );

        printf( "- Device %s\n", State.AReset?"is After Reset":"Position Already Set" );

        printf( "- Input Synchronization Logical Pin State - %s\n", State.SyncIN?"TRUE":"FALSE" );

        printf( "- Output Synchronization Logical Pin State - %s\n", State.SyncOUT?"TRUE":"FALSE" );

        printf( "- Rotary Transducer Logical Pin State - %s\n", State.RotTr?"TRUE":"FALSE" );

        printf( "- Rotary Transducer Error Flag - %s\n", State.RotTrErr?"Error":"Clear" );

        printf( "- Emergency Disable Button - %s\n", State.EmReset?"Pushed":"Unpushed" );

        printf( "- Trailer 1 Press State - %s\n", State.Trailer1?"Pushed":"Unpushed" );

        printf( "- Trailer 2 Press State - %s\n", State.Trailer2?"Pushed":"Unpushed" );

        if( State.Voltage == 0.0f )

                printf( "- Input Voltage - Low\n", State.Voltage);

        else

                printf( "- Input Voltage - %.1fV\n", State.Voltage);
}

// Function that scans start parameters
void ScanDStartParameters(int &DPos, float &Speed, USMC_StartParameters &SP )
{
	char str[105];

	// Defaults
	Speed = 2000.0f;
	DPos = 0;
	SP.SDivisor = 8;

	printf( "Destination position:" );
	gets(str);
	sscanf(str,"%d", &DPos );
	printf( "\nSpeed (in tacts):" );
	gets(str);
	sscanf(str,"%f", &Speed );
	printf( "\nSteps Divisor:" );
	gets(str);
	sscanf(str,"%d", &(SP.SDivisor) );
}

// Function that prints information about device start parameters to console
```

```
void PrintDStartParameters(int DPos, float Speed, const USMC_StartParameters &SP )

{

	printf( "Destination position - %d\n", DPos);

	printf( "Speed - %.2ftacts/s\n", Speed );

	printf( "Steps Divisor - %d\n", SP.SDivisor );

	if(SP.SDivisor == 1)

	{

		printf( "Slow start/stop mode - %s\n", SP.SlStart?"Enabled":"Disabled" );

	}else if(SP.LoftEn)

	{

		printf( "Automatic backlash operation - Enabled\n" );

		printf( "Automatic backlash operation direction - %s\n", SP.DefDir?"CCW":"CW" );

		printf( "Force automatic backlash operation - %s\n", SP.ForceLoft?"TRUE":"FALSE" );

	}else{

		printf( "Automatic backlash operation - Disabled\n" );

	}

	if(SP.WSyncIN)

		printf( "Controller will wait for input synchronization signal to start\n" );

	else

		printf( "Input synchronization signal ignored \n" );

	printf( "Output synchronization counter will %sbe reset\n", SP.SyncOUTR?"":"not " );

}


// Function that prints information about device parameters to console

void PrintDParameters(USMC_Parameters &Parameters)
```

```
{
	printf( "The parameters are:\n" );
	printf( "Full acceleration time - %.0f ms\n", (double) Parameters.AccelT );
	printf( "Full deceleration time - %.0f ms\n", (double) Parameters.DecelT );
	printf( "Power reduction timeout - %.0f ms\n", (double) Parameters.PTimeout );
	printf( "Button speedup timeout 1 - %.0f ms\n", (double) Parameters.BTimeout1 );
	printf( "Button speed after timeout 1 - %.2f steps/s\n", (double) Parameters.BTO1P );
	printf( "Button speedup timeout 2 - %.0f ms\n", (double) Parameters.BTimeout2 );
	printf( "Button speed after timeout 2 - %.2f steps/s\n", (double) Parameters.BTO2P );
	printf( "Button speedup timeout 3 - %.0f ms\n", (double) Parameters.BTimeout3 );
	printf( "Button speed after timeout 3 - %.2f steps/s\n", (double) Parameters.BTO3P );
	printf( "Button speedup timeout 4 - %.0f ms\n", (double) Parameters.BTimeout4 );
	printf( "Button speed after timeout 4 - %.2f steps/s\n", (double) Parameters.BTO4P );
	printf( "Button reset timeout - %.0f ms\n", (double) Parameters.BTimeoutR );
	printf( "Button reset operation speed - %.2f steps/s\n", (double) Parameters.MinP );
	printf( "Backlash operation distance - %d steps\n", (int)Parameters.MaxLoft );
	printf( "Revolution distance - %d steps\n", (int)Parameters.RTDelta );
	printf( "Minimal revolution distance error - %d steps\n", (int)Parameters.RTMinError );
	printf( "Power off temperature - %.2f\xf8\x43\n", (double)Parameters.MaxTemp );
	printf( "Duration of the output synchronization pulse - ");
	if(Parameters.SynOUTP == 0)
		printf( "minimal\n");
	else
		printf( "%.1f * [Tact Period]\n", Parameters.SynOUTP - 0.5);
	printf( "Speed of the last phase of the backlash operation - ");
	if(Parameters.LoftPeriod == 0.0f)

		printf( "normal\n" );

	else

		printf( "%.2f steps/s\n", (double)Parameters.LoftPeriod );

	printf( "<Angular Encoder Step> Equals <Angular Step Motor Step>/<%.2f>\n",
Parameters.EncMult);


}


// Function that prints information about device "mode" parameters to console

void PrintDMode(USMC_Mode &Mode)

{

	printf( "Mode parameters:\n" );

	printf( "Buttons - ");

	if(Mode.PMode)

	{

		printf( "Disabled\n" );

	}else{

		printf( "Enabled\nButton 1 TRUE state - %s\n", Mode.Butt1T?"+3/+5 V":"0 V(GND)" );

		printf( "Button 2 TRUE state - %s\n", Mode.Butt2T?"+3/+5 V":"0 V(GND)" );

	}

	printf( "Current reduction regime - %s\n", Mode.PReg?"Used":"Not Used" );


	if(Mode.ResetD)

		printf( "Power - %s\n", Mode.EMReset?"Emerjency Off":"Off" );

	else
```

```
        printf( "Power - On\n" );
if(Mode.Tr1En || Mode.Tr2En)
        printf( "Trailers are - %s\n", Mode.TrSwap?"Swapped":"Direct" );


printf( "Trailer 1 - ");
if(Mode.Tr1En)
        printf( "Enabled\nTrailer 1 TRUE state - %s\n", Mode.Tr1T?"+3/+5 V":"0 V(GND)" );
else
        printf( "Disabled\n");
printf( "Trailer 2 - ");
if(Mode.Tr2En)
        printf( "Enabled\nTrailer 2 TRUE state - %s\n", Mode.Tr2T?"+3/+5 V":"0 V(GND)" );
else
        printf( "Disabled\n");


if(Mode.EncoderEn)
{
        printf( "Encoder - Enabled\n");
        printf( "Encoder Position Counter is %s\n", Mode.EncoderInv?"Inverted":"Direct");
        printf( "Rotary Transducer and Input Syncronisation are\n"
                        " Disabled Because of Encoder\n");
}else{
        printf( "Encoder - Disabled\n");
        printf( "Rotary Transducer - ");
        if(Mode.RotTeEn)
```

```
		{
			printf( "Enabled\nRotary Transducer TRUE state - %s\n", Mode.RotTrT?"+3/+5 
V":"0 V(GND)" );
			printf( "Rotary Transducer Operation - %s\n", Mode.RotTrOp?"Stop on 
error":"Check and ignore error" );
			printf( "Reset Rotary Transducer Check Positions - %s\n", 
Mode.ResetRT?"Initiated":"No, why?");
		}else{
			printf("Disabled\n");
		}
		printf("Synchronization input mode:\n");
		if(Mode.SyncINOp)
			printf("Step motor will move one time to the destination position\n");
		else
			printf("Step motor will move multiple times by [destination position]\n");


	}
	printf( "Output Syncronization - ");
	if(Mode.SyncOUTEn)
	{
		printf( "Enabled\nReset Output Synchronization Counter - %s\n", 
Mode.SyncOUTR?"Initiated":"No, why?" );
		printf( "Number of steps after which synchronization output sygnal occures - %u\n", 
Mode.SyncCount );
	}else{
		printf("Disabled\n");
	}
	printf("Synchronization Output is ");

	if(Mode.SyncInvert)

		printf("INVERTED\n");

	else

		printf("NORMAL\n");

}


void PrintEnState(USMC_EncoderState EnState, USMC_Parameters up)

{

	printf( "The encoder state is:\n" );

	printf( "- Current Position in microsteps - %.2f\n", EnState.ECurPos/up.EncMult );

	printf( "- Encoder Position in microsteps - %.2f\n\n", EnState.EncoderPos/up.EncMult );

	printf( "- Current Position in \"Half of Encoder Step\"s - %d\n", EnState.ECurPos );

	printf( "- Encoder Position in \"Half of Encoder Step\"s - %d\n", EnState.EncoderPos );

}


// Function that prints last error information

void PrintError(void)

{

	char er[101];

	USMC_GetLastErr(er,100);

	er[100] = '\0';

	printf("\n%s",er);

}
```

```
USMC_State State;

USMC_StartParameters StPrms;

USMC_Parameters Prms;

USMC_Mode Mode;

USMC_EncoderState EnState;


// Reference to Functions For Every Command (defined at the end of the file)

BOOL F1_Get_Device_State(void);

BOOL F2_START(void);

BOOL F3_STOP(void);

BOOL F4_Set_Parameters(void);

BOOL F5_Set_Mode(void);

BOOL F6_Set_Current_Position(void);

BOOL F7_Turn_Off_and_Save_Current_Position_to_Flash(void);

BOOL FS7_Revert_Start_Position_to_0(void);

BOOL F8_Get_Encoder_State(void);

BOOL FP_Change_Power(void);


BOOL SMPower = FALSE;


int Menu(void)

{

        BOOL Err = FALSE;
```

```
printf("Menu:\n"

        "1 - Get Device State\n"

        "2 - START\n"

        "3 - STOP\n"

        "4 - Set Parameters (can be changed only in c++ code)\n"

        "5 - Set Mode (can be changed only in c++ code)\n"

        "6 - Set Current Position (can be changed only in c++ code)\n"

        "7 - Turn Off and Save Current Position to Flash\n"

        "SHIFT + 7 - Revert Start Position to 0\n"

        "8 - Get Encoder State\n"

        "\n"

        "p - Turn %s Power"

        "\n"

        "9 - Select other device\n"

        "0 - Exit\n"

        "Choose:", SMPower?"Off":"On");

char c = _getch();

switch(c)

{

case '1':

      Err = F1_Get_Device_State();

      break;

case '2':

      Err = F2_START();

      break;
```

```
case '3':
            Err = F3_STOP();
            break;
case '4':
            Err = F4_Set_Parameters();
            break;
case '5':
            Err = F5_Set_Mode();
            break;
case '6':
            Err = F6_Set_Current_Position();
            break;
case '7':
            Err = F7_Turn_Off_and_Save_Current_Position_to_Flash();
            break;
case '&':             // 'SHIFT 7' on my keyboard
            Err = FS7_Revert_Start_Position_to_0();
            break;
case '8':
            Err = F8_Get_Encoder_State();
            break;
case 'p':
case 'P':
            Err = FP_Change_Power();
            break;
```

```
        case '9':
                return 1;
        case '0':
                printf("\n");
                return 0;
        default:
                system("cls");
    }

        if(Err){
                PrintError();
                printf("\nPerforming Refressh...");
                _getch();
                if( USMC_Init( DVS ) )
                {
                        PrintError();
                        return 0;
                }
                return 1;
        }
        return 2;
}

int SelectMenu(void)
```

```
{
	char str[105] = { 100 };
	int ret;

	do{
		Dev = -1;
		system("cls");
		PrintDevices( DVS );
		printf("\n\t0\tExit\n");
		printf("\tx\tReInitialize\n");
		printf("\txx\tClose Driver Window and Exit\n");

		printf("Select:\n");
		gets(str);
		if(strcmp(str,"x")==0)
		{
			//// Perform "Refresh" of Driver
			if( USMC_Init( DVS ) )
			{
				PrintError();
				return 0;
			}
			continue;
		}else if(strcmp(str,"xx")==0)
		{
```

```
                //// Close the MicroSMC.exe process

                if( USMC_Close( ) )

                {

                        PrintError();

                        return 0;

                }

                return 0;

        }


        sscanf(str, "%u", &Dev);

        if(Dev == 0)

                return 0;

}while(Dev>DVS.NOD);

Dev--;

system("cls");


if(USMC_RestoreCurPos(Dev))

        PrintError();

else

        printf("Last saved position is restored. See GetState.\n");


do{

        ret = Menu();

}while( ret == 2 );

return ret;
```

```
}


int Exit(void)

{

        printf("\nPress any key to exit");

        _getch();

        return 0;

}


int _tmain(int argc, _TCHAR* argv[])

{

        system("cls");

        if( USMC_Init( DVS ) )

        {

                PrintError();

                return Exit();

        }


        while(SelectMenu());


        return Exit();

}
```

```
BOOL F1_Get_Device_State(void)

{

        if( USMC_GetState(Dev, State) )

                return TRUE;


        system("cls");

        PrintDState(State);

        printf("\n");

        return FALSE;

}


BOOL F2_START(void)

{

        float Speed = 2000.0f;

        int DestPos = 0;

        if( USMC_GetStartParameters(Dev, StPrms) )

                return TRUE;

        system("cls");

        ScanDStartParameters(DestPos, Speed, StPrms);

        StPrms.SlStart = TRUE;

        if( USMC_Start(Dev, DestPos, Speed, StPrms) )

                return TRUE;

        system("cls");

        PrintDStartParameters(DestPos, Speed, StPrms);

        printf("\n");
        return FALSE;
}

BOOL F3_STOP(void)
{
        if( USMC_Stop(Dev) )
                return TRUE;
        system("cls");
        return FALSE;
}

BOOL F4_Set_Parameters(void)
{
        if( USMC_GetParameters(Dev, Prms) )
                return TRUE;

        // Set anything you want here
        Prms.MaxTemp = 70.0f;
        Prms.AccelT = 200.0f;
        Prms.DecelT = 200.0f;
        Prms.BTimeout1 = 500.0f;
        Prms.BTimeout2 = 500.0f;
        Prms.BTimeout3 = 500.0f;
        Prms.BTimeout4 = 500.0f;
        Prms.BTO1P = 100.0f;
        Prms.BTO2P = 200.0f;

        Prms.BTO3P = 300.0f;

        Prms.BTO4P = 600.0f;

        Prms.MinP = 500.0f;

        Prms.BTimeoutR = 500.0f;

        Prms.LoftPeriod = 500.0f;

        Prms.RTDelta = 200;

        Prms.RTMinError = 15;

        Prms.EncMult = 2.5f;

        Prms.MaxLoft = 32;

        Prms.PTimeout = 100.0f;

        Prms.SynOUTP = 1;

        //


        if( USMC_SetParameters( Dev, Prms ) )

                return TRUE;


        if( USMC_SaveParametersToFlash( Dev ) )

                return TRUE;


        system("cls");

        PrintDParameters( Prms );

        printf("\nThese Parameters are Saved to Flash");

        printf("\nPress any key to exit");

        _getch();
```

```
	system("cls");
	return FALSE;
}

BOOL F5_Set_Mode(void)
{
	if( USMC_GetMode(Dev, Mode) )
		return TRUE;

	// Set anything you want here
	Mode.SyncInvert = !Mode.SyncInvert;
	Mode.EncoderEn = TRUE;
	Mode.RotTrOp = FALSE;
	Mode.ResetRT = TRUE;
	// Set anything you want here

	if( USMC_SetMode(Dev, Mode) )
		return TRUE;
	system("cls");
	PrintDMode(Mode);
	printf("\nPress any key to exit");
	_getch();
	system("cls");
	return FALSE;
}
```

```
BOOL F6_Set_Current_Position(void)

{

	int newCurPos = 1000;


	if( USMC_SetCurrentPosition(Dev, newCurPos) )

		return TRUE;

	system("cls");

	printf("\nSetCurrentPosition executed with argument %d",newCurPos);

	printf("\nPress any key to exit");

	_getch();

	system("cls");

	return FALSE;

}


BOOL F7_Turn_Off_and_Save_Current_Position_to_Flash(void)

{

	// Initialize structures (in case this function runs first)

	if( USMC_GetParameters(Dev, Prms) )

		return TRUE;

	if( USMC_GetMode(Dev, Mode) )

		return TRUE;

	// Go to Full Step (automaticaly), then Turn Off

	Mode.ResetD = TRUE;

	if( USMC_SetMode(Dev, Mode) )
```

```
			return TRUE;
		// Wait until Previous Comand is Done
		do{
			Sleep(50);
			if( USMC_GetState(Dev, State) )
				return TRUE;
		}while(State.Power == TRUE);
		// Remember CurPos in Parameters Only While State.Power - FALSE
		Prms.StartPos = State.CurPos;
		if( USMC_SetParameters( Dev, Prms ) )
			return TRUE;
		// Then of Course You Need to SaveToFlash
		if( USMC_SaveParametersToFlash( Dev ) )
			return TRUE;
		// Now You Can Unplug ME
		system("cls");
		printf("\nThe Position \"%d\" is Saved to Flash\n",Prms.StartPos);
		printf("\nWhen Controller is Powered Up Next Time\n");
		printf("\nIt Will Start From This Position\n");
		printf("\nPress any key to exit");
		_getch();
		system("cls");
		return FALSE;
	}
```

```
BOOL FS7_Revert_Start_Position_to_0(void)

{

	// Initialize structures (in case this function runs first)

	if( USMC_GetParameters(Dev, Prms) )

		return TRUE;

	Prms.StartPos = 0;

	if( USMC_SetParameters( Dev, Prms ) )

		return TRUE;

	// Then of Course You Need to SaveToFlash

	if( USMC_SaveParametersToFlash( Dev ) )

		return TRUE;

	system("cls");

	printf("\nStart Position is Reset to 0\n");

	printf("\nPress any key to exit");

	return FALSE;

}


BOOL F8_Get_Encoder_State(void)

{

	if( USMC_GetEncoderState(Dev, EnState) )

		return TRUE;

	if( USMC_GetParameters(Dev, Prms) )

		return TRUE;

	system("cls");

	PrintEnState(EnState, Prms);
```

```
        printf("\n");

        return FALSE;

}


BOOL FP_Change_Power(void)

{

        if( USMC_GetMode(Dev, Mode) )

                return TRUE;


        SMPower = !SMPower;

        Mode.ResetD = !SMPower;


        if( USMC_SetMode(Dev, Mode) )

                return TRUE;

        system("cls");

        printf("Now, Power is %s\n", Mode.ResetD?"Off":"On");

        return FALSE;

}
```