

Bee Dynamic Link Library (DLL) (Beta release Version 1.2)

(Provided by www.pc-control.co.uk)

The Bee library is a set of functions that provide easy programmer access to all of the facilities available on the complete “BEE” range of boards available from PC Control Ltd. They are contained within a DLL (Dynamic link library) that has a standard interface, which can be accessed from most of the popular languages that support DLL calls (such as Visual Basic, C, C++ etc...). The library is divided into distinct sections corresponding to each of the specific boards and a general section on initialisation. If you only have the one board then you only need concern yourself with the section devoted to that board and the general section. However, the real power of this library is that it enables a complete mix of board types and multiples of each to be used together on the same PC. This enables the designer to construct his custom control system on a modular basis, adding input, output, motor control etc.. etc... as needed, without having to adopt multiple libraries or duplicate code.

The descriptions below look at each of the functions available and examine the function prototype details. Before any of these functions can be used the header file associated with the library should be included in the source code of the program you are going to be using them in. This file is called <be.h> and contains the type declarations and some number definitions that make your programming more intuitive. For more general information on how to call functions in a DLL from within a program (C or VBasic for example) see your compiler manual or the manual provided with the “Bee” board you are using. These manuals can be downloaded from the pc-control website.

General Functions

int InitBee()

Parameters

No parameters

Return

Returns an integer corresponding to the total number of “Bee” boards found.

Description

InitBee() performs general initialisation of the DLL and the Bee environment. It also scans all of your attached USB devices in order to find any of the Bee range of boards. When it finds a “Bee” board it will prepare it for subsequent function calls specific to that board. It is important that this function be called before any specific board functions are used. It is also important to call this function after any changes to the number or location of connected boards.

If, for example, InitBee() returned the number 5, it means that 5 boards are currently connected. Each board within this 5 may then be uniquely identified according to its order within the 5. i.e. a number in the range 1 to 5. This number is called the Device Number” and it is the number used by all subsequent function calls described below.

int GetBoardType(int devnum)

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

Returns

An integer corresponding to a unique number identifying each type of board

Description

Once InitBee() has been called, we know how many boards are connected but not what type they are. This is the function of GetBoardType(). It is called using the “Device Number” as a parameter. The number returned by GetBoardType() corresponds to a unique product identifier number stored on each board and may be compared to a list of product numbers provided in the header file <be.h>. For example, if a WASP was the third device in the connected list then the call....

GetBoardType(3) would return the value 87. For readability some definitions have been included with the <be.h> header file that your program can use. The one for the WASP is BRD_WASP. A typical use of this is shown below...

```
int type
```

```
type = GetBoardType(3);
if(type == BRD_WASP)
{
    ..... specific code for wasp ...
}
```

MiniBee Functions

int MB_SetOutputs(int devnum, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to set the on/off state of MiniBee’s switching outputs. Bits 0 to 13 in the outputs parameter correspond to the desired on/off state of the outputs. A ‘1’ corresponds to “ON” and 0 for “OFF”.

For example the following call (which assumes a MiniBee is device 1 in the list) will set outputs 2,4 and 5 ON with all others OFF.

```
MB_SetOutputs(1, 26)
```

The number ‘26’ is actually 0000 0000 0001 1010 in binary, which should make this clear.

WASP Functions

int WSP_ReadInputs(int devnum, int *analogue)

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

***analogue** The address (or reference to) an array of 4 integers to be used to hold the measured analogue input values.

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Calling this function will result in an analogue to digital conversion of the 4 inputs. The results of this measurement will be placed in the array of 4 integers pointed to (referenced by) the ***analogue** parameter.

Assuming a WASP is device 1, the following call

```
int analogue[4];
```

```
....
```

```
WSP_ReadInputs(1, analogue);
```

Will place the measured values of all 4 analogue inputs into the variables

```
analogue[0], analogue[1], analogue[2], analogue[3]
```

int WSP_SetOutputs(int devnum, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to set the on/off state of WASP's switching outputs. Bits 0 to 6 in the outputs parameter correspond to the desired on/off state of the outputs. A '1' corresponds to "ON" and 0 for "OFF".

For example the following call (which assumes a WASP is device 1 in the list) will set outputs 1,3 and 7 ON with all others OFF.

```
WSP_SetOutputs(1, 69)
```

The number '69' is actually 0000 0000 0100 0101 in binary, which should make this clear.

int WSP_SetSensitivity(int devnum, int sensitivity);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

sensitivity. Integer corresponding to the desired sensitivity setting of the analogue inputs. A zero indicates normal sensitivity which gives full scale 0 – 255 over the range 0 to 5v. A ‘1’ indicates a high sensitivity which gives full scale 0-255 over the range 0 to 2.4v

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Use this function to set the sensitivity of the analogue to digital convertor so that it best suits your application voltage range.

MaxiBee Functions

int MXB_SetOutputs(int devnum, unsigned int outputs1, unsigned int outputs2)

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

outputs1 unsigned integer corresponding to the bit pattern to be applied to the outputs 1 to 32

outputs2 unsigned integer corresponding to the bit pattern to be applied to the outputs 33 to 64

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to set the on/off state of MaxiBee’s switching outputs. Bits 0 to 31 in the outputs1 parameter correspond to the desired on/off state of the first 32 outputs and outputs2 for the second 32. A ‘1’ corresponds to “ON” and 0 to “OFF”.

For example the following call (which assumes a MaxiBee is device 1 in the list) will set outputs 1,3 ,7, 32, 33 and 64 ON with all others OFF.

MB_SetOutputs(1, 69, 32771)

The number ‘69’ is actually 0000 0000 0100 0101 in binary,

The number ‘32771’ is actually 1000 0000 0000 0011 in binary, which should make this clearer.

DigiBee Functions

int DGB_SetOutputs(int devnum, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to set the on/off state of DigiBee's digital outputs. Bits 0 to 15 in the outputs parameter correspond to the desired on/off state of the outputs. A '1' corresponds to "ON" and 0 for "OFF".

For example the following call (which assumes a DigiBee is device 3 in the list) will set outputs 1,3 ,7 and 16 ON with all others OFF.

```
DGB_SetOutputs(3, 32837)
```

The number '32837' is actually 1000 0000 0100 0101 in binary, which should make this clear.

int DGB_ReadInputs(int devnum, int *inputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

***inputs.** The address of (or reference to) an integer corresponding to the bit pattern read from the inputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to read the on/off state of DigiBee's digital inputs. Bits 0 to 15 in the inputs parameter correspond to the digital state of the inputs. A '1' corresponds to "Logic High" or +5v and '0' for "Logic low" or 0v.

For example the following call (which assumes a DigiBee is device 3 in the list).....

```
int inputs
```

```
.....
```

```
DGB_ReadInputs(3, &inputs)
```

Will result with the variable "inputs" having the value of 39 if the current state of the digital inputs is ... All 0v except inputs 1,2,3 and 6 which are +5v

The number '39' is actually 0000 0000 0010 0111 in binary, which should make this clear.

DigiBee Plus Functions

int DGBP_SetOutputs(int devnum, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to set the on/off state of DigiBee+'s digital outputs. Bits 0 to 15 in the outputs parameter correspond to the desired on/off state of the outputs. A '1' corresponds to "ON" and 0 for "OFF".

For example the following call (which assumes a DigiBee+ is device 4 in the list) will set outputs 1,3 ,7 and 16 ON with all others OFF.

```
DGBP_SetOutputs(4, 32837)
```

The number '32837' is actually 1000 0000 0100 0101 in binary, which should make this clear.

int DGBP_ReadInputs(int devnum, int *inputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

***inputs.** The address of (or reference to) an integer corresponding to the bit pattern read from the inputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to read the on/off state of DigiBee+'s digital inputs. Bits 0 to 15 in the inputs parameter correspond to the digital state of the inputs. A '1' corresponds to "Logic High" or +5v and '0' for "Logic low" or 0v.

For example the following call (which assumes a DigiBee+ is device 5 in the list).....

```
int inputs
```

```
.....
```

```
DGBP_ReadInputs(5, &inputs)
```

Will result with the variable "inputs" having the value of 39 if the current state of the digital inputs is ... All 0v except inputs 1,2,3 and 6 which are +5v

The number '39' is actually 0000 0000 0010 0111 in binary, which should make this clear.

int DGBP_ReadAnalogueInputs(int devnum, int *aip1, int *aip2, int *aip3, int *aip4);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

***aip1 - *aip4** The address (or reference to) 4 integers to be used to hold the measured analogue input values.

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Calling this function will result in an analogue to digital conversion of the 4 inputs. The results of this measurement will be placed in the array of 4 integers pointed to (referenced by) the ***aip1 - *aip4** parameters.

Assuming a DigiBee+ is device 1, the following call

```
int aip1, aip2, aip3, aip4 ;
```

```
....
```

```
DGBP_ReadAnalogueInputs (1, &aip1, &aip2, &aip3, &aip4);
```

Will place the measured values of all 4 analogue inputs into the variables aip1, aip2, aip3 and aip4

int DGBP_SetSensitivity(int devnum, int sensitivity);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

sensitivity. Integer corresponding to the desired sensitivity setting of the analogue inputs. A zero indicates normal sensitivity which gives full scale 0 – 255 over the range 0 to 5v. A ‘1’ indicates a high sensitivity which gives full scale 0-255 over the range 0 to 2.4v

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Use this function to set the sensitivity of the analogue to digital convertor so that it best suits your application voltage range.

MotorBee Functions

```
int MTB_SetMotors(int devnum, int on1, int speed1, int on2, int speed2, int on3, int speed3, int on4, int speed4, int servo);
```

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

on1 – on4. The on/off state of motors 1 to 4 respectively. '0' implies off and '1' implies ON

speed1 – speed4. The speed of motor1 to motor4 respectively. Each motor can have a speed set in the range 0-250. Note that the speed parameter for a given motor is irrelevant if the motor is 'off'

servo. A value of 128 sets the servo to its mid position, '0' to full anti-clockwise, '255' for full clockwise and all other values are proportional between these limits.

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Sets the speed and on/off status of all 4 motors and the position of the servo.

Note 1: When two outputs are used to provide reversible motor control then one of the pair of outputs should be set 'off' For example: if a motor is connected between outputs 1 and 2 then to make it go forward at half speed you would set....

speed1 = 125, on1 = 1, speed2 = 0, on2=0

for full speed reverse...

speed1 = 0, on1 = 0, speed2 = 250, on2=1

When in this reversible motor configuration you should not have both outputs on at the same time.

Note 2: Although servos are generally very similar in the way they operate, there are some differences in some of their specifications. For example, the range over which they can turn can vary from almost a full circle to just 90 degrees. By far the most common type has a range of 180 degrees. For this type the servo control using values of 0 – 255 will correspond to 0 to 180 degrees (or more correctly –90 to +90 degrees from mid point). The control signal used by MotorBee to move the servo conforms to the most common standard of a pulse of varying width between 1 and 2ms with 1.5ms being the neutral (mid) point. There are some servos that will not operate correctly at the extremes of this range (i.e. near the 1ms and 2ms pulse width settings). You should restrict your choice of position to accommodate any such requirements.

int MTB_Digital_IO(int devnum, int *inputs, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

***inputs.** The address of (or reference to) an integer corresponding to the bit pattern read from the inputs

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Used to both read the on/off state of MotorBee's digital inputs and to set the on/off state of MotorBee+'s digital outputs. Bits 0 to 5 in the inputs parameter correspond to the digital state of the inputs and bits 0 to 3 in the outputs parameter correspond to the desired state of the digital outputs. A '1' corresponds to "Logic High" or +5v and '0' for "Logic low" or 0v.

For example the following call (which assumes a MotorBee is device 2 in the list).....

int inputs

.....

MTB_Digital_IO (2, &inputs, 9)

Will set outputs 1 and 4 ON with 2 and 3 OFF.

The number '9' is actually 0000 0000 0000 1001 in binary, which should make this clear.

It will also result with the variable "inputs" having the value of 39 if the current state of the digital inputs is ...

All 0v except inputs 1,2,3 and 6 which are +5v

The number '39' is actually 0000 0000 0010 0111 in binary, which should make this clear.

StepperBee Functions

int STP_RunMotor1(int devnum, int steps, int interval, int direction, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

steps. The number of discrete steps to be taken by the motor in the range 1 to 16,000

interval. The interval in milliseconds between each step in the range 1 to 16,000

direction. Forward or reverse. (0 for forward, 1 for reverse)

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

This function is used to set a task for the Stepper motor to perform. The task is based on a number of steps in a given direction with a specified time interval between them (i.e. speed). The state of StepperBee's switching outputs can also be set for the duration of this task by using the outputs parameter. Bits 0 to 2 in the outputs parameter correspond to the desired on/off state of the three outputs associated with this motor. A '1' corresponds to "ON" and 0 for "OFF".

For example the following call (which assumes a StepperBee is device 1 in the list) will cause the motor to rotate clockwise 200 steps with 10ms between steps with outputs 1 and 2 ON and 3 OFF.

```
STP_RunMotor1 (1, 200, 10, 0, 3)
```

The number '3' is actually 0000 0000 0000 0011 in binary, which should make this clear.

int STP_StopMotor1(int devnum, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

This function is used to stop the stepper motor immediately, even if it is in the middle of a task. The state of StepperBee's switching outputs can also be set at the same time using the outputs parameter. Bits 0 to 2 in the outputs parameter correspond to the desired on/off state of the three outputs associated with this motor. A '1' corresponds to "ON" and 0 for "OFF".

For example the following call (which assumes a StepperBee is device 1 in the list) will cause the motor to stop immediately and set outputs 1 and 2 ON and 3 OFF.

```
STP_StopMotor1 (1, 3)
```

The number '3' is actually 0000 0000 0000 0011 in binary, which should make this clear.

Tip: This function call can be useful for manipulating the outputs while a motor is stopped, by using repeated calls to stop the motor with the output bits set according to your requirements.

```
int STP_RunMotor2(int devnum, int steps, int interval, int direction, int outputs);
```

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

steps. The number of discrete steps to be taken by the motor in the range 1 to 16,000

interval. The interval in milliseconds between each step in the range 1 to 16,000

direction. Forward or reverse. (0 for forward, 1 for reverse)

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

This function is used to set a task for the Stepper motor to perform. The task is based on a number of steps in a given direction with a specified time interval between them (i.e. speed). The state of StepperBee's switching outputs can also be set for the duration of this task by using the outputs parameter. Bits 0 to 2 in the outputs parameter correspond to the desired on/off state of the three outputs associated with this motor. A '1' corresponds to "ON" and 0 for "OFF".

For example the following call (which assumes a StepperBee is device 1 in the list) will cause the motor to rotate clockwise 200 steps with 10ms between steps with outputs 1 and 2 ON and 3 OFF.

```
STP_RunMotor2 (1, 200, 10, 0, 3)
```

The number '3' is actually 0000 0000 0000 0011 in binary, which should make this clear.

int STP_StopMotor2(int devnum, int outputs);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

outputs. Integer corresponding to the bit pattern to be applied to the outputs

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

This function is used to stop the stepper motor immediately, even if it is in the middle of a task. The state of StepperBee's switching outputs can also be set at the same time using the outputs parameter. Bits 0 to 2 in the outputs parameter correspond to the desired on/off state of the three outputs associated with this motor. A '1' corresponds to "ON" and 0 for "OFF".

For example the following call (which assumes a StepperBee is device 1 in the list) will cause the motor to stop immediately and set outputs 1 and 2 ON and 3 OFF.

```
STP_StopMotor2 (1, 3)
```

The number '3' is actually 0000 0000 0000 0011 in binary, which should make this clear.

Tip: This function call can be useful for manipulating the outputs while a motor is stopped, by using repeated calls to stop the motor with the output bits set according to your requirements.

int STP_SetStepMode(int devnum, int M1Mode, int M2Mode);

Parameters

devnum. An integer corresponding to the position of the connected board in the list found by InitBee()

M1Mode. Integer corresponding to the Step mode required. For motor 1
'0' sets Wave Step and '1' sets Full Step.

M2Mode. Integer corresponding to the Step mode required for motor 2
'0' sets Wave Step and '1' sets Full Step.

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

Stepper motors can be driven by a number of different patterns of pulses. StepperBee provides for two of these patterns. Setting the step mode to Wave step means that at any one time only one phase of their 4-phase coils is energized. This results in a more energy efficient way of driving the motor requiring less current. Setting the step mode to full step means that there are always two phases on at any one time. This provides more torque at the expense of more energy / current. The number and degree of movement of any steps is not affected by this choice.

```
int STP_GetCurrentStatus(int devnum, int *M1Active, int *M2Active, int *M1Steps, int *M2Steps, int *Inputs);
```

Parameters

- devnum.** An integer corresponding to the position of the connected board in the list found by InitBee()
- *M1Active.** The address of (or reference to) an integer corresponding to the current status of motor 1. Return value of '0' implies stopped , '1' implies active (running).
- *M2Active.** The address of (or reference to) an integer corresponding to the current status of motor 2. Return value of '0' implies stopped , '1' implies active (running).
- *M1Steps.** The address of (or reference to) an integer corresponding to the current status of motor 1. The return value is the number of steps that motor 1 has left to do to complete it's current task.
- *M2Steps.** The address of (or reference to) an integer corresponding to the current status of motor 2. The return value is the number of steps that motor 2 has left to do to complete it's current task.
- *Inputs.** The address of (or reference to) an integer corresponding to the current state of the digital inputs. The return value corresponds to the bit pattern of the digital inputs. Bits 0-5 correspond to inputs 1 – respectively

Returns

An integer reporting the success or failure of this function. Return value of zero indicates success. Any other value corresponds to one of the defined error codes in <be.h>

Description

The GetCurrentStatus() function can be called at any time to view the progress of both motors and examine the state of the digital inputs. As an example we will assume motor 1 is stopped, motor 2 is active with 150 steps left to go and only digital input 3 is on (+5v). The following call.....

```
int M1Active, M2Active, M1Steps, M2Steps, Inputs;  
.....  
STP_GetCurrentStatus(1, &M1Active, &M2Active, &M1Steps, &M2Steps, &Inputs)  
.....
```

Will result in the following values returned in the passed variables

M1Active = 0, M2Active = 1, M1Steps = 0, M2Steps = 150, Inputs = 4

The number '4' is actually 0000 0000 0000 0100 in binary, which should make this clear.

Using BeeHive DLL Functions From Visual Basic

Although the DLL functions described above operate exactly the same way when called from visual basic, there are a few small differences in syntax that should be observed in the construction of your software. The correct visual basic format for function declarations and calling syntax is listed below for all of the functions already described above..

General Functions

Function Declarations

```
Declare Function InitBee Lib "bee.dll" () As Integer
```

```
Declare Function GetBoardType Lib "bee.dll" (ByVal devnum As Integer) As Integer
```

Function Usage

```
Dim NumBoards, BoardType, devnum As Integer
```

```
NumBoards = InitBee()
```

```
BoardType = GetBoardType(devnum)
```

(Where devnum is a number between 0 and NumBoards)

Mini Bee

Function Declarations

```
Declare Function MB_SetOutputs Lib "bee.dll" ( ByVal devnum As Integer, ByVal outputs As Integer) As Integer
```

Function Usage

```
Dim devnum, outputs As Integer
```

```
MB_SetOutputs(devnum, outputs)
```

WASP

Function Declarations

```
Declare Function WSP_SetOutputs Lib "bee.dll" ( ByVal devnum As Integer, ByVal outputs As Integer) As Integer
```

```
Declare Function WSP_SetSensitivity Lib "bee.dll" ( ByVal devnum As Integer, ByVal sensitivity As Integer) As Integer
```

```
Declare Function WSP_ReadInputs Lib " bee.dll" (ByVal devnum As Integer, ByVal analogue As Integer) As Integer
```

Function Usage

```
Dim devnum, outputs, sensitivity As Integer
```

```
Dim analogue(4) As Integer
```

```
WSP_SetOutputs(devnum, outputs)
```

```
WSP_SetSensitivity(devnum, sensitivity)
```

```
WSP_ReadInputs(devnum, analogue(0) )
```

Maxi Bee

Function Declarations

```
Declare Function MXB_SetOutputs Lib "bee.dll" ( ByVal devnum As Integer,  
                                               ByVal outputs1 As Integer,  
                                               ByVal outputs2 As Integer) As Integer
```

Function Usage

```
Dim devnum, outputs1, outputs2 As Integer
```

```
MXB_SetOutputs(devnum, outputs1, outputs2)
```

DigiBee

Function Declarations

```
Declare Function DGB_SetOutputs Lib "bee.dll" ( ByVal devnum As Integer, ByVal outputs As Integer ) As Integer  
Declare Function DGB_ReadInputs Lib "bee.dll" ( ByVal devnum As Integer, ByVal inputs As Integer ) As Integer
```

Function Usage

```
Dim devnum, inputs, outputs As Integer
```

```
DGB_SetOutputs(devnum, outputs)  
DGB_ReadInputs(devnum, inputs)
```

DigiBee Plus

Function Declarations

```
Declare Function DGBP_SetOutputs Lib "bee.dll" ( ByVal devnum As Integer, ByVal outputs As Integer ) As Integer  
Declare Function DGBP_ReadInputs Lib "bee.dll" ( ByVal devnum As Integer, ByVal inputs As Integer ) As Integer  
Declare Function DGBP_ReadAnalogueInputs Lib "bee.dll" ( ByVal devnum As Integer,  
                                                         ByRef aip1 As Integer,  
                                                         ByRef aip2 As Integer,  
                                                         ByRef aip3 As Integer,  
                                                         ByRef aip4 As Integer  
                                                         ) As Integer
```

Function Usage

```
Dim devnum, inputs, outputs As Integer  
Dim aip1, aip2, aip3, aip4 As Integer
```

```
DGBP_SetOutputs(devnum, outputs)  
DGBP_ReadInputs(devnum, inputs)  
DGBP_ReadAnalogueInputs(devnum, aip1, aip2, aip3, aip4)
```

MotorBee

Function Declarations

```
Declare Function MTB_SetMotors Lib "bee.dll" ( ByVal devnum As Integer,  
                                              ByVal on1 As Integer,  
                                              ByVal speed1 As Integer,  
                                              ByVal on2 As Integer,  
                                              ByVal speed2 As Integer,  
                                              ByVal on3 As Integer,  
                                              ByVal speed3 As Integer,  
                                              ByVal on4 As Integer,  
                                              ByVal speed4 As Integer,  
                                              ByVal servo As Integer  
                                              ) As Integer
```

```
Declare Function MTB_Digital_IO Lib "bee.dll" ( ByVal devnum As Integer,  
                                               ByRef inputs As Integer  
                                               ByVal outputs As Integer  
                                               ) As Integer
```

Function Usage

```
Dim devnum, on1, speed1, on2, speed2, on3, speed3, on4, speed4, servo, inputs, outputs As Integer
```

```
MTB_SetMotors(devnum, on1, speed1, on2, speed2, on3, speed3, on4, speed4, servo)
```

```
MTB_Digital_IO(devnum, inputs, outputs)
```

Stepper Bee

Function Declarations

```
Declare Function STP_RunMotor1 Lib "bee.dll" ( ByVal devnum As Integer,  
                                              ByVal steps As Integer,  
                                              ByVal interval As Integer,  
                                              ByVal direction As Integer,  
                                              ByVal outputs As Integer  
                                              ) As Boolean
```

```
Declare Function STP_RunMotor2 Lib "bee.dll" ( ByVal devnum As Integer,  
                                              ByVal steps As Integer,  
                                              ByVal interval As Integer,  
                                              ByVal direction As Integer,  
                                              ByVal outputs As Integer  
                                              ) As Boolean
```

```
Declare Function STP_StopMotor1 Lib "bee.dll" ( ByVal devnum As Integer, ByVal outputs As Integer ) As Boolean
```

```
Declare Function STP_StopMotor2 Lib "bee.dll" ( ByVal devnum As Integer, ByVal outputs As Integer ) As Boolean
```

```
Declare Function STP_SetStepMode Lib "bee.dll" (      ByVal devnum As Integer,  
                                                  ByVal M1Mode As Integer,  
                                                  ByVal M2Mode As Integer  
                                                  ) As Boolean
```

```
Declare Function STP_GetCurrentStatus Lib "bee.dll" (  ByVal devnum As Integer,  
                                                      ByRef M1Active As Integer,  
                                                      ByRef M2Active As Integer,  
                                                      ByRef M1Steps As Integer,  
                                                      ByRef M2Steps As Integer,  
                                                      ByRef Inputs As Integer  
                                                      ) As Boolean
```

Function Usage

```
Dim devnum, steps, interval, direction, inputs, outputs As Integer  
Dim M1Mode, M2Mode, M1Active, M2Active, M1Steps, M2Steps As Integer
```

```
STP_RunMotor1(devnum, steps, interval, direction, outputs)  
STP_RunMotor2(devnum, steps, interval, direction, outputs)  
STP_StopMotor1(devnum, outputs)  
STP_StopMotor2(devnum, outputs)  
STP_SetStepMode(devnum, M1Mode, M2Mode)  
STP_GetStatus(devnum, M1Active, M2Active, M1Steps, M2Steps, Inputs)
```

Terms of Use for all Goods Supplied

Definitions

'Supplier' shall mean PC Control Ltd.

'Buyer' shall mean the person, company or any other body that purchases or agrees to purchase Goods.

'Goods' shall mean all goods and services which the Buyer agrees to buy from the Supplier including replacements for defective Goods, hardware, documentation and software products licensed for use by the Buyer.

Use of the Goods in any way by the Buyer constitutes acceptance of these terms and conditions.

Terms and Conditions

1. The Goods are intended to be part of the buyer's own design of apparatus and not a finished product in their own right.
2. The Goods supplied are not to be used in any design where there is a risk, however small, either directly or indirectly, of death or personal injury.
3. The Buyer will be responsible for ensuring the fitness for purpose of the Goods for the Buyer's application.
4. To the extent permitted by law, the Supplier accepts no liability whatsoever or howsoever arising in respect of loss, damage or expense arising from errors in information or advice provided whether or not due to the Supplier's negligence or that of its employees, agents or sub-contractors save for any loss or damage arising from death or personal injury.
5. To the extent permitted by law, the Supplier shall not be liable to the Buyer by reason of any representation (unless fraudulent), or any implied warranty, condition or other term, or any duty at common law, or under the express terms of any Contract with the Buyer, for any indirect, special or unforeseen loss or damage (whether for loss of profit or otherwise), costs, expenses or other claims for compensation whatsoever (whether caused by the negligence of the Supplier, its employees or agents or otherwise) which arise out of or in connection with the supply of the Goods or their use or resale by the Buyer.
6. The entire liability of the Supplier under or in connection with the Contract with the Buyer shall not exceed the price of the Goods except as expressly provided in these terms and conditions.
7. These terms are an important part of the full terms and conditions of business as published on the website at www.pc-control.co.uk/general-terms.htm which also apply.