



## **BRAIN Software**

### **GMKE00005 Revision 3; August 2009**

#### **1.0 Overview**

This document describes the software for use with the BRAIN system. The document covers software installation, utilities, and sample programs, and it describes the BRAIN library application programming interface (API). This document does not replace the detailed manuals and help information available for the programming environment (IAR Workbench) or for the C programming language used to control the BRAIN.

#### **2.0 Installation**

The BRAIN Support installer performs multiple functions on the target system to provide all the software support needed to use the BRAIN hardware. The installer loads:

1. Sample software projects
2. A BRAIN software library and the source code
3. A utility to download code to the BRAIN (BSLUSB)
4. A Wizard program
5. USB drivers for communication with the BRAIN (FTDI's CDM driver)
6. IAR Workbench, Kickstart edition

Because the BRAIN installer includes a driver for the USB interface, you must install the software prior to connecting the BRAIN hardware to a USB port.

The default locations for the installed software elements are shown in Table 2.1.

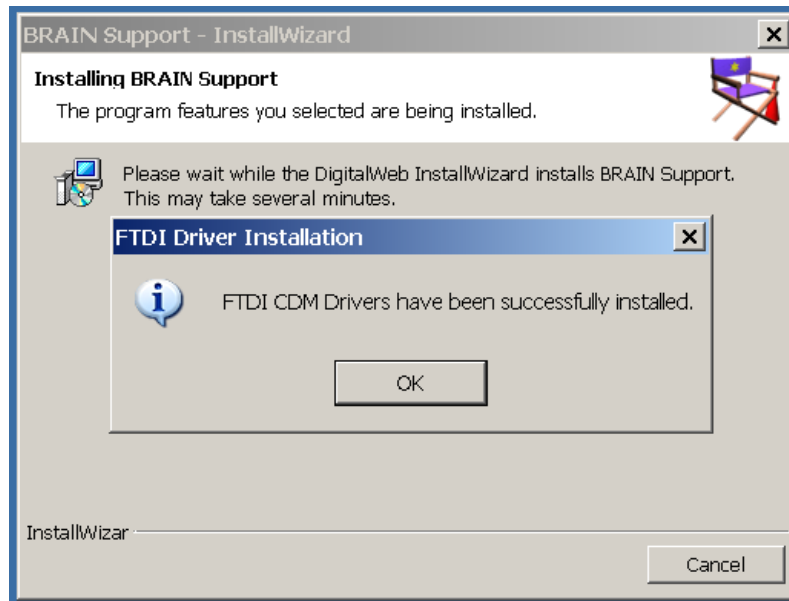
**Table 2.1. Default Locations of Files Installed by the BRAIN Software**

<b>Base folder</b>	<b>Subfolder</b>	<b>Description</b>
\My Documents\Best\Brain\Brain Projects\		
	default\	standard program on user processor
	LEDFlash\	simple project to flash LED
	wizard\	project that works with BRAIN Wizard
	splittest\	Example program that uses the split library
	driveremap\	Example program that demonstrates joystick mixing, digital input, string output, and time functions
\My Documents\Best\Brain\Brain Library Source\		
	UserLibrary	Includes setup, input/output, scaling, LED and other functions.
	CommonLibrary	Includes messaging and string functions.
	Combined	Builds the library that combines the user library and the common library.
	SplitLibrary	Example library that includes a method of splitting a joystick into multiple zones. Uses an object oriented approach.
\Program Files\Best\BRAIN\		
	bin\	location of Boot Loader program (USBBSL) and Wizard
	include\	include files for library
	lib\	library object files
\Program Files\IAR Systems\Embedded Workbench 5.0		Root location for IAR Kickstart program files.

The installation process is summarized by the screen shots shown in Figures 2.1 through 2.4. To complete the installation, make the default selections for each of the dialog boxes that appear during the installation process. Note that not all of the dialog boxes are provided in the figures.

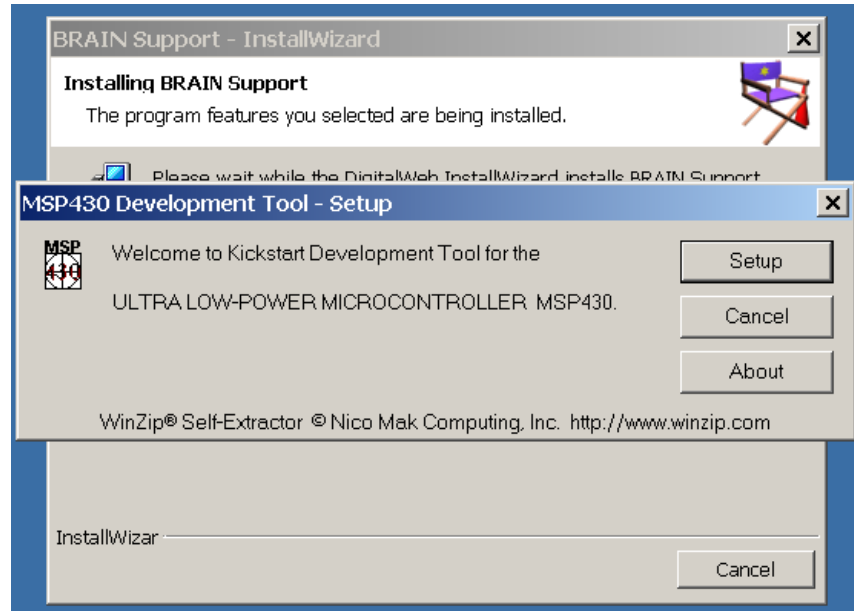


**Figure 2.1. Initial BRAIN Support Installer Dialog Box**



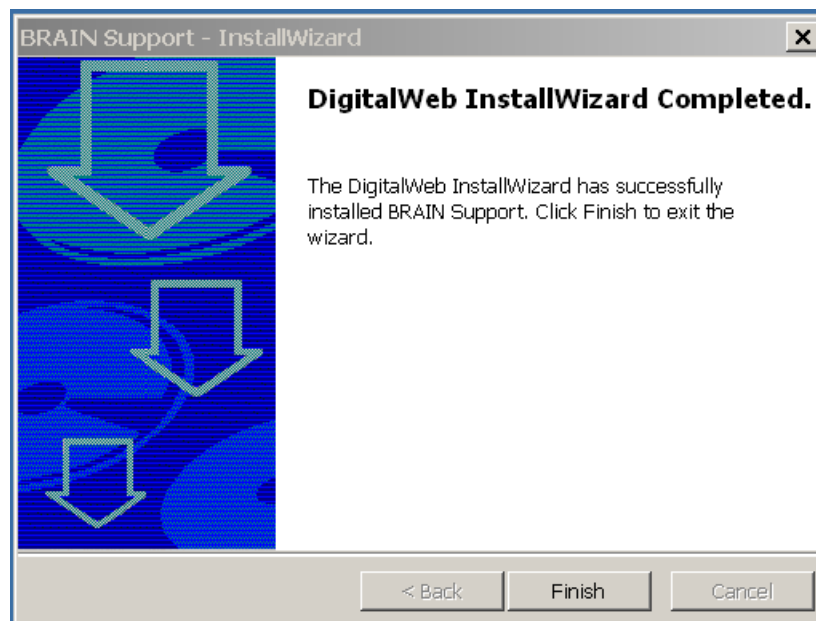
**Figure 2.2. Dialog Box Showing Completion of the USB Driver Installation**

Once the dialog box shown in Figure 2.2 appears, click the OK button. The initial setup screen shown in Figure 2.3 will appear. Click the “Setup” button in the dialog box shown in Figure 2.3 to initiate the installation of the IAR software. Sometimes the IAR setup dialog is hidden by the main BRAIN installation window and the installer will appear to “hang” but is actually waiting for user input. To fix this problem, move the BRAIN installer window to the side to uncover the IAR dialog setup button.



**Figure 2.3. Initial IAR Setup Dialog Box**

Note that the IAR installer will uninstall the IAR software if the IAR software already exists on the target system. This is normally not an issue unless the BRAIN software installer is run multiple times, or if the IAR software is already installed on the target system. The IAR setup dialog can be cancelled to avoid uninstalling the IAR software when the software is already installed on the target system



**Figure 2.4. Final Dialog Box for BRAIN Software Installation**

## 2.1 BRAIN Detected as Microsoft Serial Mouse

On some machines, when the BRAIN is plugged into a USB port, the system detects it as a Microsoft Serial Mouse. This condition will prevent the BSLUSB program from downloading programs to the BRAIN hardware. A “failed to enumerate port” error may appear when trying to download a program to the BRAIN. (This is also the error that will appear if the BRAIN is not plugged into the USB port when a download is attempted.) To fix the detected-as-mouse problem, on a Windows XP system:

1. Right click on My Computer.
2. Select Properties.
3. Select the Hardware Tab.
4. Select Device Manager.
5. Click on the + sign next to Mouse and other pointing devices.
6. Click on "Microsoft Serial Mouse".
7. Press the Delete key.
8. Confirm that you want to delete the mouse.

Note that if the BRAIN is later plugged into a different USB port, the problem may occur again.

An alternative (perhaps better) way to fix this problem is to turn off “enumeration” for the port that is created when you plug in a BRAIN.

1. Right click on My Computer.
2. Select Properties.
3. Select the Hardware Tab.
4. Select Device Manager.
5. Click on the + sign next to Ports (COM & LPT)
6. Plug in the BRAIN.
7. Select (double click) the USB Serial Port that appears on the list (e.g., COM12)
8. Select the Port Settings Tab.
9. Click the Advanced button
10. Under the Miscellaneous Options area, uncheck the box for Serial Enumerator.
11. Click the OK buttons on the various dialog boxes to complete the action.

### 3.0 Included Programs

Three programs are provided with the IAR project definitions in the standard BRAIN software installation:

1. LEDFlash – this is a simple program that can be used to demonstrate the process of compiling and loading a program onto the BRAIN.
2. default – this is the program that is normally loaded into your BRAIN when you receive it. It allows the use of all input and output channels.
3. wizard – this is the program that is used in combination with the BRAIN-wizard-generated header file to create a BRAIN program.
4. splittest – This program demonstrates the use of the SplitStick Library. It allows a joystick to be segmented into multiple zones and to have different functions defined for each of the zones. The program shows an example of an object oriented approach to programming the BRAIN (without actually using C++).
5. driveremap – This program demonstrates several aspects of the BRAIN API including string functions and serial output for debugging and time functions. It also shows how to remap the joystick input to allow for a more “natural” control, i.e., forward on the joystick produces forward motion, right on the joystick produces a right turn, etc.

The locations of these projects were provided in Table 2.1. The last two example programs (splittest and driveremap) are left for you to examine and figure out how the programs work.

#### 3.1 LEDFlash

The LEDFlash program does not access any of the input or output functions of the BRAIN; it performs a minimal amount of initialization and causes the user LED to flash. This program is TI’s version of the standard C “hello world” program that uses the user LED for output. A side effect of this program is that the control LED will also flash, indicating that there is no communication from the user processor to the control processor. The LEDFlash program provides a simple way to test the process of compiling, linking, and downloading a program to the BRAIN.

#### 3.2 Default

The default program allows all channels to be used both as servo and motor output which simplifies the use of the BRAIN for testing purposes. Each input channel provides proportional servo control and proportional motor speed control. Servo output channels 5 and 6 are mapped to input channels 3 and 4. There is a built-in input gain of 1.25 which should allow full speed output for the motor when the joystick is at either of the extreme positions. The digital inputs are defined as motor limits for each motor output channel.

Input 1 – Motor 1, joystick 1 positive (to the right)

Input 2 – Motor 1, joystick 1 negative (to the left)

Input 3 – Motor 2, joystick 2 positive (note that joystick 2 positive is down, not up)

Etc.

When the digital input is connected to ground (e.g., by a switch) the specified motor will be disabled in the specified direction.

### **3.3 Wizard**

The wizard program relies on a header file (wizardgen.h) to configure how the BRAIN will function (e.g., channel selection, thresholds, etc.). The header file is generated by an executable Windows program (BRAIN\_Wizard.exe) that allows you to select and enter options through a dialog interface as described in Section 4.1.

### **3.4 BRAIN Project Type**

In addition to the provided programs and projects, the BRAIN installation defines a BRAIN project type that creates a basic framework for creating a BRAIN program. To create a new BRAIN project, perform these steps from within the IAR Workbench:

1. Select “create new project”. A dialog box for this operation appears when IAR is first started, or you can start a new project by selecting the Project->Create New Project menu item.
2. Click the “+” sign next to the “C” in project templates dialog box.
3. Select “brain” as the project template, and then click the “OK” button.
4. Select a location and name under which to save the project, and click the “Save” button.
5. The new project will open with a single file named “main.c.” The project includes the required settings to link the BRAIN library functions, and will function without modification.
6. Select the File -> Save Workspace menu item. Enter a name for your workspace and then click the “Save” button.

The main.c file provides a good starting point for developing your own user program.

## 4.0 Programming the BRAIN

You can use the IAR Workbench to create your own programs for the BRAIN. The projects supplied with the BRAIN software installation provide the settings necessary to link the project source code with the BRAIN library described in Section 5. The basic sequence required to replace the program loaded into the BRAIN is:

1. Start the IAR Workbench.
2. Load a project.
3. Make any modifications to the project source code or other project files (including replacing the wizardgen.h file for a wizard project.)
4. Recompile and link the source code.
5. Download the executable code to the BRAIN.

The IAR Workbench can be launched from the Windows Start Menu. To load a project, select the File->Open menu item, and then browse to one of the BRAIN project locations. Alternatively, you can start the IAR Workbench by browsing to one of the BRAIN project locations using Windows Explorer and double clicking on the IAR IDE Workspace file (the one with the eww file extension).

Once the project is loaded, the source code can be modified as desired. Consult the IAR Workbench Help system for information on the mechanics of editing files within the IAR environment.

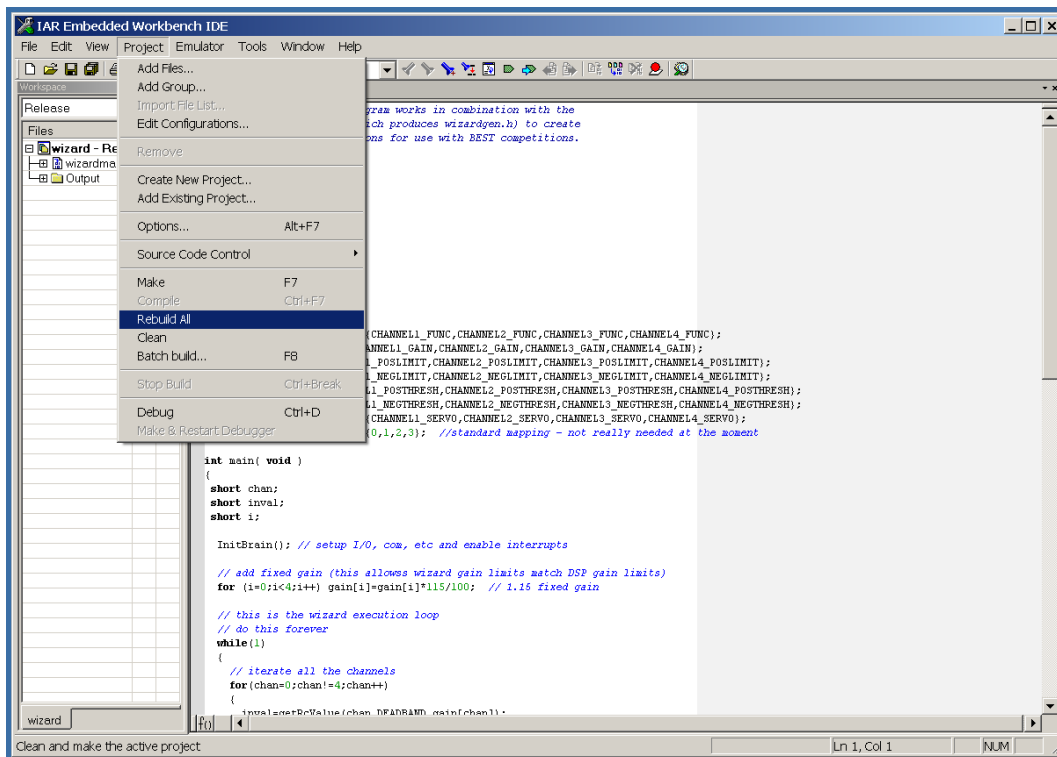


Figure 4.1. IAR Workbench, Rebuild All Function

When the modifications are complete, the code must be compiled and linked to the BRAIN library functions to create an executable program for the BRAIN. Use either the Project Rebuild or Project Rebuild All menu functions to create the executable. Figure 4.1 shows the menu for the Rebuild All function; Rebuild All forces each of the project files to be recompiled regardless of whether or not the IAR software detects that any of the project files has been modified. Although this function takes a few seconds longer to execute, it ensures that the project is up to date. This function is particularly useful when working with the wizard project where the wizardgen.h file is modified outside of the IAR environment, because it will ensure that the executable code is consistent with the configuration created by the Wizard program.

Once the project executable has been rebuilt (compiled and linked without any errors), download the program to the BRAIN. With the BRAIN connected to the USB port of the computer, select the Tools->Boot Load via USB menu item to load the program into the BRAIN as shown in Figure 4.2

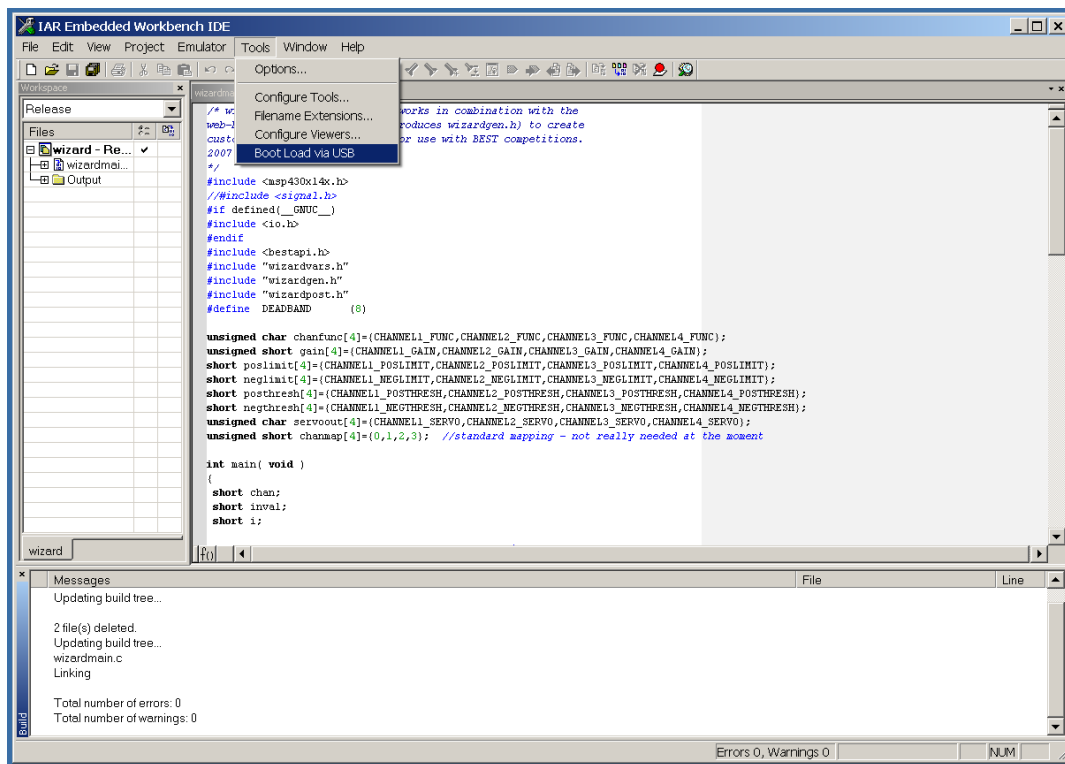


Figure 4.2. IAR Workbench, Boot Load Function

The download process takes approximately 15 seconds to complete. Once the process is complete, the output from the Bootloader will appear in the Tools window at the bottom of the IAR window. Figure 4.3 shows the text output from a successful program load.

```
MSP430 Bootstrap Loader - FTDI USB Interface (Version 1.04)
Mass Erase...
Additional mass erase cycles...
Transmit standard password...
BSL version: 1.10 - Family member: F149 - Process: 0043
Patch for flash programming required!
Load PC with 0x0C22...
Transmit standard password...
Load and verify patch "C:\Program Files\Best\BRAIN\bin\PATCH.TXT"...
Erase Check by file "wizard.txt"...
00 KByte 01 KByte 02 KByte Program "wizard.txt"...
00 KByte 01 KByte 02 KByte 2177 bytes programmed.
Verify"wizard.txt"...
00 KByte 01 KByte 02 KByte Resetting Target
Programming completed.Prog/Verify: 14.1 sec - Over all: 17.0 sec
```

**Figure 4.3. Example Tool Output Window for a Wizard Project Load**

#### 4.1 BRAIN Wizard

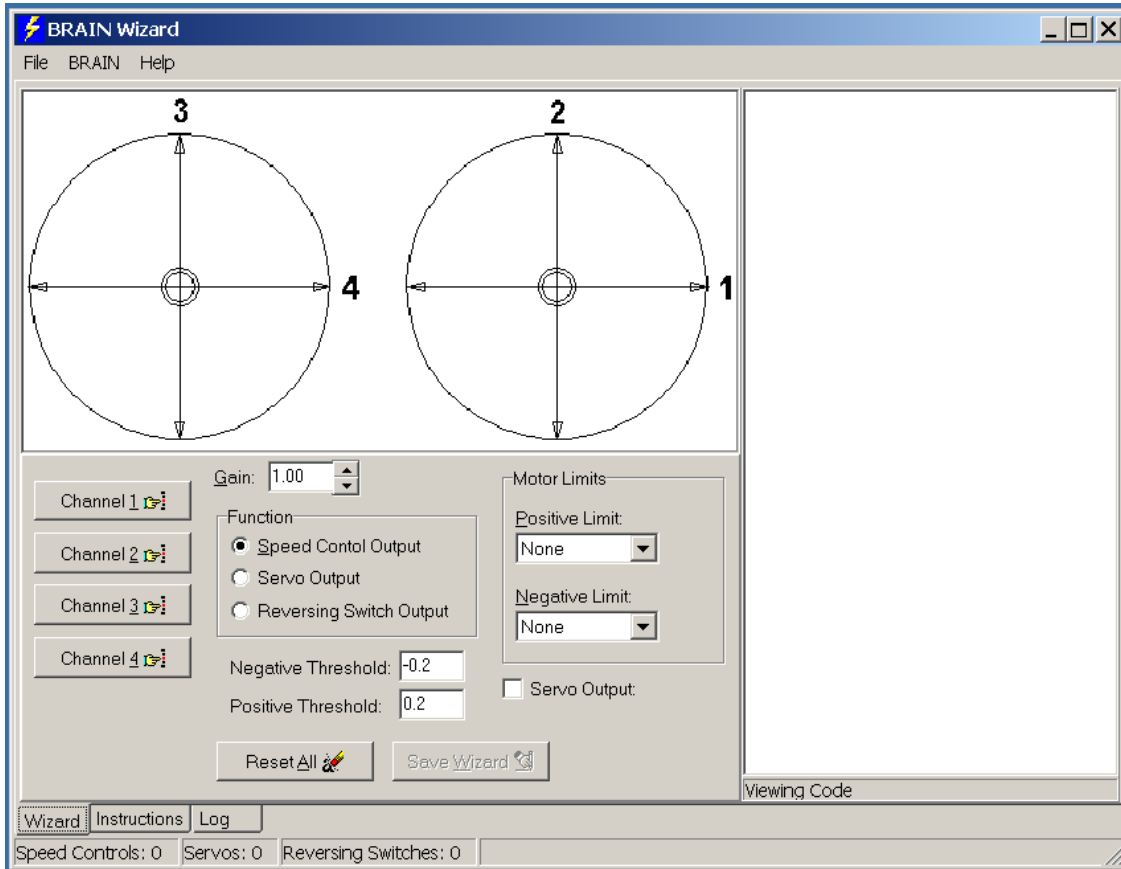
Use the Wizard program shown in Figure 4.4 to configure the control channels for the BRAIN wizard project. The Wizard program creates a header file (wizardgen.h) that is combined with the wizard source code (IAR wizard project) to create a BRAIN program.

The Wizard program can be launched from either the Windows Start menu or from the Tools menu from within the IAR Workbench. When you start the Wizard from within the IAR environment, it saves the generated header file to the current project directory. If you start the Wizard using the Windows Start menu, the default location for saving the generated header file is the My Documents folder and it must be moved to the IAR Wizard project directory in order to compile/link/download to the BRAIN. See Table 2.1 for location of BRAIN Wizard Project directory".

To create a complete header file, you must define the function of each channel and any associated options. After defining all of the channels, the header file is created by selecting the Save Wizard button. Each channel may be defined to be one of three types:

1. Speed Control Output – provides motor speed control proportional to the transmitter joystick position. Optionally, you can specify proportional servo output on the same channel.
2. Servo Output – provides servo position output proportional to the transmitter joystick position.
3. Reversing Switch Output – provides the same functionality as the servo/microswitch reversing setup shown in the *BEST Generic Kit Notes*.

When the transmitter joystick exceeds a user-specified position, the motor will be run at full speed; when the transmitter joystick exceeds another user-specified position, the motor will be run at full speed in the opposite direction. Optionally, you can specify proportional servo output on the same channel.



**Figure 4.4. BRAIN Wizard Interface**

The input/output channel mapping for the Wizard is defined so that input channel 1 always drives output channel 1, input channel 2 always drives output channel 2, and so on. This mapping is independent of the channel function definition. For example, if channels 1 and 2 are defined as speed control channels, and channels 3 and 4 are defined as servo output channels, then motors should be attached to motor output channels 1 and 2 and servos should be attached to servo output channels 3 and 4. If a channel is defined as reversing switch output with servo output, then both the motor and servo should be connected to the output connections corresponding to that channel number.

### 4.1.1 Parameters and Options

#### Gain

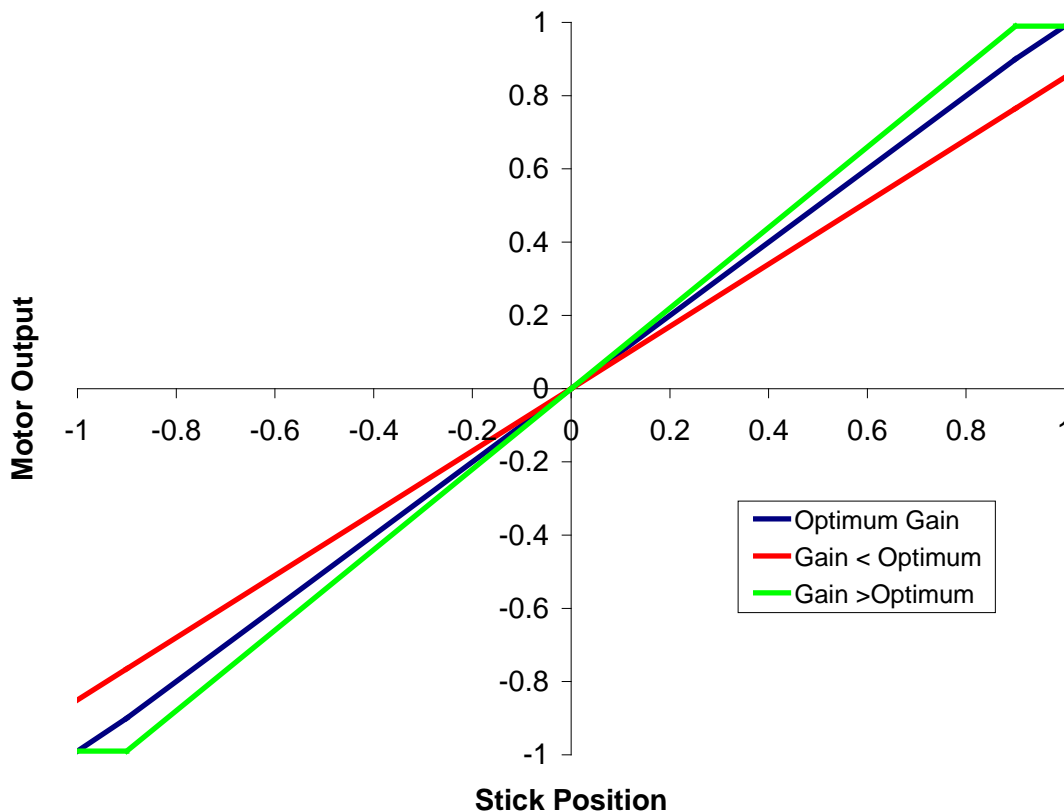
You can specify a gain value for each channel. The gain value is a multiplier by which the measured input joystick position is multiplied to determine the output on the

corresponding control channel:

$$\text{Output} = \text{Input} * \text{Gain}$$

The specification of gain allows you to optimize the control such that when the transmitter joystick reaches the maximum position, the motor speed (or servo position) reaches its maximum value. A gain value beyond the “optimized” value will not result in any further increase in speed or position, but will result in the motor reaching its maximum speed (or servo reaching its maximum position) prior to the transmitter joystick reaching its maximum position.

To achieve maximum power and speed from the motor, the gain value required is typically greater than one. Figure 4.5 provides an illustration of how a motor or servo will respond to different gain values.



**Figure 4.5. Illustration of Gain Values**

### Threshold

For reversing switch output, the threshold value is the fractional position of the transmitter joystick that must be exceeded before the motor is activated in the given direction. For purposes of determining the physical location of the joystick position, the wizard defines a value of 0 for the joystick in the centered (neutral) position, a value of

+1 when the joystick is at the end of its motion in the positive direction, and a value of -1 when the joystick is at the end of its motion in the negative direction. For example, if the threshold value is 0.2, then the joystick must be moved more than 20% of the distance from the center position to the positive stop before the motor will be activated. Note that the positive threshold must be larger than the negative threshold, but the positive threshold does not have to be a positive value and the negative threshold does not have to be a negative value. The terms positive and negative refer to the joystick direction.

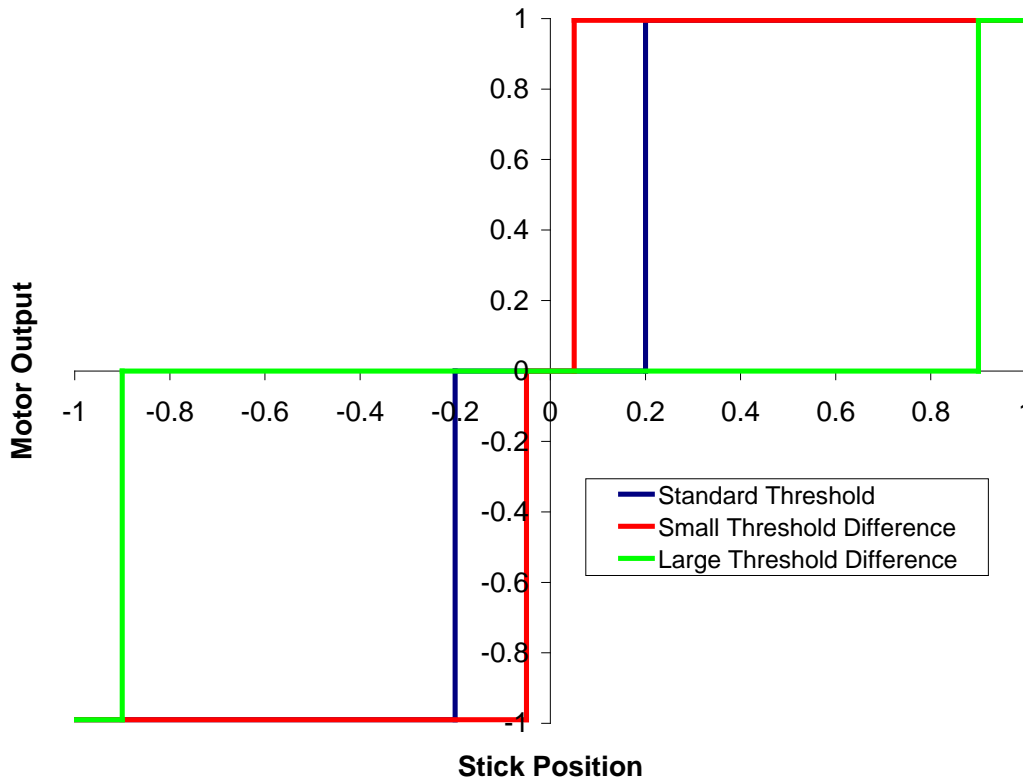


Figure 4.6. Illustration of Threshold Values

### Servo Output Checkbox

The servo output checkbox allows a channel defined for speed control or reversing switch output to also provide proportional control of a servo.

### Limit Switches

You can assign limit switches so that a switch closure connected to the specified digital input channel stops motor travel in the selected direction. For example, if option 2 is selected for the positive limit when defining channel 1, closure of a switch connected between digital input 2 and ground will cause the motor connected to channel 1 to stop when it is moving in the positive direction. There are separate inputs for positive and negative joystick directions. The limit switch function can be used with either the Speed Control Output or the Reversing Switch Output options.

## 5.0 BRAIN Library Application Programming Interface

Functions that allow programs to operate features of the BRAIN through the user processor have been combined into a library that should be linked with all BRAIN programs.

### 5.1 Default Locations

The BRAIN API library is contained in a file named `brainuser.r43` and installed in the `Program Files\Best\BRAIN\lib` folder. A header file that provides the function definitions (`bestapi.h`) is installed in the `Program Files\Best\BRAIN\include` folder. The default, wizard, and new Brain projects described in the project section of this document include the settings necessary to locate these files when compiling and linking the projects.

The source code to the library has been provided as an example of how to directly program the BRAIN hardware. BEST hubs will not support problems that arise from improper modifications to the library.

### 5.2 Operation Overview

The BRAIN API provides the means for reading control signals and performing internal communications between the user and control processors. Input signals from the radio receiver or tether input are monitored and decoded by interrupt service routines loaded when the BRAIN API is initialized. The selection of tether or radio input signal is done automatically: the tether signal takes precedence over the radio input. A time-based interrupt service routine passes servo and motor commands to the control processor via an internal communications link.

### 5.3 Functions of the BRAIN library API

*A note on indexing:* Diagrams, such as those in the *BRAIN Description* document, refer to the first input/output location for servo outputs, motor outputs and digital inputs as position 1. However in the BRAIN API, the first input/output item is given an index of zero. This numbering is consistent with the C language treatment of arrays where the first element in an array has the index value of zero. Please keep this in mind when using the API functions described below.

#### 5.3.1 Initialization

*void InitBrain(void)*

This function provides a one-call initialization of the system by calling **InitCpu()**, then calling **InitRc()**, and finally enabling interrupts. This should be the first step in every BRAIN program

*void InitCpu(void)*

This function initializes the state of the MSP430 controller and sets up various timers and interrupts. This function should be called prior to **InitRc()**. Since **initBrain()** calls this function, it is not normally called directly from the user program.

*void InitRc(void)*

This function initializes the state of the inputs used to capture the radio control signals from the receiver and tether system. This function should be called after the **InitCpu()** function. Since **initBrain()** calls this function, it is not normally called directly from the user program.

### 5.3.2 Input

*int getSwitch(unsigned char sw)*

This function returns the state of one of the input switch contacts indicated by **sw**. **sw** should be between 0 and 7. A return value of 1 indicates the switch is closed (input contact has been grounded) and a value of 0 indicates that the switch is open.

*short getRcValue(short chan, unsigned short deadband, unsigned short gainval)*

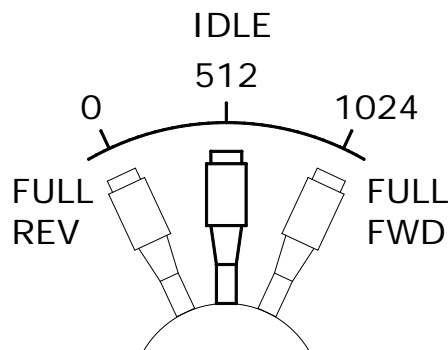
This function retrieves the input values obtained from either the tether connection or the receiver. Returned values should be in the range of 0 to 1024. 512 represents the stick center (null or idle) position (i.e.,  $idlevalue=512$ ), 0 represents full negative stick, and 1024 represents full positive stick as shown in Figure 5.1.

**chan** is the channel number and should be between 0 and 3.

**deadband** is the difference from the center position that the input must change before it is reported as being other than the center position.

**gainval** is a multiplier applied to the raw input value according to the following equation:

$$output = idlevalue + (inputvalue - idlevalue) * \frac{gainval}{100}$$



**Figure 5.1. Joystick Position Diagram (view from side).**

### 5.3.3 Output

*int setServo(short channel, short value)*

This function sets the servo value (position) of one of the six available servo output channels.

**channel** should be between 0 and 5.

**value** should be between 0 and 1024, where 0 indicates full negative value and 1024 indicates full positive value. 512 is the idle (no output) value.

The function returns 0 if it succeeds, or -1 for an out of range input parameter.

*int setServoRange(short channel, short width)*

This function sets the range of servo motion for one of the six available servo output channels.

**channel** should be between 0 and 5.

**width** specifies the pulse width variation in msec\*100 for the servo output of the specified channel. The width value should be between 50 and 180 (0.5 msec and 1.8 msec). The default (startup) value is 100 (1 ms). Width values beyond the default provide increased servo range (with decreased sensitivity), and width values less than the default provide increased sensitivity over a decreased total range. The change in sensitivity is a result of the fixed number of input values (1024) that can be used in the *setServo* function.

The function returns 0 if it succeeds, or -1 for an out of range input parameter. Since this function requires the control processor to calculate internal scaling parameters, repeated calls to this function may affect performance.

*int setMotor(short channel, short value)*

This function sets the output speed of one of the four available speed control channels.

**channel** should be between 0 and 3.

**value** should be between 0 and 1024, where 0 indicates full negative value and 1024 indicates full positive value. 512 is the idle (no output) value.

The function returns 0 if it succeeds, or -1 for an out of range input parameter.

*void writeLED(unsigned mask)*

This function controls the 8 user LEDs. Each bit of **mask** corresponds to one LED, i.e., bit 0 controls user LED 1, bit 1 controls user LED 1, etc. If the bit is 1, the LED is turned on; if the bit is 0, the LED is turned off.

*void writeLEDBinary(unsigned mask)*

This function works the same as the *writeLED* function except that the bit order is reversed so that bit 0 corresponds to user LED 8, etc. This function simplifies the display of binary numbers using the user LEDs.

*void setSingleLED(unsigned index)*

This function turns on the user LED specified by **index**.

*void clearSingleLED(unsigned index)*

This function turns off the user LED specified by **index**.

### 5.3.4 Miscellaneous

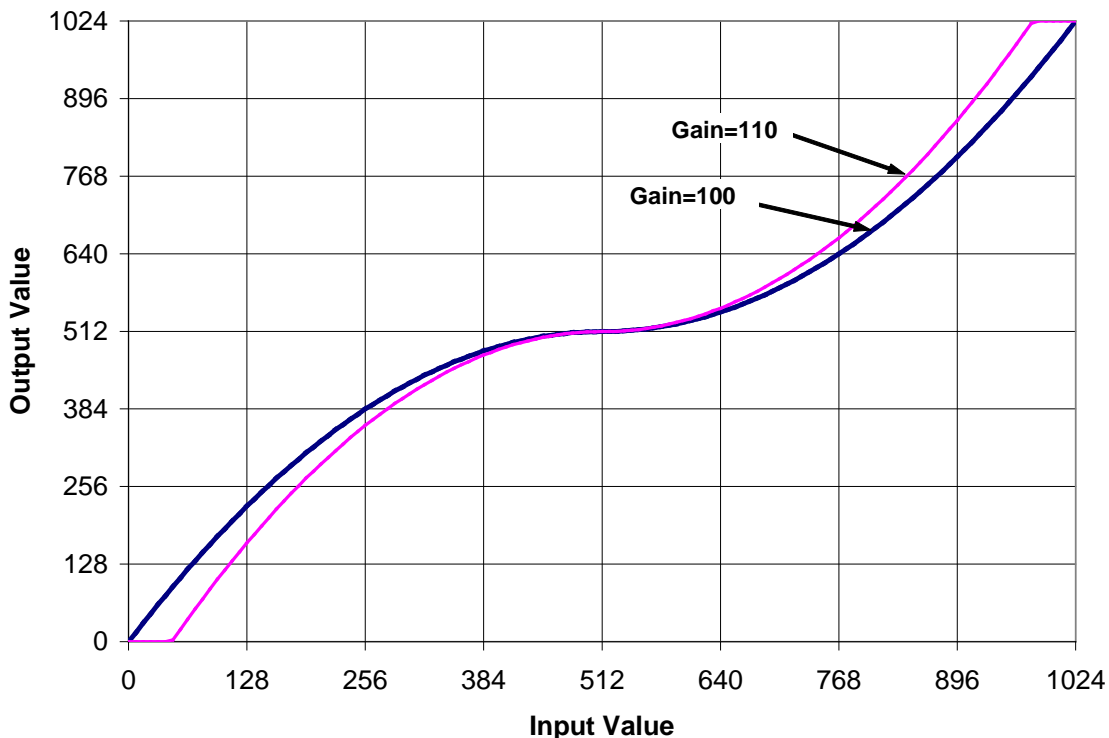
*unsigned int getClock(unsigned int offset)*

This function returns clock ticks relative to **offset**. Each tick represents 0.02 seconds (the effective clock rate is 50 hz.) The clock value wraps around to 0 after  $65535 * 0.02 / 60 \text{ min} = 21.8 \text{ min}$ . This function allows for “time stamping” and time-based controls.

*unsigned short parabolicScale(unsigned short inval, unsigned short gainval)*

This function rescales the supplied **inval** making the return value proportional to the square of the difference between **inval** and the idle value (idlevalue=512). This results in a non-linear scaling function that can be used to scale the joystick position to provide increased control fidelity near the idle position while still allowing the full range of output values. The return value is clipped to 0 and 1024. Figure 5.2 shows the relationship between input and output for two gain levels. The equation used for this operation is equivalent to:

$$return = idvalue + \left( \frac{gainval * (inval - idvalue)}{100} \right)^2 * \left( \frac{1}{fullscale - idvalue} \right)$$



**Figure 5.2. Example of Parabolic Scale Function**

### 5.3.5 Output and Debugging

The functions listed below provide a means of generating messages from within a BRAIN program that can be sent via the USB port to a terminal emulation program running on an external computer. The BSLUSB program described in Section 6.0 has the capability of displaying the output.

*void outHostsz(char\* pch)*

This function sends the null terminated string at **pch** out the BRAIN USB port. Communications settings for the port are 9600 baud, No Parity, 8 data bits, 1 stop bit.

*void itoa(int n, char s[])*

This function converts the binary integer **n** into the equivalent string in base ten notation, storing the result in the character array pointed to by the buffer **s**. A null character is appended to the result.

*char \*utoa(unsigned value, char \*digits, int base)*

This function converts the unsigned binary integer **value** into the equivalent string in base **base** notation, storing the result in the character array pointed to by **digits**. A null character is appended to the result. The return value is a pointer to **digits**.

*void revstr(char s[])*

This function reverses the order of the null-terminated string **s** in place.

*int strlen(char s[])*

This function returns the number of characters in the null-terminated string **s**

*short hexc2s(char c)*

This function converts an ASCII character representation of a hexadecimal digit **c** (0-9, A-F) into the corresponding number and returns that number as a short integer.

*void s2hexsz(short value, char \*where)*

This function converts a short **value** into a hexadecimal string representation and stores the result in the supplied character buffer **where**.

*char \*strcat(char \*dest, const char \*src)*

This function appends a copy of the string pointed to by **src** (including the terminating null character) to the end of the string pointed to by **dest**. The first character of **src** overwrites the null character at the end of **dest**. The return value is a pointer to **dest**.

### 5.3.6 Useful Constants

The header (.h) files associated with the BRAIN library define a few constants that may be helpful in developing programs for the BRAIN. A summary of the most useful constants and their meaning is provided in Table 5.1.

**Table 5.1. Constants**

<b>Name</b>	<b>Value</b>	<b>Description</b>
NSERVO	6	Number of servo output channels
NMOTOR	4	Number of motor output channels
RCIDLEVAL	512	Center or null value for scaling of inputs and outputs
RCMAXVAL	1024	Maximum value for scaling inputs and outputs
RCMINVAL	0	Minimum value for scaling inputs and outputs

## 6.0 BSLUSB Program

### 6.1 Overview

The BSLUSB program allows the BRAIN user processor to be programmed via its USB interface. Programming of the onboard flash memory is performed using the MSP430's embedded Bootstrap Loader (BSL) program. The BSLUSB program is a modified version of the program described in the Texas Instruments application report titled *Application of Bootstrap Loader in MSP430 with Flash Hardware and Software Proposal* (Document Number SLAA096D). The modifications were required to allow use of the BRAIN's USB hardware and to account for other circuit differences from the design presented in the referenced document. BSLUSB automatically downloads a patch to the MSP430 BSL program if it detects that the patch is required.

### 6.2 Use

If the IAR Workbench environment has been properly configured, which is done when the BRAIN support software is installed, there is no need to manually execute the BSLUSB program since you can execute it via the tools menu in the IAR Workbench. However, the description that follows may be useful in certain cases.

BSLUSB is a command-line tool (not Windows-based) that loads compiled programs, in TI-TXT format files, into the BRAIN system. The basic syntax of the program is:

*BSLUSB program.txt*

where *program.txt* is the compiled program file in TI-TXT format.

Run *BSLUSB -h* from a command prompt for a complete list of options. The most useful command option for troubleshooting is to include a *-d* option when executing the program, (*BSLUSB -d program.txt*). The *-d* option causes the error status of each command to be displayed; a nonzero error status indicates the step where the program failed.

BSLUSB assumes that there is only one BRAIN connected to the computer and attempts to download the code to the first BRAIN it detects.

Typical downloads take approximately 15 seconds to complete; larger programs will take more time.

The most common error is not having the USB cable connected to the BRAIN, or not waiting long enough after connecting the cable before attempting to download a program. Either of these situations will cause a "*Failed to enumerate port*" error message.

### 6.3 Terminal Mode

An option added to help with debugging BRAIN programs is a terminal mode invoked by

adding `-t` to the command, i.e., `BSLUSB -t program.txt`. This function should be pre-configured in the IAR Workbench Tools menu. After downloading a program to the BRAIN, the terminal option causes BSLUSB to enter a mode where messages received from the BRAIN are displayed in the BSLUSB window. The BRAIN should automatically reset/restart prior to entering terminal mode. Once in terminal mode, pressing “r” will cause the BRAIN to pause; pressing “R” after the BRAIN has paused will cause the BRAIN to restart. Press the ESC key to exit terminal mode.

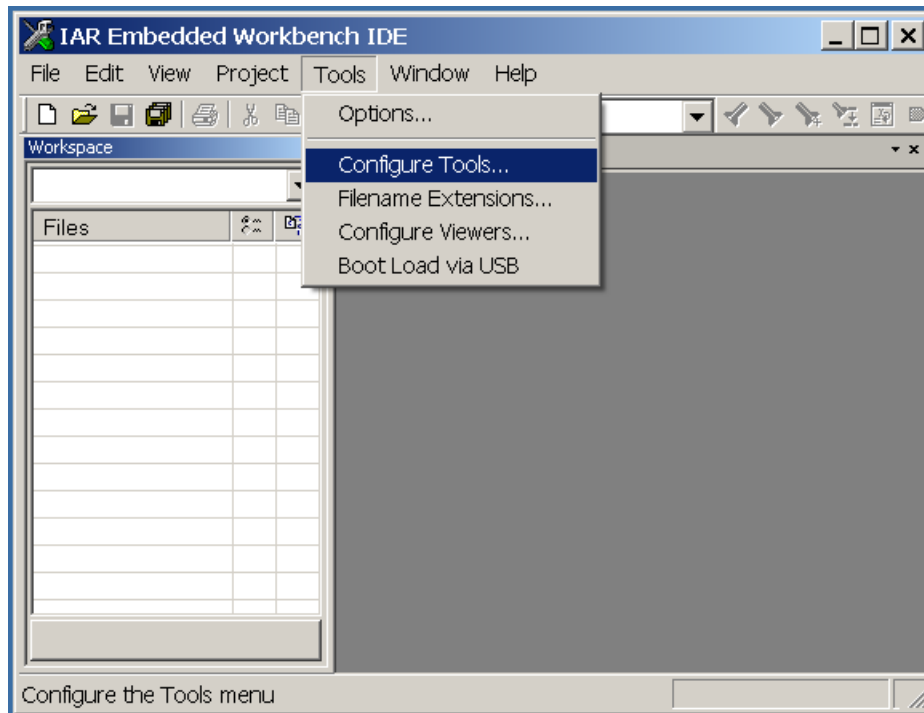
Note that if BSLUSB in terminal mode is being launched from the IAR Workbench, you should not use the “redirect to output window” option when configuring the IAR tools menu because this will cause the program to appear to hang since BSLUSB in terminal mode cannot exit without input from you, and the redirected output only appears after the program exits.

## 7.0 Configuring IAR Workbench Tools

The BRAIN installation program normally configures the IAR Workbench tools to allow the use of the BSLUSB program and the Wizard program from the tools menu. The sections that follow describe how to manually perform this configuration.

### 7.1 Configuring IAR Workbench to use BSLUSB

If the tools menu in the IAR workbench does not include a command called “Boot Load via USB,” as shown in Figure 7.1, the description below describes how to configure the tool.



**Figure 7.1. IAR Workbench Tools Menu**

The procedure for configuring the tool usage is:

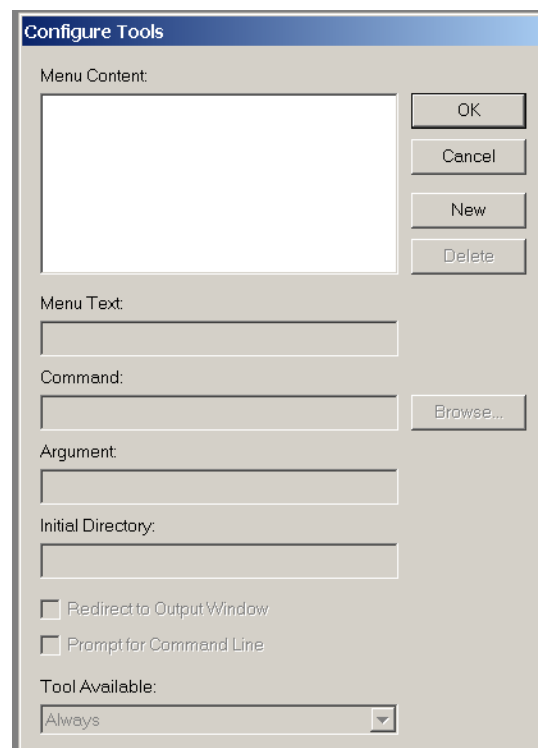
1. Select the Tools->Configure Tools menu as shown in Figure 7.1. The dialog box shown in Figure 7.2 will appear.
2. Click the “New” button. The dialog box shown in Figure 7.3 will appear.
3. Enter “Boot Load via USB” into the Menu Text input box.
4. Click the “Browse” button and locate the BSLUSB.exe program. It should be located in the “Program Files\Best\BRAIN\bin\” folder. Click the “Open” button in the file browse window after highlighting the BSLUSB.exe file.
5. Enter \$TARGET\_FNAME\$ into the Argument text input box.
6. Enter \$TARGET\_DIR\$ in the Initial Directory text input box.
7. Select the “redirect to output window” checkbox

8. Once the dialog box looks like that shown in Figure 7.4, click the “OK” button to complete the definition of the Tool item.
9. The tools menu should now have a “Boot Load via USB” entry as shown in Figure 7.1.

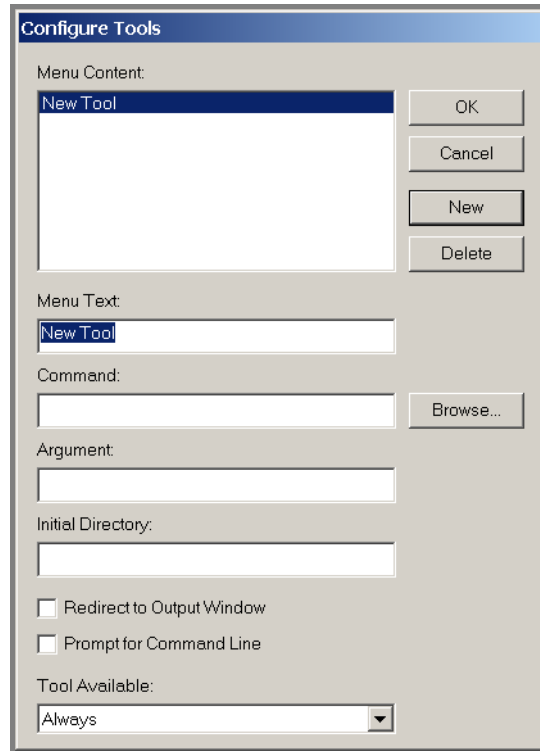
Note that the terminal option to the BSLUSB program can be added in a similar fashion with two minor modifications:

1. The argument should be: `-t $TARGET_FNAME$`
2. The “redirect to output window” checkbox should not be checked.

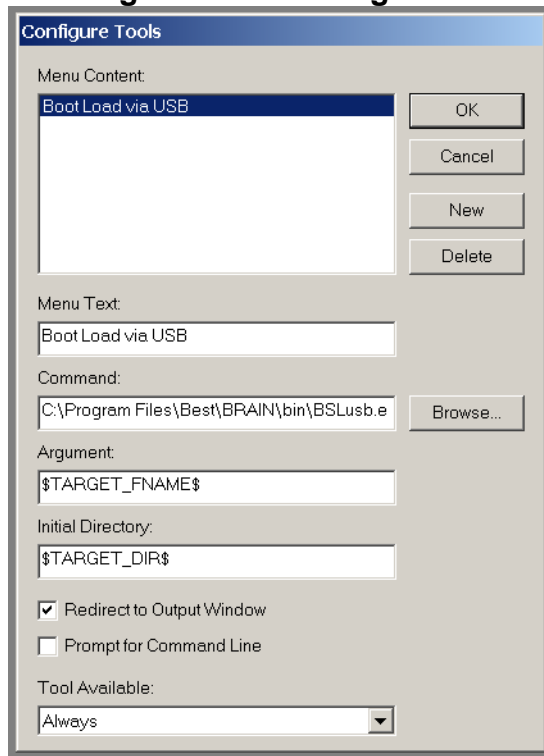
It is suggested that the terminal option be configured in addition to the standard download option since it will not always be used. The installer configures the tools menus this way.



**Figure 7.2. Configure Tools Dialog**



**Figure 7.3. Configure Tools Dialog After Selecting New**

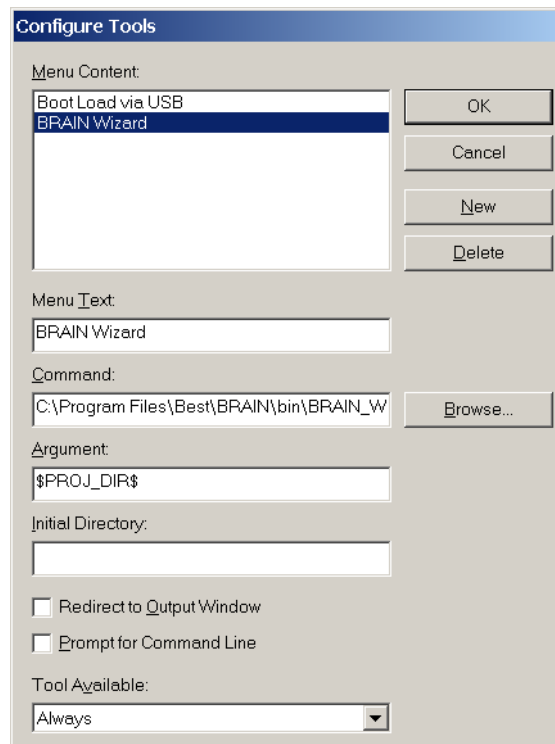


**Figure 7.4. Completed IAR Configure Tools Dialog**

## 7.2 Configuring IAR Workbench to Launch the Wizard

A similar procedure to that described in section 7.1 can be used to configure IAR Workbench to launch the BRAIN Wizard application.

1. Select the Tools->Configure Tools menu as shown in Figure 7.1. The dialog box shown in Figure 7.2 will appear.
2. Click the “New” button. The dialog box shown in Figure 7.3 will appear.
3. Enter “BRAIN Wizard” into the Menu Text input box.
4. Click the “Browse” button and locate the BRAIN\_Wizard.exe program. It should be located in the “Program Files\Best\BRAIN\bin\” folder. Click the “Open” button in the file browse window after highlighting the BRAIN\_Wizard.exe file.
5. Enter \$PROJ\_DIR\$ into the Argument text input box.
6. Once the dialog box looks like that shown in Figure 7.5, click the “OK” button to complete the definition of the Tool item.



**Figure 7.5. Completed IAR Configure Tools Dialog for Wizard Configuration**