W5D Users Manual

Revison 2.0
February 15, 2013
www.entactrobotics.com

Table of Contents

1.0 Initial Setup

1.1 Connecting the W5D to your Network or PC

Connection to a Router

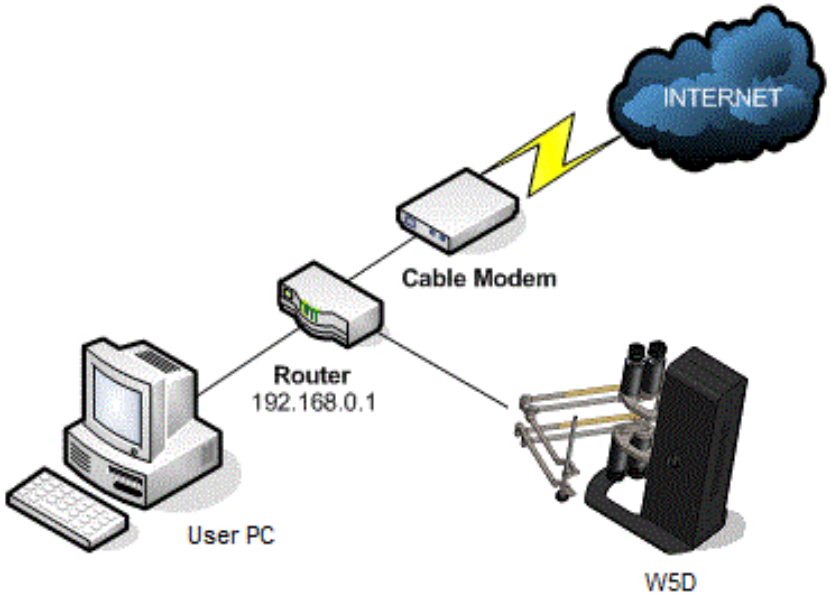
To initially connect with a W5D you will need a PC networked via a router. We recommend a dedicated router (which only connects to your W5D and PC, see image below) be used for initial setup. A dedicated router will allow a user to log onto the router status page and easily search out the W5D's IP address (in the DHCP client list). The W5D by default is configured as a **DHCP client**. It is possible to connect a W5D onto an office network, but it can be tricky to determine the IP address assigned to the W5D if access to the office network DHCP client list is unavailable.

DHCP Client

Simply put, the W5D requires a router (A DHCP Server) to provide it with an IP address when it first boots up. A DHCP Client requests an IP address from a DHCP Server.

For in-depth details see here (http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol)

Your PC is not (normally) set up as a DHCP Server, so connecting the W5D into your PC directly will not network the W5D properly. Below is a connection diagram showing how the W5D should be networked initially.



Note: Our motivation for shipping the W5D devices configured as DHCP clients stems from the variability in the setup of our users networks. Also, W5D devices are typically used in pairs (for left and right handed operation); DHCP guarantees there will not be an IP address conflict in this situation.

Powering up the W5D

Once the W5D is connected to the network, plug in the power cable and power up the system. On power up the W5D will take some initial time to boot its operating system and firmware. You will notice an LED located on the side panel of the W5D, this LED is used to indicate the W5D's state of operation (see here for details). Below is the power up sequence.

- 1) Boot-up: Wait approximately 10-15 seconds (Panel LED will be off)
- 2) W5D obtains its IP Address on the Network (or cannot, either way this step completes)
- 3) W5D firmware starts running (Panel LED begins blinking)

When the system boot-up is complete the W5D starts in the Disabled State, meaning all of its motor drives are deactivated.

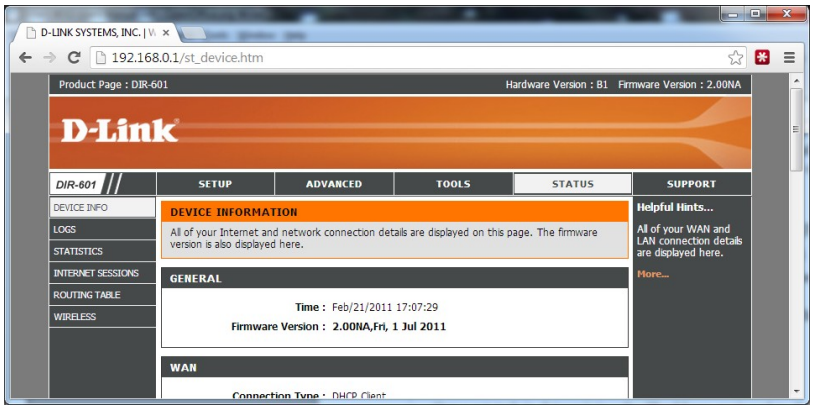
Determining the IP address

Our next task is determining the IP address assigned to the W5D by the router; there are two routes which can be used.

1) Running the w5d_command application to search the network for the W5D.
2) Logging onto the routers firmware pages and checking the “attached clients” list.

For **option 1)** you will need to have already installed the W5D's driver (instructions are here) and have a copy of the **w5d_command.exe** executable. When the w5d command console is run, its first task is to auto search the network for any w5d devices it can find. It will then list the IP address of your attached W5D. More detail into using the w5d_command console application can be found here.

For **option 2)** you first log onto your routers firmware pages using a web browser. If you know the gateway IP address of your router (typically on routers back sticker, and commonly “192.168.0.1” or “192.168.1.1”) enter this into a web browser and search for your routers list of DHCP Client devices. For our testing d-link router, this is on its status page shown below.



D-link Status Page (example router interface)

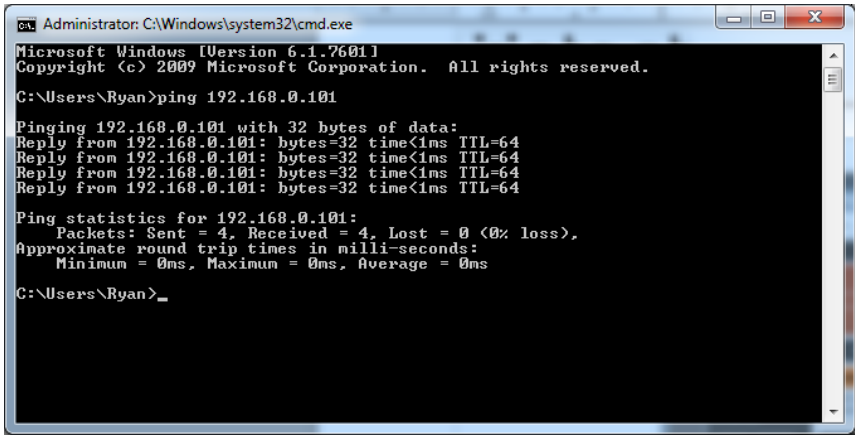
Find the the list of attached DHCP Clients (ours is scrolled to the bottom) and make note of the W5D's IP address. Note, in the client list below there are only two devices connected to the router; the W5D at 192.168.0.101, and the user PC (called Media) at 192.168.0.100.

LAN COMPUTERS

IP Address	Name(If any)	MAC
192.168.0.100	Media	54:04:a6:37:2a:d3
192.168.0.101	Entact W5D	00:01:02:03:04:05

D-link DHCP Client List

Make note of the W5D's IP address and open a command prompt window (cmd.exe). Enter the ping command followed by the IP address of the W5D. In our case we enter **ping 192.168.0.101,** the results shown below indicate the W5D is connected and communicating properly.



cmd.exe (ping command)

With the W5D networked correctly, you are now ready to install the W5D driver (here) and begin running demo software (here).

1.2 Installing the W5D API Driver

The following set of instructions is intended for manual installation of the W5D API driver in windows XP, 7 and 8. At present there is no automatic installer, so ensure you follow these steps to properly install the API.

A visual studio 2010 solution is also available with the API's source code, which can be re-built. This solution will be briefly outlined.

Creating the Directory Structure

Create the following three directories under your C: drive.

C:\EntactAPI\bin32

C:\EntactAPI\lib32

C:\EntactAPI\include

Copy over API files

Unpack the w5dAPI zip file and copy over the three driver files as follows.

C:\EntactAPI\bin32**w5dAPI.dll**

C:\EntactAPI\lib32**w5dAPI.lib**

C:\EntactAPI\include**w5dAPI.h**

Create a PATH environment variable entry

For detailed instructions on creating a path entry see the following link;

http://www.computerhope.com/issues/ch000549.htm

In the **PATH** entry prepend (at the beginning) the following directory, make sure to add the semi-colon at the end. Applications which call the W5D dll file use this path to find the driver.

C:\EntactAPI\bin32;

The W5D API is now ready for linking in user applications. The W5D Console Application is also ready to be run.

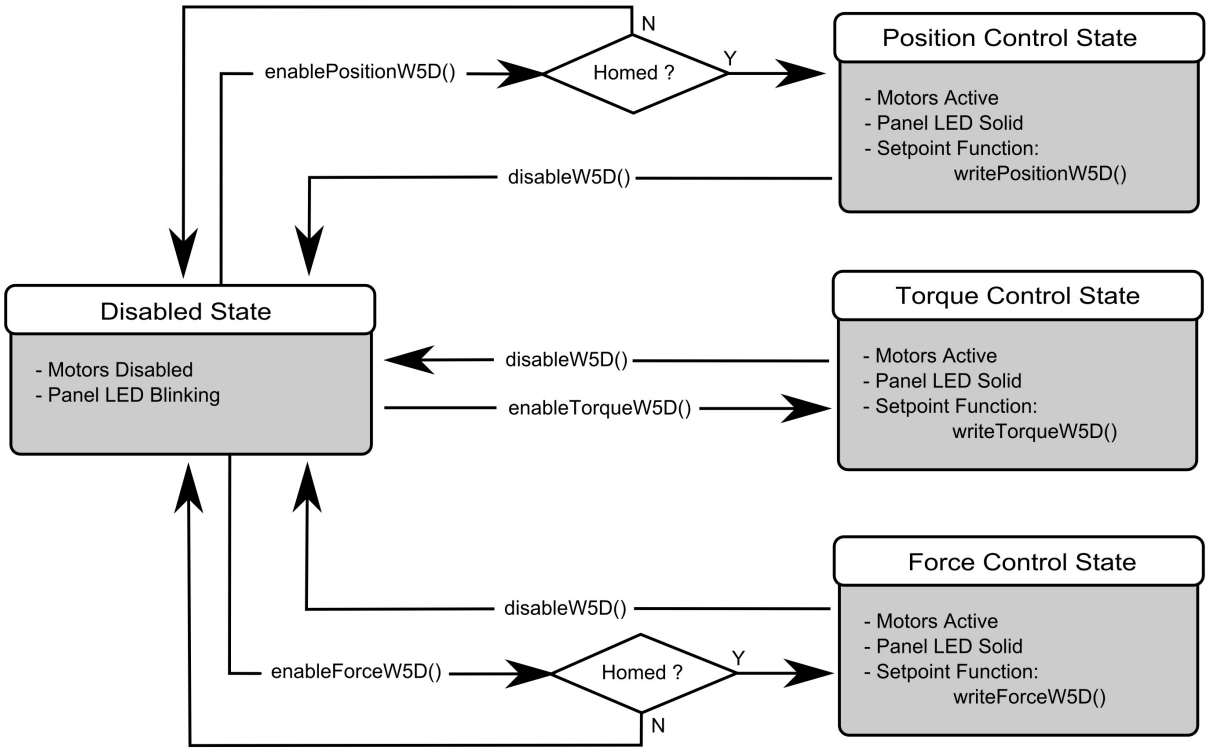
W5D API Visual Studio 2010 Solution

w5dAPI.sln is provided for users wishing to make modifications to the API, or better understand its method of operation. See the API Overview section for more details on the w5d API. On successful build of the visual studio solution the new driver, library and header files are copied over to the C drive EntactAPI directories.

1.3 Understanding the W5D States of Operation

Power Up

On power up the W5D will take time to boot its operating system and firmware; you will notice the panel LED will take 5 to 10 seconds to light up. When the system boot-up is complete the W5D starts in the Disabled State. The panel LED will begin to blink, the firmware loads, and the W5D is ready for use. Below is a state diagram showing each of the W5D's states of operation, along with functions which allow transition between each state.



W5D States of Operation

Disabled State

This is the default state for the W5D, and the first state the device enters on system power-up. In this state the panel LED blinks signalling the system is inactive. The power amplifiers used to drive the W5D's motors are disabled.

State entry:

To enter into the disabled state call the disableW5D() function.

Homing behaviour:

A homeW5D() command may be issued in any of the W5D's states. Immediately after the W5D will disable itself (if not already disabled). This protects the system from a change in calibration while the amplifiers are active.

It is recommended that homing be performed exclusively in the disabled state.

Set-point behaviour:

Setpoint commands in the disabled state are stored for future use. You may, for example, want to command an initial position setpoint while the system is disabled such that the handle will move to a desired initial position when enablePositionControlW5D() is called.

Caveat: If the W5D transitions into this state from an active state (Force, Position or Torque Control) all system set-points are cleared to their defaults. For Force and Torque this is a command of zero, for Position this is the home position. You will need to re-command your desired initial setpoint again before re-enabling an active state.

Force Control State

In the force control state the W5D is capable of producing a world space force and torque onto its handle. These forces and torques are referenced to the world space coordinate system (see here). In this state the panel LED will be solid ON indicating the motors are active and can produce device movement.

Before entering into this state it is mandatory for the W5D to be calibrated using the homeW5D() function.

State entry:

To enter into the force control state call the enableForceControlW5D() function. Before entry is allowed the W5D must first be calibrated using homeW5D() and currently in the disabled state. "Hot-swap" transitions from active states have been disabled.

Note: if you want to make a transition from one active state to another (ex, from Force Control to Position Control) first disable the device then make the the transition to position control.

Set-point behaviour:

To command a world space force/torque onto the handle call the writeForceW5D() function. Calling other setpoint functions like writePositionW5D() or writeTorqueW5D() will not be blocked, and won't effect the force setpoint. By default, when the W5D powers up, the force setpoint is zero. After a new setpoint is commanded, the W5D will produce this force in Force Control. The W5D will continue to try and produce a force matching the most recently received force setpoint.

Note: If the W5D disables, or you disable the W5D your force setpoint will be cleared to zero.

Position Control State

In position control the W5D handle is capable of tracking a world space position and orientation trajectory. This trajectory is referenced to the world space coordinate system (see here). In this state the panel LED will solid ON indicating the motors are active and can produce device movement.

Before entering into this state it is mandatory for the W5D to be calibrated using the homeW5D() function.

State entry:

To enter into the position control state call the enablePositionControlW5D() function. Before entry is allowed the W5D must first be calibrated using homeW5D() and currently in the disabled state. "Hot-swap" transitions from active states have been disabled.

Set-point behaviour:

To command a world space positions onto the handle call the writePositionW5D() function. Calling other setpoint functions like writeForceW5D() or writeTorqueW5D() will not be blocked, and won't effect the force setpoint. After a new setpoint is commanded, the W5D will continue to hold position until a different position setpoint is received.

Note: If the W5D disables, or you disable the W5D your position setpoint will be cleared to the home position. You will then need to re-enstate your desired position with a writePositionW5D() call.

Default Position:

When this W5D is first powered up it is preloaded with the home position as its setpoint. If you enable position control without updating the position setpoint beforehand, the W5D will assume you want it to hold position at home.

W5D Home position values

Position	
X	0
Y	0
Z	0

Orientation Matrix		
1	0	0
0	1	0
0	0	1

If you want to have the W5D hold a different position call the writePositionW5D() function before entering Position Control state. Note: it is common to want to hold the W5D at its current position upon enabling position control. To do this, first read the current position by calling getPositionW5D() and then pass the position into writePositionW5D(), then finally transition into position control.

Torque Control State

In the torque control state the W5D is capable of producing joint space torques on each of its motors. These torques follow the motor assignment and direction convention outlined here. In this state the panel LED will solid ON indicating the motors are active and can produce device movement.

State entry:

To enter into the position control state call the enableTorqueControlW5D() function. Before entry is allowed the W5D must currently be in the disabled state. "Hot-swap" transitions from active states have been disabled.

Set-point behaviour:

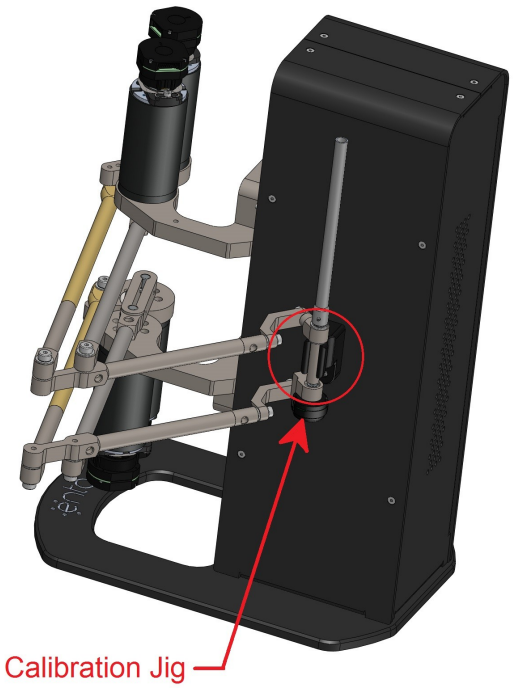
To command a joint space torque command call the writeTorqueW5D() function. Calling other setpoint functions like writeForceW5D() or writePositionW5D() will not be blocked, and won't effect the force setpoint. After a new setpoint is commanded, the W5D will produce the array of torques on its motors until a different torque setpoint is received.

Note: If the W5D disables, or you disable the W5D your torque setpoint will be zeroed.

1.4 World Coordinate Frame, Calibration and Home Position

Calibration (Homing) of the W5D

Calibration of the W5D is required before world positions and world forces/torques can be produced. The W5Ds sensors (encoders) provide only relative positions. The arms of the W5D must be put in a calibration “reference” position before absolute position can be known. Below is an image of a W5D with its calibration jig highlighted. Place the handle of the W5D into this jig (The handle will snap into place) as shown in the image.



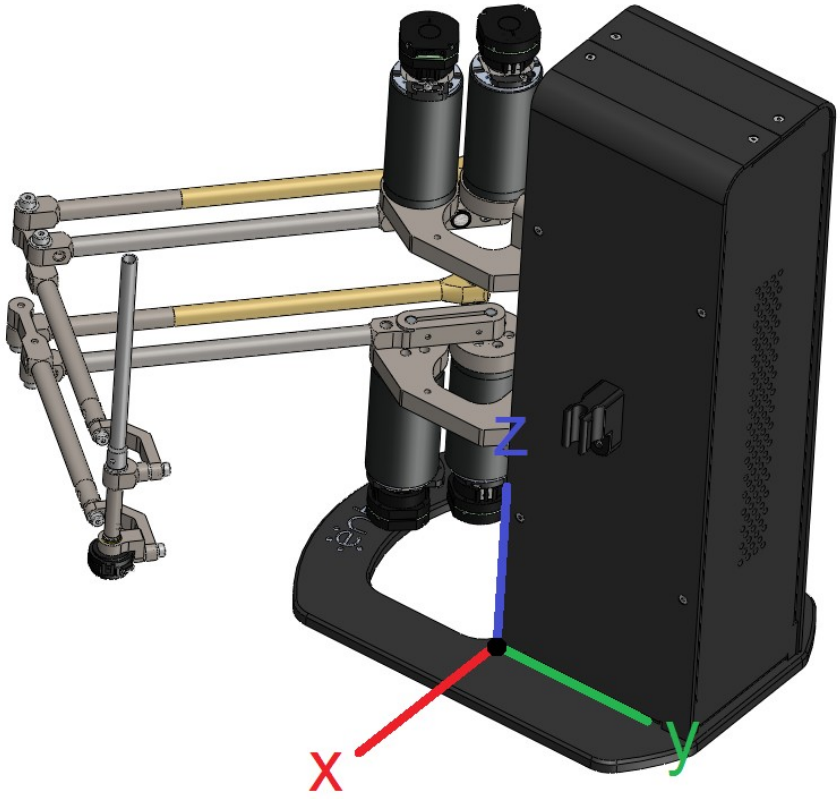
With the handle held in this position you can now call the homeW5D() function, and the W5D will have proper calibration. Note, calibration is only required once after power-up.

It is recommended to perform homing with the W5D in the disabled state (panel led flashing), although it is possible to calibrate in active states (Force, Torque, Position Control). If you call the homing function in an active state the W5D will disable itself after the homing is complete.

Call isHomedW5D() to check if the W5D is already homed.

World Coordinate System (Home Orientation)

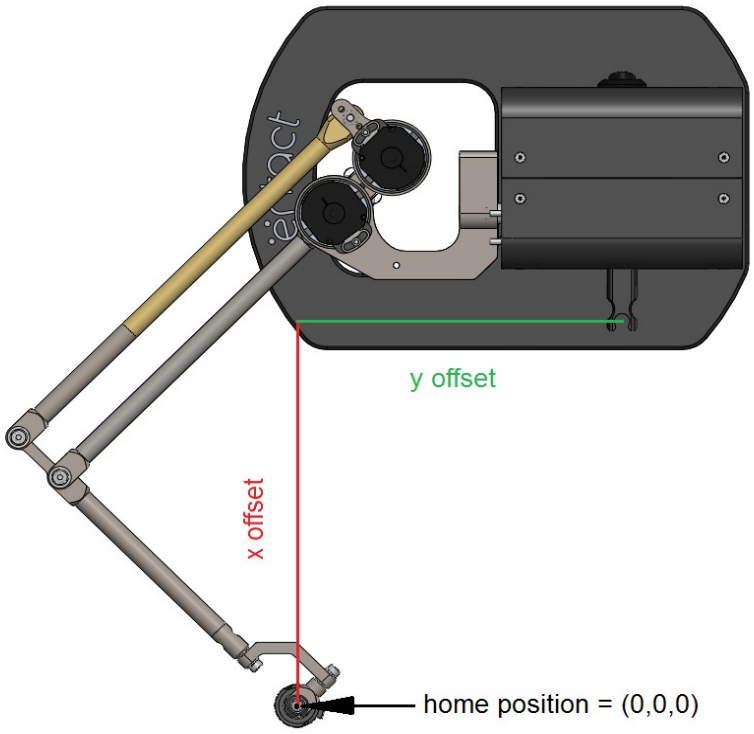
Below is an image showing the direction vectors for the W5D's world frame. The frame origin is not located at the (0,0,0) position, but instead located to best illustrate the directionality of the X, Y and Z axes. A left handed W5D is shown, if your W5D is a mirror image of this system (right handed W5D) note that the direction of these axis do not change. Z will always point vertically, X will always point out toward the user , and Y will always be from a users left hand toward a users right hand.



World coordinate directionality for W5D (Left Handed system shown)

Home Position

The home position is where the W5D will report a position of zero for X, Y and Z along with the identity matrix for orientation. The image below shows a W5D in its home position viewed from above. A positional offset between the home and calibration position exists. This offset is in both X and Y, but not in Z. For the left handed W5D (shown below) the y offset is in the negative y direction, and for a right handed W5D in the positive y direction.

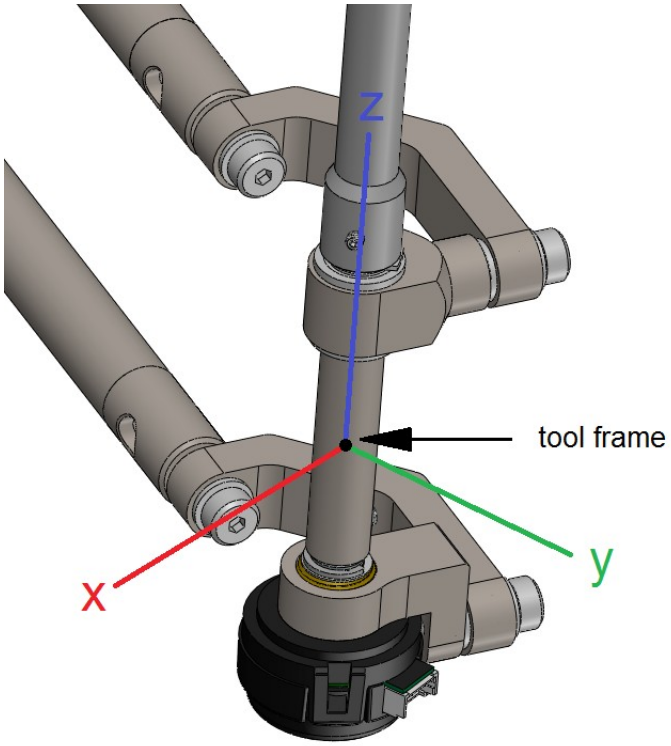


Home Position from above (Left Handed system)

X offset	190.7 mm
Y offset	- 155.0 mm (LH W5D) 155.0 mm (RH W5D)

Tool (Handle) Frame

The tool position is located at the centre of the W5D handle. An image below shows the handle along with the tool frame attached.



W5D Handle and Tool Frame

World Forces and Torques

The tool frame centre point is where world forces and torques are applied. Forces are along each world frame axis, and the torques are about each world frame axis.

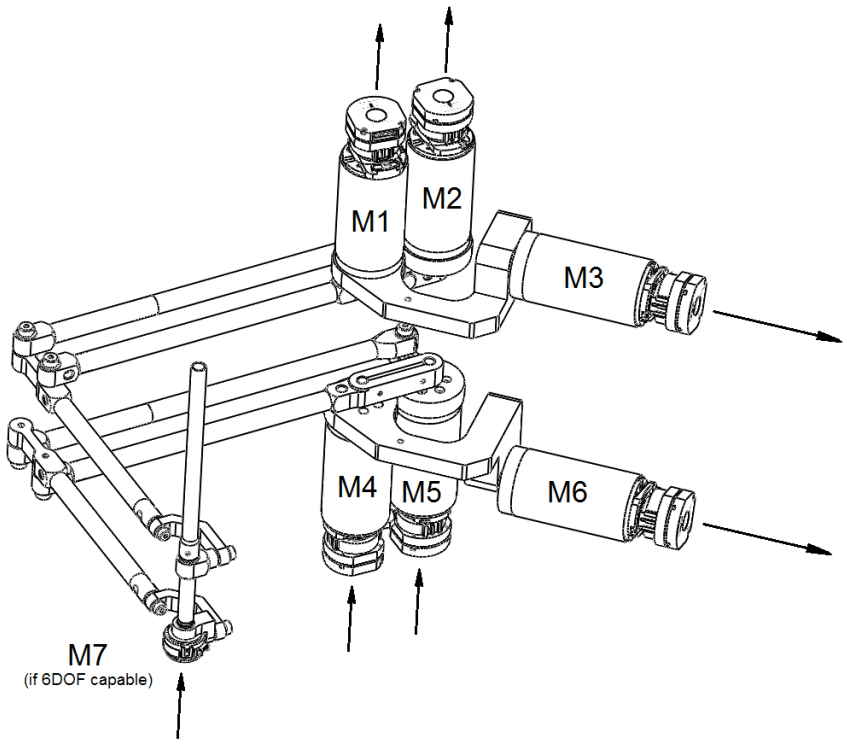
> For a 5.D.O.F. W5D system only 5 out of the possible 6 world force/torques can be produced. In the home position the Z direction torque cannot be produced. Depending on where the handle is moved different torques will become possible, and others (aligned with the tool frame z axis) will not be possible.

World Positions and Orientation

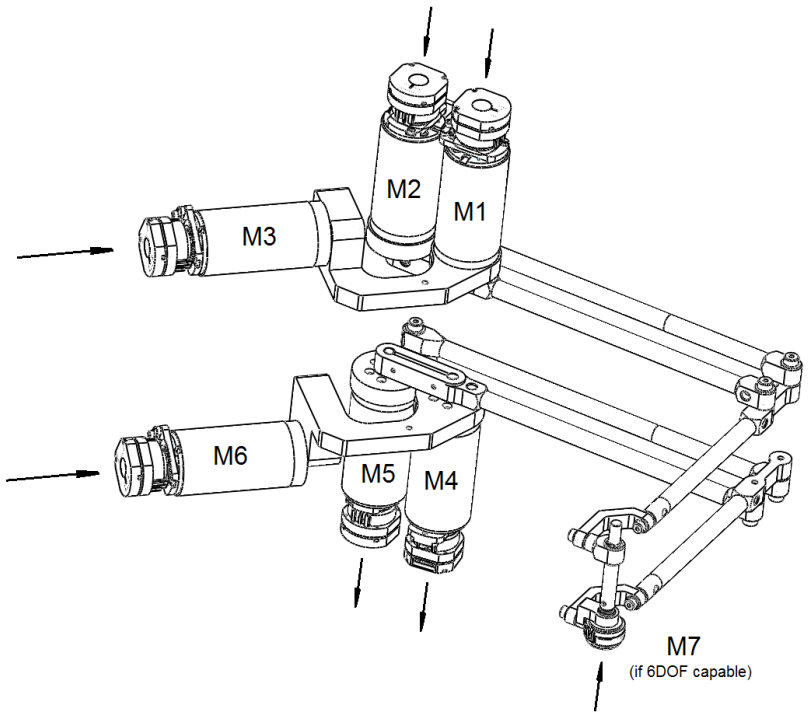
The positions reported by the device are the position of the tool frame point relative to the home position (shown above) expressed in the world coordinate frame. Orientations are the orientation of the tool frame (attached to the handle) relative to the world frame (attached to the W5D base). Orientation is expressed in a 9 element rotation matrix.

1.5 Motor and Joint Numbering Convention

With joint torques and joint positions each joint is referenced using a numbering system. The convention for the W5D is shown below. Arrows indicate the direction of positive rotation and torque using the **right hand rule**. The image below shows a left handed W5D system, the convention for a right handed system is on the following page.



Motor Labeling Convention (Left Handed W5D)



Motor Labeling Convention (Right Handed W5D)

Joint angles are relative, when the system powers up all angles are initially zero. Once the system is homed using the calibration jig, the new zero position for all joints is at the same place as the world home position.

1.6 Using the W5D Command Console Application

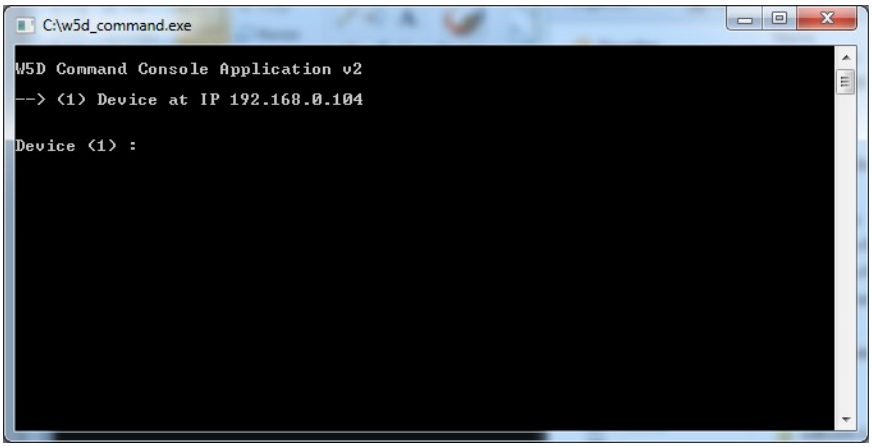
w5d_command.exe

The W5D command console application provides a set of console commands for interacting with W5D devices connected on the network. The console application makes use of the w5d API which should be installed before running the application. Prerequisites for issuing commands are as follows;

- 1) The W5D is networked properly and has a reachable IP address (here). Note: the command console is a good test to see if the device has a reachable IP. It's worth while to run this application to test what IP has been assigned to the W5D.
- 2) The w5d API is installed on the user PC (here)

Application Startup

Run the executable **w5d_command.exe,** and a Windows console terminal will load. The application will first scan the network for attached devices and report the IP addresses of each attached device. The user will then be prompted to select which device (out of the list) to connect with. If only one device is present this device will be connected with automatically. The window below shows the w5d command terminal finding a single attached W5D device at 192.168.0.104.



w5d_command console window

Using the Command Terminal

Below is a list of some common and useful commands provided with he command terminal. For a list of all possible commands simply hit enter at the command prompt. Entering an invalid command will have the same result. Commands are listed below in brackets [ ].

[home]

Calibrates the W5D. This is useful when the system is first powered up at the start of a development session. Once the device has been calibrated you will not be required to calibrate again before power-down.

[disable]

Sets the W5D into a disabled state. It is not uncommon for an application to accidentally leave the W5D in an enabled state because of an unforeseen shutdown. You can use the command console and this command to alleviate the problem.

[display_joints]

Outputs a refreshing display of the W5D joint (encoder) angles for roughly 10 seconds. This can be useful to quickly determine the current joint angles, and debug faulty encoders.

[display_position]

Outputs a refreshing display of the W5D task position for roughly 5 seconds. This can be useful to quickly determine the current position, and debug faulty kinematic modifications. You can also use the W5D as a crude measuring device.

2.0 W5D Customization

2.1 Connecting to a W5D using PUTTY

Prerequisites

- You must first have your W5D networked successfully. See here for instructions on how to network your W5D properly.
- You must have knowledge of the IP address assigned to your W5D.

Installing Putty

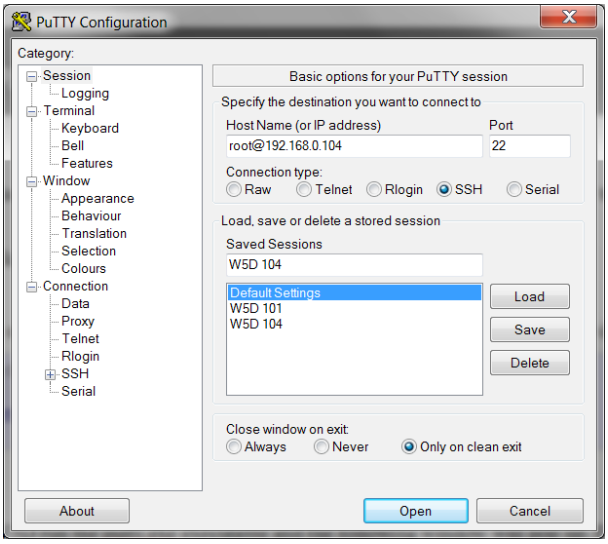
Putty is a standalone executable (putty.exe) which creates an SSH connection with your W5D device.

Download putty at the following web location, choose x86 for windows version (for windows users)

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

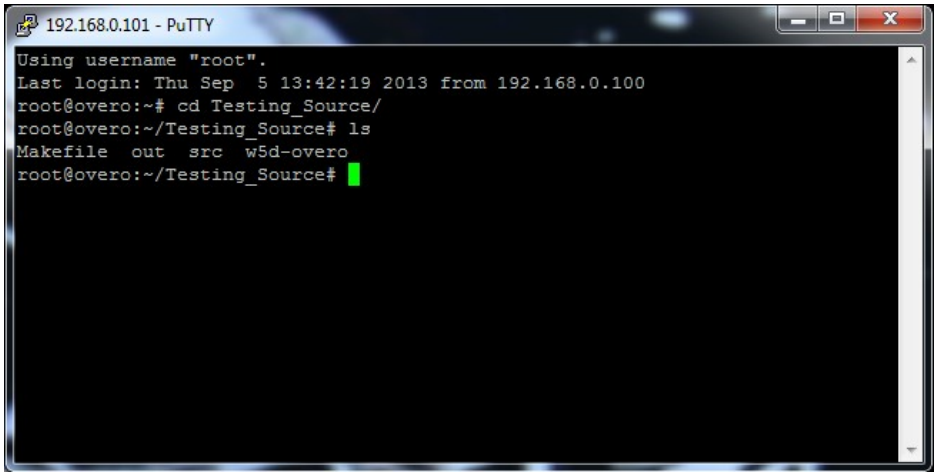
Running Putty

To connect to your W5D run the putty.exe executable and the following window will pop up.



- In the Host Name (or IP address) field fill in “root@###.###.###.###” for the IP address of your W5D. In this case we're connecting to a W5D at 192.168.0.104.
- In the Port field fill in 22.

Click open and putty will attempt to make a connection with your W5D device. If the connection is successful a terminal window will open giving you a linux command prompt.



At this point you will have a linux prompt, which will allow you to start issuing commands to the W5D's embedded system.

- If connection to the W5D was unsuccessful you most likely have entered in an improper IP address for the W5D. See the first section here for instructions on how to connect and determine and test your W5D's IP address.
- Once you have the Linux terminal open you can do several configurations on the W5D including;
  - Switching from Dynamic to Static IP addressing (here)
  - Modifying and Re-building the W5D's firmware (here)
  - Diagnostic testing on the W5D's subsystems (here)

2.2 Switching from Dynamic to Static IP Addressing

Step 1:

First a terminal connection must be made with the W5D. Open an SSH client, like putty.exe (make sure the W5D is connected to the network and booted (LED is blinking or solid ON). Instructions on how to connect using putty are in the previous section (here).

Step 2:

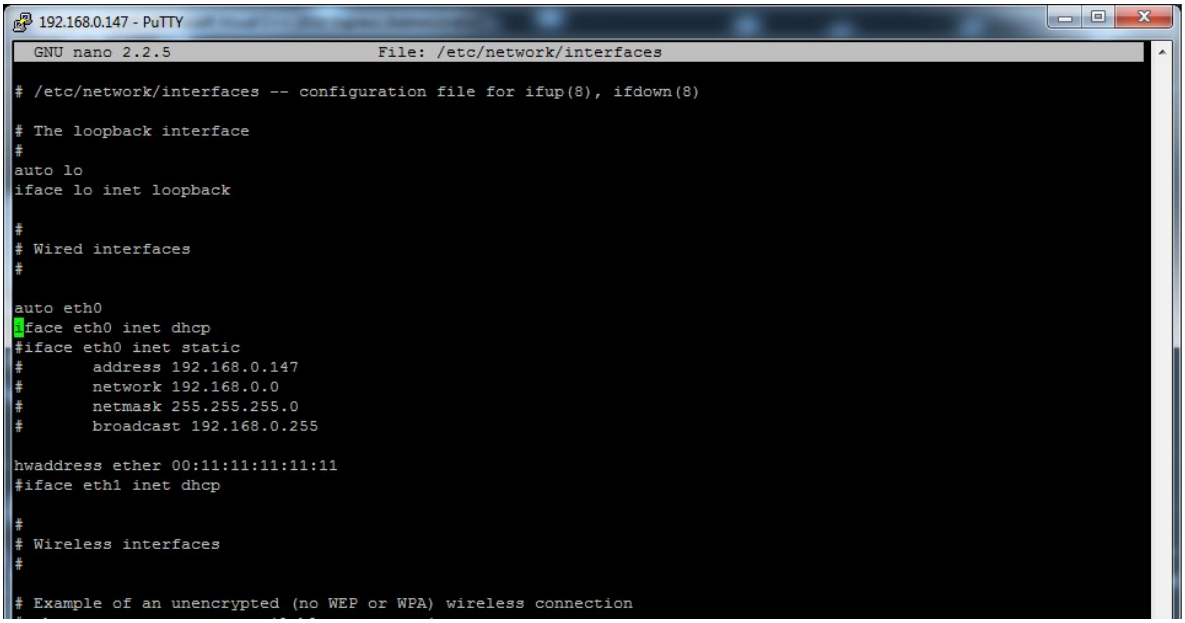
Set the Host Name to root@192.168.XXX.XXX (set this to the devices IP address) and connect using your SSH client program. You should now have a linux prompt which you can use to configure the W5D in the SSH window.

Step 3:

At the linux prompt type the command: **nano /etc/network/interfaces**

Then click enter. A file will be opened where you can configure the IP address settings of the W5D. The image below shows a putty window with the interfaces file opened. In this file the system is configured for DHCP.

Note: nano is a lightweight text editor which runs native on linux, and is available onboard the W5D. Use nano for editing text files. See the following link for a nano tutorial: http://www.nano-editor.org/dist/v2.2/nano.html



Interfaces file (ethernet configured as DHCP)

Step 4:

Decide if you want to configure the W5D for Static IP addressing or Dynamic. Both of these options are present in the interfaces file. You will also notice lines start with a command or a "#" character. The "#" character is used for de-activating the command that follows on the same line.

The section of the interfaces file which pertains to the W5D's IP address is in and around the line **auto eth0**. The eth0 module is what controls the ethernet communications port on the W5D.

It is important for the line containing **auto eth0** to always be active (or never proceeding with a #). This command is what activates the ethernet port. Past situations where this line was accidentally commented caused a total loss of communication with the W5D. The lines proceeding auto eth0 will configure the module as either static or dynamic (depending on which are commented out).

For reference, here is an example of the interfaces file

http://www.cyberciti.biz/faq/setting-up-an-network-interfaces-file/

Static IP Addressing

Comment out the line **iface eth0 inet dhcp,** and un-comment the following five lines (pertaining to the static IP settings). The eth0 section should then look similar to below.

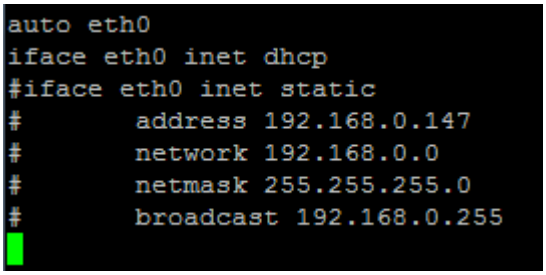
```
auto eth0
#iface eth0 inet dhcp
iface eth0 inet static
        address 192.168.0.147
        network 192.168.0.0
        netmask 255.255.255.0
        broadcast 192.168.0.255

hwaddress ether 00:11:11:11:11:11
```

- Address, sets the static IP address of the W5D
- network, netmask and broadcast should match your networks settings. Typically these do not need to be changed as most networks are set up with the "255.255.255.0" subnet mask.
- The hwaddress ether line sets the MAC address of the system, this can also be configured if desired.

Dynamic IP Addressing

Make sure the line **iface eth0 inet dhcp** is NOT commented out. Comment the following five lines (pertaining to the static IP settings). The eth0 section should then look similar to below.



Step 5:

Once you have made the changes in the interfaces file keypress Ctrl-O then hit Enter, this saves the changes. Then keypress Ctrl-X, this exits the file.

Type the **reboot** command into the linux prompt and the W5D will shutdown and restart with the new IP settings in effect.

2.3 Modifying and Rebuilding the W5D Firmware

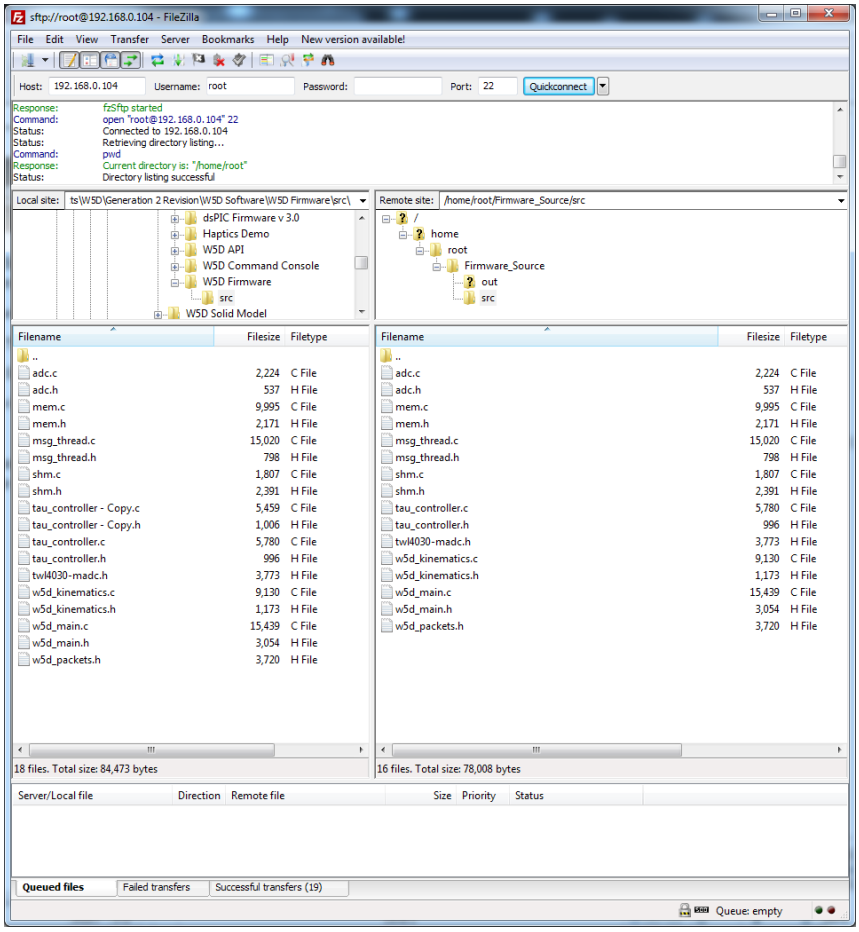
The w5d runs on open source firmware. You can find the firmware source files on the w5d device. Modification of the firmware will require some expertise in both the linux operating system and the c programming language.

First off, there are three pieces of software we typically use when doing firmware modification.

1) Putty, to create a terminal connection with the W5D device.
2) Filezilla, to create an FTP connection with the W5D for copying over new source code files.
3) A C text editor of your choice. We normally use the visual studio environment.

Step 1:

Connect the W5D to the network and make a terminal connection using a putty; instructions for doing this can be found here. Next connect to the W5D using Filezilla. An example connection using filezilla is shown below.



Filezilla connection with a W5D

For the filezilla connection set the port value to 22, set the username to root and set the host to the IP address of the W5D.

Once the connecton is established you can copy over all of the firmware files from the W5D into a local directory of your choice. The W5D's firmware is located on the following path.

> **/home/root/Firmware_Source**

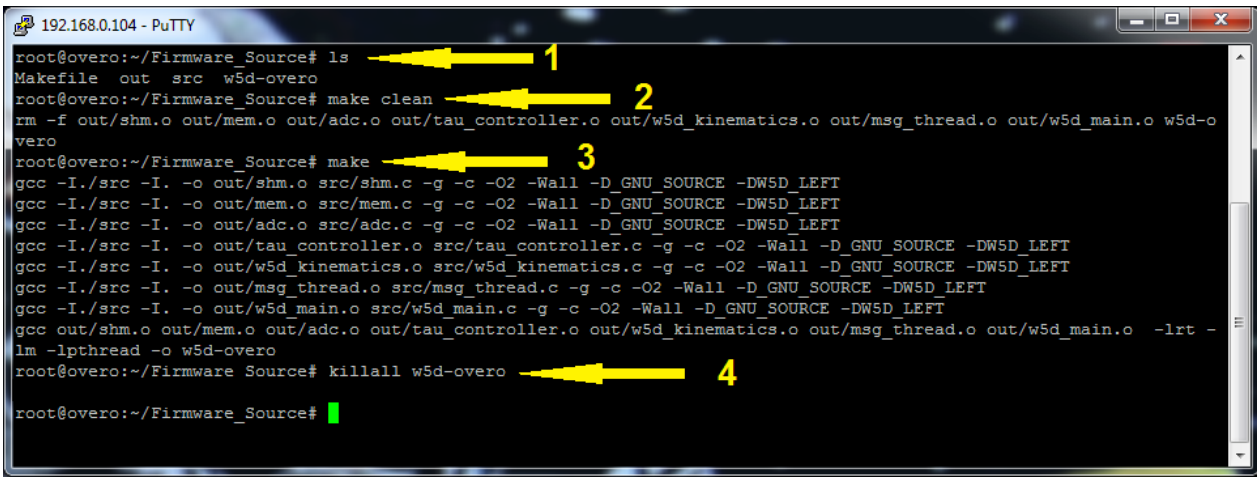
Step 2:

Open the firmware source files using your text editor. Search through the code and find the place where you want to make modifications. Once you're ready, copy the file back over to the W5D using Filezilla. A brief description of key firmware modules are presented below.

- **w5d_main**: Entry point for the firmware, where the real-time loop of the w5d is executed. All other modules are initialized in this module.
- **msg_handler**: A second thread is created in this module for handling UDP communication. The communication protocol is handled using this module. w5d_packets.h is the definition for the communication protocol.
- **tau_controller**: Handles interaction with the onboard torque contollers.
- **w5d_kinematics**: Contains all the source code for kinematic calculations including the forward, force and velocity kinematics.
- **shm**: Stores global device data which is shared between the message and real-time threads. All of the W5D's state and data parameters are stored here.

Step 3:

Once the modified files have been copied back to the W5D, it's time to re-build the firmware source. This is handled on-board the W5D and can be executed using the putty terminal window. Change directories to the Firmware_Source directory (indicated above).



Firmware build using putty terminal

Once at the Firmware_Source directory you can proceed to re-make the firmware using make.

Command 1, is optional, a linux **ls** command has been executed to show the different files which are located directly under the Firmware_Source folder. *Makefile*, is the make file used by make to build the firmware. *Src* is the directory containing all of the source code files. *Out* is the directory containing all of the built object files. *W5d-overo* is the final executable of the w5d firmware.

Command 2, a **make clean** command is issued to clean all of the previously built object files in the *out* directory. This step is also optional.

Command 3, a **make** command is issued to re-built all of the firmware from source using the make file. Once this command is issued you will see each firmware module build one by one. If this command is successful (i,e. There are no errors in the source code) you will see the gcc out line with along string of each module and finally w5d-overo. Use the above window as a reference of a successful build.

Command 4, a kill command **killall w5d-overo** is issued to kill the currently running w5d firmware process (called w5d-overo) and replace it with our newly built firmware.

3.0 W5D API Reference

3.1 API Overview

The W5D API provides a set of "C" functions for discovering, enabling and commanding attached W5D Haptic Devices. The API is implemented as a DLL (dynamically linked library) file which must be installed on your host PC.

To install the API on a windows PC see the following section for instructions.

For users of visual studio 2010 a solution is provided which allows users to trace through, modify and re-build the API.

Note: The API is supported for Windows only. Linux versions do exist but are developed and used at the customers discretion.

The **W5D API** consists of the following files;

w5dAPI.h	a C/C++ header file outlining all of the member functions and definitions. This file is intended to be included in a users C/C++ haptics application.
w5dAPI.cpp	A C/C++ source file with all the private implementations of each member function. This file is not needed unless a user is interested in tracing through the API member function calls.
w5dAPI.lib	A static library which is linked with user projects using the w5dAPI
w5dAPI.dll	A dynamic link library loaded with your projects final executable at the run-time executable.

3.2 API Functions Summary

API Initialization

openW5D()	Search for all attached W5D devices
connectDeviceW5D()	Search for a W5D at known (fixed) IP address
closeW5D()	Un-initialize the API

State Change

disableW5D()	Disable the W5D (drives deactivated)
enableForceControlW5D()	Enable the W5D for Force Control
enableTorqueControlW5D()	Enable the W5D for Torque Control
enablePositionControlW5D()	Enable the W5D for Position Control
homeW5D()	Calibrate the W5D
setFrequencyW5D()	Set the Real-time Frequency on-board the W5D
setDampingW5D()	Set the Internal Damping Field Haptic Effect
setWatchdogW5D()	Set the Watchdog Timer

Device Query

getIPAddressW5D()	Request the IP Address of a W5D
getStateW5D()	Determine the Running State of a W5D
isHomedW5D()	Determine if a W5D is Calibrated
getFrequencyW5D()	Determine the Real-time Frequency of a W5D
getJointsW5D()	Request a W5D's Joint (encoder) Positions
getPositionW5D()	Request a W5D's World Position/Orientation/Velocity

Set-point Commands

writeForceW5D()	Command a World Force/Torque vector to the W5D
writeTorqueW5D()	Command Motor Torques to the W5D
writePositionW5D()	Command a World Position/Orientation to the W5D

openW5D()

The openW5D() function is the first function a user program should call within the w5dAPI (connectDeviceW5D() is the exception in one case). The open function broadcasts onto the network in search of all attached W5D devices. It fills the handles array with devices and returns the overall number of devices discovered on the network.

```
W5DAPI_API int openW5D( eapi_device_handle handles[],
                        int size );
```

Parameters

`eapi_device_handle handles[]`	An array of W5D handles. Each handle would correspond to an attached W5D.
`int size`	The size of the handles array in number of elements. The handles array should be sized to the number of devices expected to be found on the network.

Return Values

n < 0	UDP socket initialization error, or the API was unable to broadcast properly onto the network
n = 0	No devices were discovered on the network. Common case if a device is powered up but has not received an IP address from the networks router.
n > 0	**n** devices were discovered on the network.

Example Code

Note, calling openW5D() should be the first API call when setting up the W5D device in your application.

```
#define ATTACHED_DEVICES (1)

eapi_device_handle handles[ATTACHED_DEVICES];      // array of W5D handles
int n_devices = 0;

n_devices = openW5D(handles, ATTACHED_DEVICES);    // openW5D()

if (n_devices < 0) return(0);       // initialization error
if (n_devices == 0)  return(0);     // prompt user to connect their W5D
if (n_devices > 0)
{
       // begin to call other API functions
       // handles[0] would reference the attached device
}
```

connectDeviceW5D()

The connectDeviceW5D() function is used when a user is aware of the W5D's IP address before an application begins. This is the case for Static IP configuration (see the following section for details) and usually the case when multiple W5D devices are sharing the same network. The connect function will fill element [index] of the handles array with the handle to the W5D at IP address [ip_address].

```
W5DAPI_API int connectDeviceW5D( eapi_device_handle handles[],
                                 int index,
                                 char ip_address[] );
```

Parameters

eapi_device_handle handles[]	An array of W5D handles. Each handle would correspond to an attached W5D.
int index	The index of the handles array (starting at 0) to attach the handle
char ip_address[]	The IP address of the device to connect in "###.###.###.###" format.

Return Values

n < 0	[index] is out of range, or [ip_address] has improper formatting, or UDP socket initialization error
n = 0	No device was discovered on the network at [ip_address]
n = 1	Device discovered at [ip_address] and attached at handle array element [index]

Example Code

```
#define ATTACHED_DEVICES   (2)  // bi-manual W5D setup
#define LEFT_W5D           (0)
#define RIGHT_W5D          (1)

eapi_device_handle handles[ATTACHED_DEVICES]; // array of W5D handles
int rc = 0;

rc = connectDeviceW5D(handles, LEFT_W5D, "192.168.0.101"); // connect @ 0.101
if (rc <= 0) return(0);

rc = connectDeviceW5D(handles, RIGHT_W5D, "192.168.0.102"); // connect @ 0.102
if (rc <= 0) return(0);

// Begin interacting with each device through;
// handles[LEFT_W5D], left device
// handles[RIGHT_W5D], right device
```

closeW5D()

The closeW5D function is called to uninitialize the API. Close will break all links in the handles array, uninitialize UDP sockets and clear out all internal API data structures.

```
W5DAPI_API int closeW5D();
```

Parameters

N/A

Return Values

n < 0	If the API is not already initialized by connectDeviceW5D() or openW5D(), or UDP socket closing caused an error.
n = 1	The API was successfully closed.

Comments

Close is typically called in a user applications de-initialization handler. In haptics applications this could be a handler for closing down the simulation because of window closing or keypressing.

disableW5D()

The disableW5D function is used to put the W5D device into a disabled state; where the device's motor drives are disabled. The disabled state is the W5D's default state when first powered on. For an in depth overview of the W5D's states of operation see the following section. For safety reasons the W5D is always permitted to be disabled, thus a user can disable the device no matter which other running state the device is in (ex, Force or Torque control states).

```
W5DAPI_API int disableW5D( eapi_device_handle handle );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to disable
n = 1	Device disabled successfully

Comments

Disabling the W5D has the following effects;

1) All motor drivers are disabled
2) All device set-points are cleared (force is zeroed, torque is zeroed, position is set to home)
3) Damping, Vibration and Real-time frequency settings **are unchanged**

Example Code

```
void disableHaptics()
{
	int rc = 0;

	rc = disableW5D(handles[0]); // disable W5D
	if (rc < 0) return(0);  // called an invalid device handle
	if (rc == 0) return(0); // W5D may have detached from the network (handle this error)
}
```

enableForceControlW5D()

The enableForceControlW5D function is used to enable the W5D for “impedance control” haptics. In force control the W5D will output world frame forces and torques onto its handle. These force/torque set-points are commanded using the writeForceW5D function. For an overview of the W5D home position and world coordinate system see the following section. For an in depth overview of the W5D's states of operation see the following section.

```
W5DAPI_API int enableForceControlW5D( eapi_device_handle handle,
                                      double *params,
                                      int len );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.	
`double *params`	params[0]	Maximum X force allowed (N)
	params[1]	Maximum Y force allowed (N)
	params[2]	Maximum Z force allowed (N)
	params[3]	Maximum X torque allowed (N.mm)
	params[4]	Maximum Y torque allowed (N.mm)
	params[5]	Maximum Z torque allowed (N.mm)
	params[6]	Maximum Accessory 1 force or torque allowed
	params[7]	Maximum Accessory 2 force or torque allowed
`int len`	The length of the `[params]` array if `[len]` is set to zero, the `[params]` argument is ignored and the W5D will use its current stored force/torque saturation values	

Return Values

n < 0	Invalid device handle
n = 0	Device failed to enter force control state; commonly because it has not been calibrated using homeW5D()
n = 1	Device successfully entered into force control state

Continued on following page...

Comments

Entering into the force control state requires that the W5D has previously been calibrated (see the following section) or also see the homeW5D() function. It is also prerequisite that the W5D is in the disabled state. Hot swapping from "active" enabled states is not permitted with the W5D.

Example Code

```
int rc = 0;

// assuming the W5D is disabled at this point (Can use getStateW5D to check)

rc = homeW5D(handles[0], NULL, 0); // calibrate the W5D
if (rc <= 0) return(0);   // homing failed

rc = enableForceControlW5D(handles[0], NULL, 0); // using standard force/torque saturation
if (rc <= 0) return(0);   // force control enable failed

// device is now ready for writeForceW5D() commands
```

enableTorqueControlW5D()

The enableTorqueControlW5D function is used to enable the W5D for torque control on each of its motors. The torque control state can be thought of as the "open architecture" state because it gives users the increased flexibility of direct control over the W5D mechanism. In torque control, set-points are commanded using the writeTorqueW5D function.

```
W5DAPI_API int enableTorqueControlW5D( eapi_device_handle handle,
                                       double *params,
                                       int len );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`double *params`	params[0] Maximum Motor 1 Torque (N.mm) params[1] Maximum Motor 2 Torque (N.mm) params[2] Maximum Motor 3 Torque (N.mm) params[3] Maximum Motor 4 Torque (N.mm) params[4] Maximum Motor 5 Torque (N.mm) params[5] Maximum Motor 6 Torque (N.mm) params[6] Maximum Motor 7 Torque (N.mm) params[7] Maximum Motor 8 Torque (N.mm)
`int len`	The length of the `[params]` array if `[len]` is set to zero, the `[params]` argument is ignored and the W5D will use its current stored torque saturation values

Return Values

n < 0	Invalid device handle
n = 0	Device failed to enter torque control state
n = 1	Device successfully entered into torque control state

Continued on following page...

Comments

The torque control state allows direct control over the motor torques on the W5D. Entering this state is useful if a user is interested in implementing their own control law's for the W5D.

It is a prerequisite that the W5D is in the disabled state. Hot swapping from “active” enabled states is not permitted with the W5D.

The absolute maximum torque the W5D is capable of for Motors 1 through 6 is **226 N.mm,** a torque saturation setting above this value will be clipped to this absolute maximum.

Example Code

```
int rc = 0;
double max_tau[8] = {85, 85, 85, 85, 85, 85, 125, 125}; // maximum torques allowed

// assuming the W5D is disabled at this point (Can use getStateW5D to check)

rc = enableTorqueControlW5D(handles[0], &max_tau, 8);
if (rc <= 0) return(0);   // torque control enable failed

// device is now ready for writeTorqueW5D() commands
```

enablePositionControlW5D()

The enablePositionControlW5D function is used to enable the W5D for world frame trajectory tracking. In this state the W5D will accept a set-point which consists of a position and orientation in world frame coordinates. These set-points are commanded using the writePositionW5D function. For an overview of the W5D home position and world coordinate system see the following section. For an in depth overview of the W5D's states of operation see the following section.

```
W5DAPI_API int enablePositionControlW5D( eapi_device_handle handle,
                                         double *params,
                                         int len );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`double *params`	params[0] Proportional Gain Motor 1 params[1] Derivative Gain Motor 1 params[2] Proportional Gain Motor 2 params[3] Derivative Gain Motor 2 ... params[14] Derivative Gain Motor 8 params[15] Proportional Gain Motor 8
`int len`	The length of the `[params]` array if `[len]` is set to zero, the `[params]` argument is ignored and the W5D will use its current stored gain values for position control. Note: the default gains are quite weak.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to enter the position control state; commonly because it has not been calibrated using homeW5D()
n = 1	Device successfully entered into the position control state

Comments

Entering into the position control state requires that the W5D has previously been calibrated (see the following section) or also see the homeW5D() function. It is also prerequisite that the W5D is in the disabled state. Hot swapping from "active" enabled states is not permitted with the W5D.

For the proportional gain typical values range from 500 for a weak holding controller to 4500 for a strong holding controller. For the derivative gain settings between 0 and 15 seem to work well, with anything above 15 starting to introduce vibration feedback.

Note: The W5D by design is a low force range haptic device, although it can be used in position control settings, its position holding capabilities are weak at best. Keep this in mind, the "strong" holding controllers with gains up in the 4000 range, are not actually strong (when compared with heavily geared robot systems).

Example Code

```
#define  PG  (2000.0) // Proportional Gain: typical values between 500.0 and 4500.0
#define  DG  (8.0) // Derivative Gain: typical values between 0.0 and 15.0

int rc = 0;
double gains[16] = {PG,DG,PG,DG,PG,DG,PG,DG,PG,DG,PG,DG,0,0,0,0}; // controller gains
// for first 6 motors (not using 6DOF motor or handle motor)

// assuming the W5D is disabled at this point (Can use getStateW5D to check)

rc = homeW5D(handles[0], NULL, 0); // calibrate the W5D
if (rc <= 0) return(0);   // homing failed

rc = enablePositionControlW5D(handles[0], &gains, 16);
if (rc <= 0) return(0);   // position control enable failed

// device is now ready for writePositionW5D() commands
```

homeW5D()

The homeW5D function is used to calibrate the position of the W5D to a known reference point. This procedure is required because the W5D uses relative sensors, which require a reference point to gain absolute position capabilities. This routine should be called when the W5D is positioned into its calibration position, see the following section for details about the W5D's calibration procedure.

```
W5DAPI_API int homeW5D( eapi_device_handle handle,
                        double *offset,
                        int len );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.	
`double *offset`	offset[0]	Homing angle offset for Encoder 1 (radians)
	offset[1]	Homing angle offset for Encoder 2 (radians)
	offset[2]	Homing angle offset for Encoder 3 (radians)
	offset[3]	Homing angle offset for Encoder 4 (radians)
	offset[4]	Homing angle offset for Encoder 5 (radians)
	offset[5]	Homing angle offset for Encoder 6 (radians)
	offset[6]	Homing angle offset for Encoder 7 (radians)
	offset[7]	Homing angle offset for Encoder 8 (radians)
`int len`	The length of the `[offset]` array if `[len]` is set to zero, which is typically the case, the `[offset]` argument is ignored and the W5D will use its standard calibration offset (at the calibration jig position)	

Return Values

n < 0	Invalid device handle
n = 0	Device failed to home
n = 1	Device successfully homed

Continued on following page...

Comments

Homing is required to enter position and force control states but can be performed in any state (recommended to home in disabled state). After homing is complete the W5D will transition into the disabled state. A homed W5D is capable of passing back valid positions and producing world forces, or holding position.

For most use cases, the calibration jig is used for homing. A user should pass in NULL for the offset argument, and pass in 0 for the len argument to perform a standard calibration jig homing.

If a user desires to calibrate the W5D at an alternate position then the offset values can be passed in. These offset values must be correct (at the new position) for the calibration to be accurate.

Example Code

```
int rc = 0;

// assuming the W5D is disabled and held in the calibration position.
// - can use getStateW5D to check the devices current state
// - can use isHomedW5D to check the current homed status

rc = homeW5D(handles[0], NULL, 0); // calibrate the W5D (using calibration jig)
if (rc <= 0) return(0);   // homing failed

// 1) device is now ready to enter position or force control states.
// 2) device will now relay back valid world positions/orientations/velocities.
```

setFrequencyW5D()

The setFrequencyW5D function changes the Real-time frequency for the W5D's real-time loop.

```
W5DAPI_API int setFrequencyW5D( eapi_device_handle handle,
                                int frequency );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`int frequency`	The new real-time frequency in Hz. Valid values are between 100 and 5000.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to change the real-time frequency
n = 1	Device successfully changed the real-time frequency

Comments

If a frequency value is out of the 100Hz to 5000Hz range the W5D will saturate the frequency value at the appropriate bound. By default the W5D runs at a real-time frequency of 1000Hz.

setDampingW5D()

The setDampingW5D function is used to enable the on-device damping features of the W5D. The damping effect can be felt in world coordinates if position or force control states are active, and can be felt on individual motors if torque control state is active. The values passed in by setDampingW5D() are therefore interpreted differently depending on which state the W5D is currently in.

```
W5DAPI_API int setDampingW5D( eapi_device_handle handle,
                              double *damping,
                              int len );
```

Parameters

`eapi_device_handle handle` A valid W5D device handle.

`double *damping`

	Position/Force Control		Torque Control	
	Axis	Input Range	Axis	Input Range
damping[0]	X Force	[0 ... 0.050 max]	Motor 1 Torque	[0 ... 120.0]
damping[1]	Y Force		Motor 2 Torque	
damping[2]	Z Force		Motor 3 Torque	
damping[3]	X Torque	[0 ... 25.0]	Motor 4 Torque	
damping[4]	Y Torque		Motor 5 Torque	
damping[5]	Z Torque		Motor 6 Torque	
damping[6]	Handle Torque 1	unknown	Motor 7 Torque	[0 ... 25.0]
damping[7]	N/A	N/A	Handle Torque 1	unknown

`int len` The length of the damping array, typically 6 to 8 elements.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to change the damping set-point
n = 1	Device successfully changed the damping set-point

Comments

A note of caution using this function, it is very possible to pass damping constants which cause device instability. Suitable value ranges for world and motor space are also quite different. The W5D enforces no restrictions on valid ranges. There are also no restrictions on applying negative (feed-forward) damping values. Force damping constants are in **Newtons per mm/s** and Torque (world and joint space) damping constants are in **Newton-millimeters per radian/s**.

setWatchdogW5D()

The setWatchdogW5D function is used to enable the on-device watchdog timer. Once this timer is enabled, the W5D will automatically disable itself if no commutation is received before the time out period.

```
W5DAPI_API int setWatchdogW5D( eapi_device_handle handle,
                               int ms_timeout );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`int ms_timeout`	The timeout value in milliseconds. The W5D has a minimum and maximum bound for this timeout value from 250ms to 25000ms. If a positive value out of this range is sent, it will be saturated to the appropriate bound. To disable the watchdog timer, set `[ms_timeout]` less than or equal to 0.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to set the watchdog timer
n = 1	Device successfully set the watchdog timer

Comments

To disable the watchdog timer, set `[ms_timeout]` less than or equal to 0. By default the watchdog timer is disabled.

getIPAddressW5D()

The getIPAddressW5D function is used to query a W5D for its IP address. The address is passed back in a printable/readable format of "###.###.###.###". This function can be useful if a user has a W5D connected to a larger network, and is unsure of the IP address the router has assigned to the device.

```
W5DAPI_API int getIPAddressW5D( eapi_device_handle handle,
                                char ip_address[] );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`char ip_address[]`	This character array is filled with the devices IP address, in "###.###.###.###" format.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to return an IP address
n = 1	Device successfully returned it's IP address.

Example Code

```
int rc = 0;
int a1,a2,a3,a4;
char ip_address[256] = "0.0.0.0\n";

rc = getIPAddressW5D(handles[0], ip_address);
if (rc <= 0) return(0); // getIPAddress failed

sscanf(ip_address, "%d.%d.%d.%d", &a1, &a2, &a3, &a4);
printf("W5D at IP %d.%d.%d.%d \n", a1, a2, a3, a4);
```

getStateW5D()

The getStateW5D function is used to query the W5D for its current running state.

```
W5DAPI_API int getStateW5D( eapi_device_handle handle,
                            state_t *state );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`state_t *state`	The W5D's state is passed back in this argument. [`state_t`] is an enumeration of all the possible W5D states, defined in w5dAPI.h

Return Values

n < 0	Invalid device handle
n = 0	Device failed to return an IP address
n = 1	Device successfully returned it's IP address.

Example Code

```
int rc = 0;
state_t w5dstate;

rc = getStateW5D(handles[0], &w5dstate);
if (rc <= 0) return(0); // getState call failed

if (w5dstate == STATE_DISABLED) printf("disabled state \n");
if (w5dstate == STATE_FORCE_CONTROL) printf("force control state \n");
if (w5dstate == STATE_TORQUE_CONTROL) printf("torque control state \n");
if (w5dstate == STATE_POSITION_CONTROL) printf("positon control state \n");
```

isHomedW5D()

The isHomedW5D function is used to query the W5D for its current calibration status. See the following section for details about the W5D's calibration procedure.

```
W5DAPI_API int isHomedW5D( eapi_device_handle handle,
                           homed_t *homed );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`homed_t *homed`	The W5D's homed status is passed back in this argument. `[homed_t]` is an enumeration of all the W5D's homed status, defined in w5dAPI.h

Return Values

n < 0	Invalid device handle
n = 0	Device failed to return its homed status
n = 1	Device successfully returned it's homed status

Example Code

```
int rc = 0;
homed_t w5dhomed;

rc = isHomedW5D(handles[0], &w5dhomed); // check homed status
if (rc <= 0) return(0); // isHomed call failed

if (w5dhomed == HOMED) printf("Device previously calibrated \n");
if (w5dhomed == UNHOMED)
{
       rc = homeW5D(handles[0], NULL, 0); // calibrate the W5D
       if (rc <= 0) return(0);   // homing failed

       printf("Device calibration now \n");
}
```

getFrequencyW5D()

The getFrequencyW5D function is used to query the W5D for its current real-time frequency. By default the W5D will run at 1000Hz on power up.

```
W5DAPI_API int getFrequencyW5D( eapi_device_handle handle,
                                int *frequency );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.
`int *frequency`	The W5D's frequency in Hz passed back to the user

Return Values

n < 0	Invalid device handle
n = 0	Device failed to return its current real-time frequency
n = 1	Device successfully returned it's current real-time frequency

Comments

To change the devices real-time frequency call the setFrequencyW5D() API function

getJointsW5D()

The getJointsW5D function is used to query the W5D for its current joint angles and joint velocities. Calling this function at any time in any state will return up to date values. Joint angles are in radians and Joint velocities are in radians/second.

```
W5DAPI_API int getJointsW5D( eapi_device_handle handle,
                             double *joints,
                             int len );
```

Parameters

`eapi_device_handle handle` A valid W5D device handle.

`double *joints`

Position		Velocity	
joints[0]	Motor 1 Angle (rad)	joints[8]	Motor 1 Velocity (rad/s)
joints[1]	Motor 2 Angle (rad)	joints[9]	Motor 2 Velocity (rad/s)
joints[2]	Motor 3 Angle (rad)	joints[10]	Motor 3 Velocity (rad/s)
joints[3]	Motor 4 Angle (rad)	joints[11]	Motor 4 Velocity (rad/s)
joints[4]	Motor 5 Angle (rad)	joints[12]	Motor 5 Velocity (rad/s)
joints[5]	Motor 6 Angle (rad)	joints[13]	Motor 6 Velocity (rad/s)
joints[6]	Motor 7 Angle (rad)	joints[14]	Motor 7 Velocity (rad/s)
joints[7]	Motor 8 Angle (rad)	joints[15]	Motor 8 Velocity (rad/s)

`int len` The length of the joints array in number of double elements. This provides protection so the function will not overwrite your joints array.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to return its current joint positions
n > 0	Size of the joint data returned. Normally 16, but customized W5D devices may return more or less.

Comments

See the following section for the motor numbering convention of a W5D. For a source code example of using this function take a look at the Torque Control software example.

Note: It is also possible to return the joint angles/velocites as a return element of the writeForceW5D/writeTorqueW5D/writePositionW5D functions. If you want to obtain the joint angles in a Torque/Position or Force control setting it is more communication efficient to use these “write” functions. In this case it only takes one data exchange to update the W5D's setpoint and refresh the user applications joint positions/velocities.

getPositionW5D()

The getPositionW5D function is used to query the W5D for its current position, orientation and velocity. A prerequisite for calling this function is successful calibration of the W5D. Calling without calibration will return the home position.

```
W5DAPI_API int getPositionW5D( eapi_device_handle handle,
                               double *pos,
                               int len );
```

Parameters

`eapi_device_handle handle` — A valid W5D device handle.

`double *pos`

Position		Orientation	
pos[0]	X Position (mm)	pos[3]	Orientation Matrix [1,1]
pos[1]	Y Position (mm)	pos[4]	Orientation Matrix [1,2]
pos[2]	Z Position (mm)	pos[5]	Orientation Matrix [1,3]
pos[12]	X Velocity (mm/s)	pos[6]	Orientation Matrix [2,1]
pos[13]	Y Velocity (mm/s)	pos[7]	Orientation Matrix [2,2]
pos[14]	Z Velocity (mm/s)	pos[8]	Orientation Matrix [2,3]
pos[18]	Pinch Position (rad)	pos[9]	Orientation Matrix [3,1]
pos[19]	Pinch Velocity (rad/s)	pos[10]	Orientation Matrix [3,2]
		pos[11]	Orientation Matrix [3,3]
		pos[15]	X Angular Velocity (rad/s)
		pos[16]	Y Angular Velocity (rad/s)
		pos[17]	Z Angular Velocity (rad/s)

`int len` — The size of the `*pos` array in number of double elements.

Return Values

n < 0	Invalid device handle
n = 0	Device failed to return its current positions
n > 0	Size (in number of doubles) of the position data returned. Normally 18, but could increase for any W5D firmware customizations.

Continued on next page ...

Comments

See the following section for an outline of the W5D World reference frame and home position.

Note: It is also possible to return position/orientation/velocity data as a return element of the writeForceW5D/writeTorqueW5D/writePositionW5D functions. If you want to obtain this world position/orientation data in a Torque/Position or Force control setting it is more communication efficient to use these "write" functions. In this case it only takes one data exchange to update the W5D's setpoint and refresh the user applications world position/orientation.

Example Code

```
int rc = 0;
double task_pos[20] = {0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0};  // home pos, zero velocity
double position[3] = {0,0,0};
double orientation[9] = {1,0,0,0,1,0,0,0,1};  // identity matrix
double velocity[3] = {0,0,0};
double ang_velocity[3] = {0,0,0};
double scissor_pinch[2] = {0,0};   // [0] = angle, [1] = velocity

// assuming the W5D is calibrated

rc = getPositionW5D(handles[0], task_pos, 20); // get current position
if (rc <= 0) return(0); // get position call failed
if (rc != 20) return(0); // check, getting all 20 elements

// example of how the values would copy over
position[0] = task_pos[0];
position[1] = task_pos[1];
position[2] = task_pos[2];
orientation[0] = task_pos[3];
orientation[1] = task_pos[4];
orientation[2] = task_pos[5];
orientation[3] = task_pos[6];
orientation[4] = task_pos[7];
orientation[5] = task_pos[8];
orientation[6] = task_pos[9];
orientation[7] = task_pos[10];
orientation[8] = task_pos[11];
velocity[0] = task_pos[12];
velocity[1] = task_pos[13];
velocity[2] = task_pos[14];
ang_velocity[0] = task_pos[15];
ang_velocity[1] = task_pos[16];
ang_velocity[2] = task_pos[17];
scissor_pinch[0] = task_pos[18];
scissor_pinch[1] = task_pos[19];
```

writeForceW5D()

The writeForceW5D function is used to command a world space force and torque vector to the W5D while in the force control state. The function will also return an updated task or joint position to the user in the return data field. This function should solely be used in the simulation loop of a haptics application.

```
W5DAPI_API int writeForceW5D( eapi_device_handle handle,
                              double *force,
                              int len,
                              returndata_t returntype,
                              double *returndata );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.	
`double *force`	force[0]	X Force (N)
	force[1]	Y Force (N)
	force[2]	Z Force (N)
	force[3]	X Torque (N.mm)
	force[4]	Y Torque (N.mm)
	force[5]	Z Torque (N.mm)
	force[6]	Accessory Torque 1 (if W5D is customized)
	force[7]	Accessory Torque 2 (if W5D is further customized)
`int len`	Length of the force array (in number of double elements). For a standard W5D this would be 6 elements.	
`returndata_t returntype`	POSITION_DATA for position data, JOINT_DATA for joint data.	
`double *returndata`	The returndata array will contain either device **task position** or **joint position** data. For task position data the format will follow the same convention as in the getPositionW5D function. For joint position data the format will follow the same convention as the getJointsW5D function.	

Return Values

n < 0	Invalid device handle
n = 0	Device failed to update its force set-point
n > 0	Size (in number of doubles) of the data returned in the `[returndata]` array.

Continued on next page ...

Comments

See the following section for an outline of the W5D World reference frame and home position. This function will only take effect in the force control state but may be called any other time (ideally while disabled) to set an initial values.

Example Code

```
double task_pos[18] = {0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0}; // home position, zero velocity
double force[6] = {0,0,0,0,0,0}; // initial force command is zero
int rc = 0;

// 1) Assuming the device is calibrated
// 2) Assuming we're in the force control state
// 3) W5D is set up for handles[0]

// Haptics Loop
while (not_done) // not_done would be changed in another thread (in a shut down routine)
{
	// call writeForce and request a return value of POSITION_DATA
	rc = writeForceW5D(handles[0], force, 6, POSITION_DATA, task_pos);
	if (rc <= 0) return(0); // force command call failed

	// writeForceW5D has given us an updated task position

	// re-draw the current scene (or pass the task_pos to the graphics thread)
	drawScene(task_pos);
	// update the force based on current position
	doScenePhysics(task_pos, force);

	// Have a timing mechanism here to keep the loop refreshing at a regular interval
	loopTimer();
}
```

writeTorqueW5D()

The writeTorqueW5D function is used to command joint space torques to the W5D while in the torque control state. The function will also return an updated task or joint position to the user in the return data field.

```
W5DAPI_API int writeTorqueW5D( eapi_device_handle handle,
                               double *torque,
                               int len,
                               returndata_t returntype,
                               double *returndata );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.	
`double *torque`	torque[0]	Motor Torque 1 (N.mm)
	torque[1]	Motor Torque 2 (N.mm)
	torque[2]	Motor Torque 3 (N.mm)
	torque[3]	Motor Torque 4 (N.mm)
	torque[4]	Motor Torque 5 (N.mm)
	torque[5]	Motor Torque 6 (N.mm)
	torque[6]	Motor Torque 7 (N.mm), The 6DOF Handle Motor
	torque[7]	Accessory Torque 1 (if W5D is customized)
`int len`	Length of the torque array (in number of double elements). For a standard W5D this would be 6 elements (5DOF system) and 7 elements (6DOF system).	
`returndata_t returntype`	POSITION_DATA for position data, JOINT_DATA for joint data.	
`double *returndata`	The returndata array will contain either device **task position** or **joint position** data. For task position data the format will follow the same convention as in the getPositionW5D function. For joint position data the format will follow the same convention as the getJointsW5D function.	

Return Values

n < 0	Invalid device handle
n = 0	Device failed to update its force set-point
n > 0	Size (in number of doubles) of the data returned in the `[returndata]` array.

Continued on next page ...

Comments

See the following section for an outline of the W5D's motor torque labelling and torque direction convention. This function will only take effect in the torque control state but may be called any other time (ideally while disabled) to set an initial values.

Example Code

```
double joint_pos[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // joint positions/velocities
double torque[6] = {0,0,0,0,0,0}; // initial torque command is zero
int rc = 0;

// 1) Assuming we're in the torque control state
// 2) W5D is set up for handles[0]

// Application Loop
while (not_done) // not_done would be changed in another thread (in a shut down routine)
{
	// call writeTorque and request a return value of JOINT_DATA
	rc = writeTorqueW5D(handles[0], torque, 6, JOINT_DATA, joint_pos);
	if (rc <= 0) return(0); // torque command call failed

	// writeTorqueW5D has given us an updated joint position and velocity

	// a control law function which uses the joint positions and velocities
	// to re-calculate new torques to command to the W5D
	controlLaw(torque, joint_pos);

	// Have a timing mechanism here to keep the loop refreshing at a regular interval
	loopTimer();
}
```

writePositionW5D()

The writePositionW5D function is used to command world space position and orientation to the W5D while in the position control state. The function will also return an updated task or joint position to the user in the return data field.

```
W5DAPI_API int writePositionW5D( eapi_device_handle handle,
                                 double *position,
                                 int len,
                                 returndata_t returntype,
                                 double *returndata );
```

Parameters

`eapi_device_handle handle`	A valid W5D device handle.	
`double *position`	position[0]	X Position (mm)
	position[1]	Y Position (mm)
	position[2]	Z Position (mm)
	position[3]	Orientation Matrix [1,1]
	position[4]	Orientation Matrix [1,2]
	position[5]	Orientation Matrix [1,3]
	position[6]	Orientation Matrix [2,1]
	position[7]	Orientation Matrix [2,2]
	position[8]	Orientation Matrix [2,3]
	position[9]	Orientation Matrix [3,1]
	position[10]	Orientation Matrix [3,2]
	position[11]	Orientation Matrix [3,3]
	position[12]	Accessory Position (If w5d is customized)
`int len`	Length of the position array (in number of double elements). For a standard W5D this would be 12 elements.	
`returndata_t returntype`	POSITION_DATA for position data, JOINT_DATA for joint data.	
`double *returndata`	The returndata array will contain either device **task position** or **joint position** data. For task position data the format will follow the same convention as in the getPositionW5D function. For joint position data the format will follow the same convention as the getJointsW5D function.	

Continued on next page ...

Return Values

n < 0	Invalid device handle
n = 0	Device failed to update its position set-point
n > 0	Size (in number of doubles) of the data returned in the [`returndata`] array.

Comments

See the following section for an outline of the W5D World reference frame and home position. This function is used in a position control setting, calling it in other states will result in a return value of 0. Call the enablePositionControlW5D function to enable the position control mode, and set the gains used for the position controllers proportional controller.

This function will only take effect in the position control state but may be called any other time (ideally while disabled) to set an initial values.

Example Code

```
double read_pos[18] = {0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0};	// home position, zero velocity
double command_pos[12] = {0,0,0,1,0,0,0,1,0,0,0,1};	// home position
int rc = 0;
double t = 0.0;

// 1) Assuming the device is calibrated
// 2) Assuming we're in the position control state
// 3) W5D is set up for handles[0]
// 4) An initial position has been commanded to the W5D close to our trajectory
// ... 4 avoids a violent movement of the W5D (be well aware of the initial position command)

// Position Control Loop
while (not_done) // not_done would be changed in another thread (in a shut down routine)
{
	// call writePosition and request a return value of POSITION_DATA
	rc = writePositionW5D(handles[0], command_pos, 12, POSITION_DATA, read_pos);
	if (rc <= 0) return(0); // position command call failed

	// writePositionW5D has given us an updated task position (read_pos)
	// can use this data to draw a graphics scene if desired

	// tracking a sinusoidal trajectory solely in X
	t+=0.01; // incrementing t by the loop time period (assuming 0.01 s or 100Hz)
	command_pos[0] = 15.0*sin(0.5*t); // +/- 15mm with w=0.5 rad/s

	// Have a timing mechanism here to keep the loop refreshing at 0.01s
	loopTimer();
}
```

4.0 Application Examples

4.1 Haptic Interaction with a Virtual World

The following example code shows the haptics thread portion of a multi-threaded virtual world application which interacts with a single W5D device. A haptics thread (shown here), a graphics thread, and a signal handling thread would typically exist. Interaction with the W5D API would take place exclusively in the haptics thread. Many functions called in this code example would be implemented elsewhere in the application, including; initialize_timer(), sim_physics(), global_copy(), and timer_wait().

Example Code

```
void HapticsThread( void *MyID ) // Haptics Thread (Loop)
{
        int t_samp = 2; // sampling frequency in milliseconds
        double local_pos[18];
        double local_force[6] = {0,0,0,0,0,0}; // initialize force to zero
        int rc;
        homed_t w5d_homed = UNHOMED;
        eapi_device_handle Devices[1]; // device handle for single W5D

        rc = openW5D(Devices, sizeof(Devices)); // open W5D API
        if (rc < 1) _endthread();  // api error OR no devices attached
        rc = isHomedW5D(Devices[0], &w5d_homed); // check if device has been homed
        if (rc <= 0) _endthread();
        if (w5d_homed != HOMED) _endthread();
        rc = disableW5D(Devices[0]); // first disable, just in case device is enabled
        if (rc <= 0) _endthread();
        rc = enableForceControlW5D(Devices[0], NULL, 0); // enable force control
        if (rc <= 0) _endthread();

        initialize_timer(t_samp); // call a (user written) function which initializes a waitable timer

        // Haptics Loop ********************************************************************
        while (notdone) // notdone is set to 0 by another thread when application terminates
        {
            // Writing force to the W5D, and receiving the current position back
            writeForceW5D(Devices[0], local_force, 6, POSITION_DATA, local_pos);

            // Simulation physics function (user written)
            sim_physics(local_pos, local_force);

            // Copy over current position value (mutex protected) to global memory.
            // Memory is shared with a graphics thread (for drawing the simulation)
            global_copy(local_pos, 18);

            // Block in a (user written) function until waitable timer reaches tSamp (2 ms)
            wait_timer();
        }

         closeW5D(); // close down the W5D API
        _endthread();
}
```

4.2 Joint mapped proportional-controller

The following example code shows a two W5D setup, where a right handed W5D commands the joint position of a left handed W5D using the writeTorqueW5D() function. User implemented functions include; proportional_controller(), wait_timer(), and initialize_timer()

Example Code

```
#define ATTACHED_DEVICES   (2)  // Two W5D setup @ 0.101 and @ 0.102
#define LEFT_W5D           (0)
#define RIGHT_W5D          (1)
#define KP                 (50) // Proportional Gain (N.mm/rad)

void TorqueController() // Joint Torque control of LEFT_W5D using RIGHT_W5D
{
        int t_samp = 2; // sampling frequency in milliseconds
        double joint_pos_l[8]; double joint_pos_r[8]; double joint_tau[8];
        int rc;
        eapi_device_handle handles[ATTACHED_DEVICES]; // array of W5D handles

        rc = connectDeviceW5D(handles, LEFT_W5D, "192.168.0.101"); // connect @ 0.101
        if (rc <= 0) return(0);
        rc = connectDeviceW5D(handles, RIGHT_W5D, "192.168.0.102"); // connect @ 0.102
        if (rc <= 0) return(0);
        rc = disableW5D(handles[RIGHT_W5D], NULL, 0); // disable just in case enabled previously
        if (rc <= 0) return(0);
        rc = disableW5D(handles[LEFT_W5D], NULL, 0); // disable just in case enabled previously
        if (rc <= 0) return(0);
        rc = enableTorqueControlW5D(handles[LEFT_W5D], NULL, 0); // enable torque control for left
        if (rc <= 0) return(0);

        initialize_timer(t_samp); // call a (user written) function which initializes a waitable timer

        // Control Loop ************************************************************
        while (notdone) // notdone is set to 0 by another thread when application terminates
        {
            // Read in Right W5D's joint angles
            getJointsW5D(handles[RIGHT_W5D], joint_pos_r, 8);

            // Read in Left W5D's joint angles
            getJointsW5D(handles[LEFT_W5D], joint_pos_l, 8);

            // joint_tau[i] = KP*(joint_pos_r[i] - joint_pos_l[i]), i = 0..7
            proportional_controller(joint_pos_l, joint_pos_r, joint_tau, KP);

            // Write the joint torque command to the left w5d
            writeTorqueW5D(handles[LEFT_W5D], joint_tau, 8, JOINT_DATA, NULL);

            // Block in a (user written) function until waitable timer reaches tSamp (2 ms)
            wait_timer();
        }

        closeW5D(); // close down the W5D API
}
```

4.3 Position Controlled Trajectory Tracking

Example Code