# FT17- Preliminary Sensor Manual

**Document version FT_SMDB_B0.1**

# Table of Contents

# 1 Introduction

This is a preliminary user manual for the FT17 sensors with DataBox electronics. It provides the main information of the sensor handling and operation. It describes the network connectivity options and the communications protocol for accessing sensor data. The sensor is supplied with Libraries for Linux and examples for standard data acquisition operation. Additional information on system operation may be found with the supplied libraries and examples.
For communicating for the first time to the sensor and for checking that the sensor you have just received is functioning you can follow the steps described in section 0

This document should be treated as preliminary. Any significant changes will be communicated to the customer.

# 2 Important Information

This sensor is a sensitive instrument and should be always handled with care. Never exceed pure forces of 100N and Torques of 0.5Nm on any axis.
Be extremely careful with the loads applied through a mounted tool. Depending on the length of the tool, small forces can appear as large torques at the sensor flange and cause permanent damage. Avoid if possible the use of long tools attached at the sensor output flange.

Be extremely careful when mounting the sensor to a robot or grounding it on a mechanical structure and be especially careful when mounting tools to the sensor output flange. When mounting a tool at the sensor output flange make sure that no torque is transferred by the screw tightening action between the flange and the sensor base. If the screw tightening loads exceed the sensor specifications permanent damage may occur.

A method to achieve safe mounting of the tool to the output flange is to first mount the tool on the sensor flange while the sensor base is still unmounted (Not grounded). First insert and lightly tighten the mounting screws to hold the tool in place. Next hold the sensor-tool assembly by the output flange or by the tool and tighten the screws while the sensor base is ungrounded and completely free. Never hold the tool by the base while tightening the screws of the output flange.

Then place the sensor base at the grounding assembly or robot and insert and tighten the base mounting screws.

Never tamper with the sensor body screws as this will void the calibration and can damage the sensor permanently.

# 3 Connecting and setting up a sensor

In order to communicate with the sensor it is necessary to know its network properties IP, Subnet mask and Port and to have compatible properties set at the host LAN adapter. Communications can be achieved in one of two modes, the *User IP mode* or the *Default IP mode*. In *User IP mode* the system communicates at the user defined IP and port while in *Default IP mode* the system communicates at the default hard coded IP and port. These modes of communication are described in the following sections. Once communication has been established in either of the two modes the user has full access to the sensor settings and data.

## 3.1 Configuring the sensor through the Embedded Web Server

Your sensor has an embedded web server. Configuration of the sensor network settings (User IP Mode) and of the filter settings is done through the embedded web server.

Prior to accessing the sensor's embedded web server, the sensor network settings and the network settings of the connected PC Host should be appropriately set. The Web Server can be accessed at the active address (i.e. either at the Default IP or at the User IP) depending on the IP mode selected, as described in the next sections.
Once able to see a sensor (e.g. can successfully execute a ping at the sensor's IP address) open a browser window and type the IP address of the sensor. If the network has been configured correctly you should be able to see the home page of the Embedded Web server in the browser window (Figure 1)

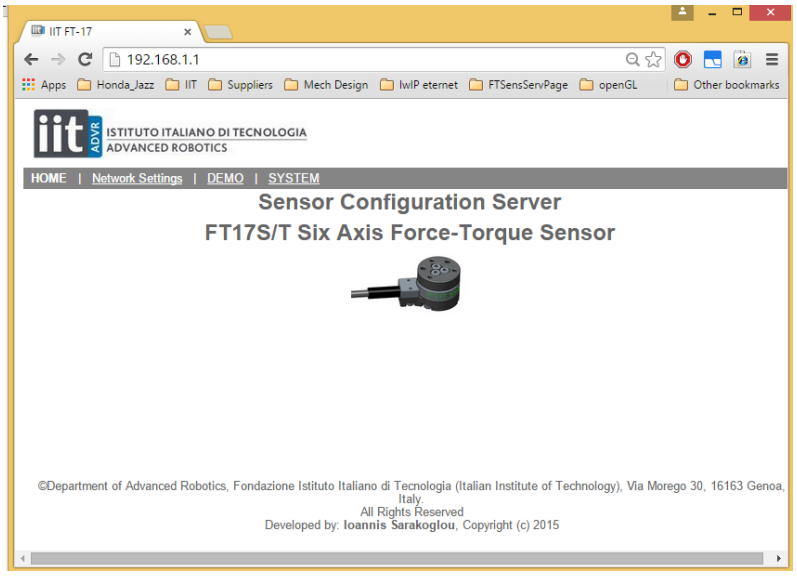


**Figure 1 Home page of the Embedded Web Server**

### 3.1.1 Default IP mode

Since it is possible that the sensor network settings set by a user can become unknown, the system has a Default IP mode, which can be activated by means of a slide switch (IP SW). By setting the IP switch to the "On" position and restarting the sensor (disconnect and reconnect the power supply) the system reverts to a predefined Default local IP, Subnet mask and port. These default network settings can then be used to access the sensor either for normal use or for setting the desired User network settings. Setting of the slide switch to the Default IP mode is also visually signalled by the sensor LED which becomes red with flashing green.
When the sensor is in the Default IP mode the active network settings are:
Default IP: 192.168.1.1
Default Subnet mask: 255.255.0.0
Default Port: 23

Once the sensor is in Default IP mode the user can communicate with it as normal using these network properties.

To be able to see the sensor from a host computer the user must set the host's LAN adapter properties accordingly so that the fixed address of the sensor is accessible. An example of compatible settings for the host when in Default IP mode is:
Host IP: 192.168.1.2
Host Subnet mask: 255.255.0.0

Once the host LAN network adapter properties have been set and the sensor is in the Default IP mode an easy way to check for network compatibility is to execute a ping command to the sensor IP address in a command window.

Once the sensor is accessible either in Default IP mode or in User IP mode access and configuration of the sensor's settings is performed through the **Embedded Web Server**.

***Quick steps for connection to the web server in Default IP mode***

This is a quick list of steps for connecting to a sensor through a web browser.
List of Steps:

- Set the sensor IP switch "IP SW" to the "ON" position.
- Connect the sensor with an RJ45 Ethernet cable to the computer.
- Remove and reconnect the power supply connector to the 5V power input.

At this point the sensor Mode LED should be red with slowly blinking green light. The sensor address now is 192.168.1.1 with subnet mask 255.255.0.0

- Set the computer's Ethernet IP to 192.168.1.2 and the network mask to 255.255.0.0 or 255.255.255.0
- Open a Web Browser and type the address 192.168.1.1

The sensor web server should load on the web browser's window. You can see the various user accessible settings and load data in the sensor's web pages.

### 3.1.2 Setting User IP mode Network Properties

When configuring the sensor through the web server in order to accept the changes made to the IP address make sure you click the save parameters to flash button and then restarting the sensor (remove and reinsert the power connector). After restart and when in User IP Mode (IP switch set to OFF) in order regain access to the sensor through the web server you will need to have correct Host network settings and to type the new IP address at the browser's address bar.

### 3.1.3 Single- and multi-sensor operation in User IP mode

The sensor firmware and the accompanying libraries offer two modes of operation in terms of the number of sensors in a network. These are the single sensor operation and the multi sensor operation. These two modes can be configured through the embedded web server network settings. The purpose of these two modes is to provide along with the provided libraries automated connectivity to both single and multiple sensors and provide a method access to all sensor data through high level commands and data structures.

#### *3.1.3.1 Single sensor mode*

The standard mode is the single sensor operation where the IP of the sensor in User IP mode (IP switch set to OFF) is explicitly defined through the *IP Address* field of the *Single Sensor Network Settings* of the web server. This IP is the active address when in User IP mode.
Effective sensor IP = *IP Address*

It is advised when in *Single Sensor Network* mode to check in the *Multi Sensor Settings* that the sensor *Board Number* is 1. If it is not it is advised that the *Board Number* is set to 1 followed by a *Save Settings to Flash* and a power up sequence.

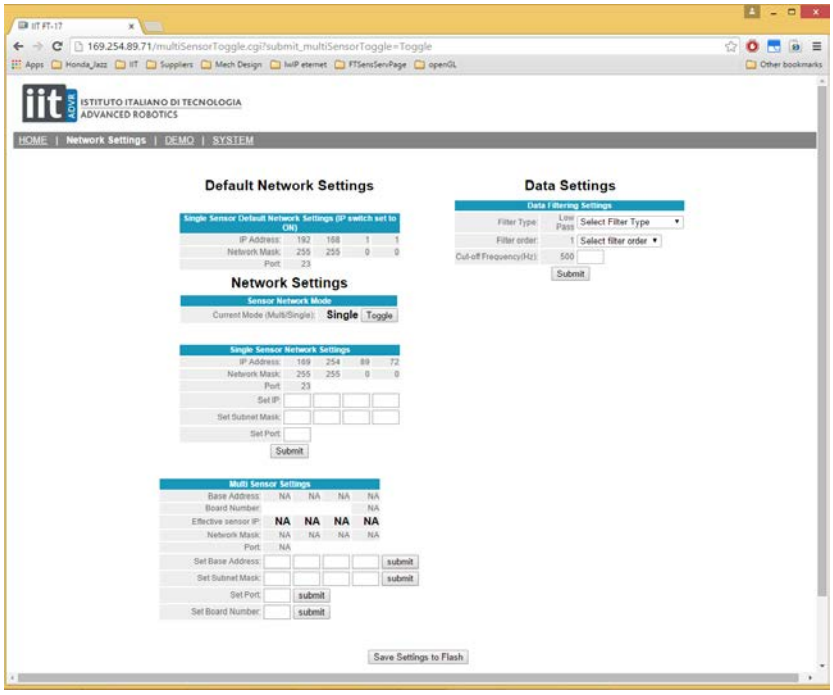


**Figure 2 Single sensor network settings**

### *3.1.3.2 Multi sensor mode*

In the multi sensor mode the *Effective Sensor* IP of each sensor on the network while in User IP mode (IP switch set to OFF) is the combination of the *Base Address* and the *Board Number* parameters accessible through the relevant settings in the web server.

Effective sensor IP = *Base address+ Board number*

A network of multiple sensors should have a common *Base Address* and *Network Mask*. These must be compatible with the network settings of the host. Each sensor should have a unique *Board Number* starting from 1 for the first sensor on the network and up to the maximum number of supported sensors.

When in multi sensor mode after a successful initialisation the data the sensor are available in an array of sensor structures. The $i^{th}$ element of the array is a structure holding the data of the sensor with B*oard Number=i*.

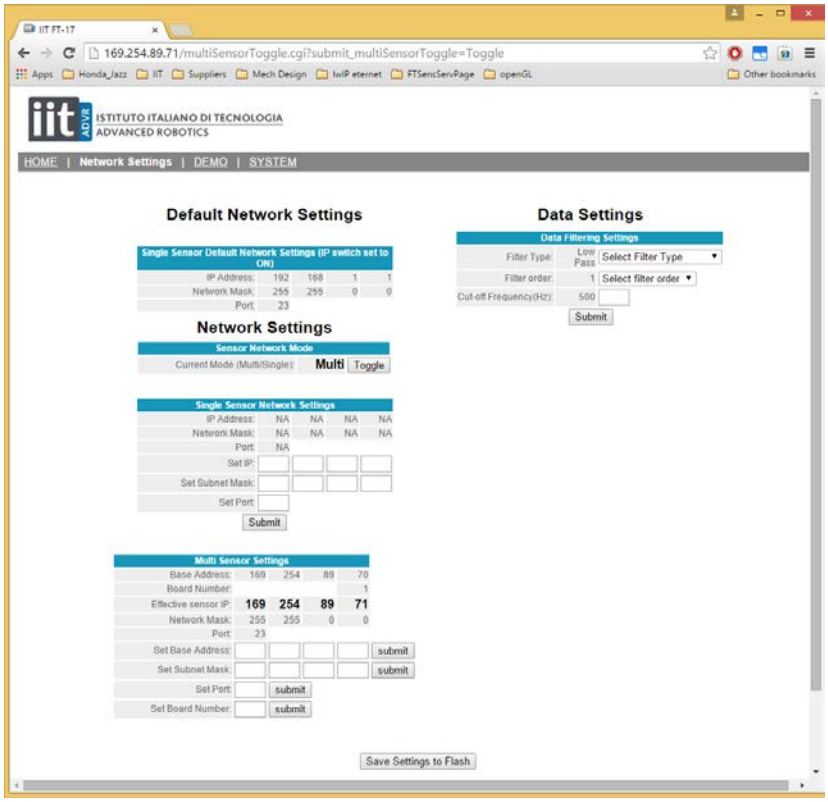


**Figure 3 Multi sensor network settings**

### 3.1.4 Embedded Filters

The sensor provides embedded filtering of the sensor data. Access to filters is through the web server *Data Filtering Settings.* There are two *Filter Type* options i.e. Low and High pass Butterworth filters with up to 3 poles.
After setting the *Filter Type*, *Pole* and *Frequency* settings click *Save Settings to Flash* in order for the settings to take effect.

To simply toggle between a currently set *Filter Type* (i.e. either a low Pass or a High Pass) and the option of *No filtering* you do not need to click *Save Settings to Flash.*

### 3.1.5 Sensor Demo webpage

For troubleshooting purposes the web server has a *Demo* webpage Figure 5where is displayed a snapshot of the load and strain gauge readings of the sensor. The Demo webpage also gives access to a bias command. The sensor readings can be updated by a reload of this webpage.

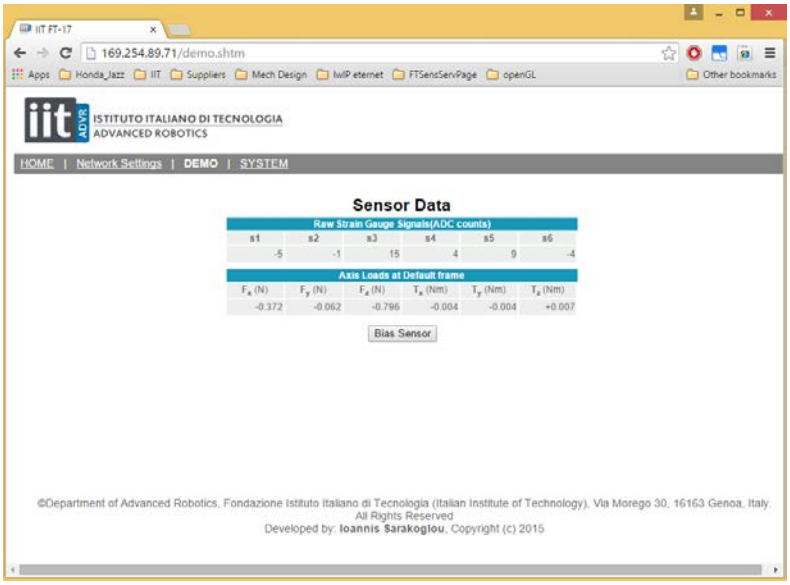


**Figure 4 Sensor Demo webpage**

# 4 Data Acquisition

The sensor uses a command protocol consisting of TCP and UDP commands for implementing two modes of data acquisition; The *Streaming Mode* and the *Polling Mode*.

In this release the user TCP/UDP command protocol is not yet fully released. Standard communications in *Streaming Mode* is implemented at the low level by the supplied libraries. More User TCP/UDP commands will be released in the near future.

Direct use of the low level Communication protocol is considered an advanced use and will be intended for expert users who may want to implement their own communications functions and have to need to construct and parse the packets to and from the sensor.

## 4.1 Streaming mode

In streaming mode the sensor is set up with TCP commands to send a UDP stream at a predefined rate and with predefined content in terms of the sensor data to be present in the UDP packet. The computer host is then responsible for receiving and parsing these data according to the sensor data protocol. The supplied libraries implement streaming mode through standard TCP and UDP libraries (windows, Linux) and provide the streamed sensor data in a structure.

The following code extracted from the file FT17_reading.cpp demonstrates the standard method of setting up and acquiring data from the sensor in *Streaming Mode.*

The call to *ft17.configure(Rate, Policy)* sets the broadcasting *Rate* in milisecconds and the broadcast data *Policy* according to the description in Appendix A1.1.

```
#include <FT17_reading.h>
#include <iostream>

int main() {

        ft_data ft_bc_data; //declare a sensor structure

        FT17Interface ft17 ( "eth0" ); //Set the ethernet adapter

        ft17.init(); //Call the sensor discovery and initialisation Fnc.
        ft17.configure ( 1, 127 );// Set the broadcast rate in milliseconds
and the Policy according to Appendix A1.1
```

```
        ft17.start_broadcast();// start broadcasting
        for ( int i = 0; i < 100000; i++ ) {
            ft17.get_broadcast_data ( ft_bc_data );//Receive Sensor Data

                std::cout << " FT Filtered : "   << ft_bc_data.FT_filt[0]
<< " \t " << ft_bc_data.FT_filt[1] << " \t " << ft_bc_data.FT_filt[2] << "
\t "
                                                 << ft_bc_data.FT_filt[3]
<< " \t " << ft_bc_data.FT_filt[4] << " \t " << ft_bc_data.FT_filt[5] << "
\t " << std::endl;

        }
        ft17.stop_broadcast();//Stop broadcasting

        std::cout << " 2 second sleep... " << std::endl;
        sleep(2.0);
        std::cout << " Calibrating offset... " << std::endl;
        ft17.calibrate_offset();//Bias the sensor
        std::cout << " 2 second sleep... " << std::endl;
        sleep(2.0);

        ft17.start_broadcast();// start broadcasting
        for ( int i = 0; i < 100000; i++ ) {
           ft17.get_broadcast_data ( ft_bc_data ); //Receive Sensor Data

                std::cout << " FT Filtered : "   << ft_bc_data.FT_filt[0]
<< " \t " << ft_bc_data.FT_filt[1] << " \t " << ft_bc_data.FT_filt[2] << "
\t "
                                                 << ft_bc_data.FT_filt[3]
<< " \t " << ft_bc_data.FT_filt[4] << " \t " << ft_bc_data.FT_filt[5] << "
\t " << std::endl;

        }
        ft17.stop_broadcast();//Stop broadcasting


        return 0;

}
```

## 4.2 Polling mode

In polling mode the sensor accepts a request to send a predefined packet of data and in response it sends one packet of the current sensor data. The polling mode is implemented entirely using the UDP communications protocol described in the Appendix. Acquiring data in polling mode may increase the communication overhead due to the need for continuous requests to the sensor, however it can assist in data synchronisation as data arrive through a user controlled sequence.

The polling mode has not been implemented in the current library in the way that the streaming method has and is up to the user to use their UDP libraries of preference for creating and parsing the command and data packets at a low level. This should give the freedom to expert users to control communications timing as they wish in order to tackle data synchronisation.

# 5 Communications issues due to firewall activity (Windows)

The above should indicate that the user has access to the sensor. If this test is successful but communications with the sensor through the provided libraries (either in streaming or in polling mode) is problematic first consider adjusting your firewall settings.

In windows go to the advanced settings of the firewall, find the relevant service in the inbound and in the outbound services that is communicating with the sensor and make sure it is not blocked. If it has been blocked appropriately set its properties to unblock it.

# Appendix

## A1. Poling mode Protocol

To perform data acquisition in polling mode, after successful creation of the UDP socket at the sensor IP and PORT, the HOST can send datagrams to the sensor according to the UDP command protocol described in Table 1. First it is necessary to issue the command SET_SINGLE_UDP_PACKET_POLICY in order to set the POLICY which defines the data to be returned by the sensor in its response. The SET_SINGLE_UDP_PACKET_POLICY command consists of the fixed header bytes, the board number byte, POLICY lower byte (POLICY0), the POLICY higher byte (POLICY1) and the checksum CHKSUM byte. Descriptions of the data defined by the POLICY with examples are presented in Appendix A1.1. Examples of calculating the CHKSUM are shown in C code in section
The SET_SINGLE_UDP_PACKET_POLICY command must be sent once at the beginning of a data acquisition session. The sensor will then respond with the requested data in subsequent responses. After the policy has been correctly set the host can request single packets by issuing the GET_SINGLE_UDP_PACKET command. Upon successful reception of the GET_SINGLE_UDP_PACKET command the sensor will respond with the BCAST_DATA_PACKET_MT UDP packet. This packet contains the header, the data and the checksum. The Host can parse the returned data according to the packet structure and the data defined by the selected POLICY.

**UDP Commands to Board**

| | | | Board Number | | | | |
|---|---|---|---|---|---|---|---|
| SET_SINGLE_UDP_PACKET_POLICY | FF | 3 | 3 | n | POLICY0 | POLICY1 | CHKSUM |
| GET_SINGLE_UDP_PACKET | FF | 1 | 4 | n | CHKSUM | | |
| UDP_CALIBRATE_OFFSETS | FF | 1 | 5 | n | CHKSUM | | |

**Table 1 UDP Commands for Poling mode operation**

**UDP Data Sent from Board**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| BCAST_DATA_PACKET_MT | FD | n+2 | BC | Board ID | data0 | ... | datan | CHKSUM |

**Table 2 UDP response from the sensor with data packet**

## A1.1 Broadcast policy and data packet description (common with Streaming Mode)

The Policy consists of the POLICY0 and POLICY1 bytes described in Table 3. The POLICY1 Byte is reserved for future use and should be always set to 0. The POLICY bytes are effectively bytes of flags which are used to select the data to be returned by the sensor. Table 3 describes the available data and the

number of data that will be returned inside the UDP response from the sensor for each of the selected data fields. Examples of setting the POLICY bytes are presented below. The data bytes shown in the response packet described in Table 2 are ordered according to their order in the POLICY order shown in Table 3. The data in the UDP command and response packets appear in bytes with the least significant placed first in the packet and the most significant byte last.

Example 3 presents the case for POLICY0=65 (binary= 01000001) for receiving the data *Raw Data with Offset* and *Filtered Forces and Torques*. The first 12 bytes would be the *Raw Data with Offset* of the 6 strain gauge channels (Ch1, Ch2, Ch3, ch4,ch5, ch6) and the next 24 bytes would be the *Filtered Forces and Torques* (Fx,Fy,FZ,Mx,My,Mz)

**POLICY0**

| **bit$_i$** | **Data** | **No. of bytes b0 $_i$** | **Data Type** | **Policy Value POLICY0$_i$** |
|---|---|---|---|---|
| 0 | Raw Data with Offset (Ch1, Ch2, Ch3, ch4,ch5, ch6) | 6*2=12 | 6*short | 1 |
| 1 | Forces-Torques (Fx,Fy,FZ,Mx,My,Mz) | 6*4=24 | 6*int | 2 |
| 2 | Raw Data (Ch1, Ch2, Ch3, ch4,ch5, ch6) | 6*2=12 | 6*ushort | 4 |
| 3 | Temp & DC supply | 4 | 2*ushort | 8 |
| 4 | Time Stamp | 4 | 1*uint | 16 |
| 5 | Fault flags | 2 | 2*bytes | 32 |
| 6 | Filtered Forces-Torques (Fx,Fy,FZ,Mx,My,Mz) | 6*4=24 | 6*int | 64 |
| 7 | RESERVED | NA | | 128 |

**POLICY1 (Reserved set always to 0)**

| **bit$_i$** | **data** | **No. of bytes b1 $_i$** | **Policy Value POLICY1$_i$** |
|---|---|---|---|
| 0 | RESERVED | NA | 1 |
| 1 | RESERVED | NA | 2 |
| 2 | RESERVED | NA | 4 |
| 3 | RESERVED | NA | 8 |
| 4 | RESERVED | NA | 16 |
| 5 | RESERVED | NA | 32 |
| 6 | RESERVED | NA | 64 |
| 7 | RESERVED | NA | 128 |

Table 3 Broadcast policy and data Packet description

Examples of Broadcast Policy.

Example 1

In this example the

| bit*i* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Data** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| **Select** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **POLICY0$_i$** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**POLICY0=1**

| bit*i* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Data Select** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| **POLICY1$_i$** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|

**POLICY1=0**

**P0LICY= POLICY0+ 256*POLICY1=1**

**UDP Command sequence:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SET_SINGLE_UDP_PACKET_POLICY | FF | 3 | 3 | n | POLICY0 | POLICY1 | CHKSUM |
| GET_SINGLE_UDP_PACKET | FF | 1 | 4 | n | CHKSUM | | |

**UDP response received from the sensor:**

| | | | | | | |
|---|---|---|---|---|---|---|
| BCAST_DATA_PACKET_MT | FD | n+2 | BC | Board ID | Raw Data with Offset (12 bytes) | CHKSUM |

Example 2

| bit*i* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Data Select** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **POLICY0$_i$** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**POLICY0=64**

| bit*i* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Data Select** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **POLICY1$_i$** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**POLICY1=0**

**P0LICY= POLICY0+ 256*POLICY1=64**

**UDP Command sequence:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SET_SINGLE_UDP_PACKET_POLICY | FF | 3 | 3 | n | POLICY0 | POLICY1 | CHKSUM |
| GET_SINGLE_UDP_PACKET | FF | 1 | 4 | n | CHKSUM | | |

**UDP response received from the sensor:**

| | | | | | | |
|---|---|---|---|---|---|---|
| BCAST_DATA_PACKET_MT | FD | n+2 | BC | Board ID | Filtered Forces-Torques (24 bytes) | CHKSUM |

Example 3

| bit*i* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Data Select** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| **POLICY0$_i$** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**POLICY0=127**

| bit*i* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Data Select** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **POLICY1$_i$** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**POLICY1=0**

**P0LICY= POLICY0+ 256*POLICY1=127**

**UDP Command sequence sent to sensor:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SET_SINGLE_UDP_PACKET_POLICY | FF | 3 | 3 | n | POLICY0 | POLICY1 | CHKSUM |
| GET_SINGLE_UDP_PACKET | FF | 1 | 4 | n | CHKSUM | | |

**UDP response received from the sensor:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| BCAST_DATA_PACKET_MT | FD | n+2 | BC | Board ID | Raw Data with Offset (6 bytes) | Filtered Forces-Torques (24 bytes) | CHKSUM |

**Load data in the UDP packet**

| Raw Data with Offset (6 shorts in 12 bytes) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Ch1 | | Ch2 | | Ch3 | | Ch4 | | Ch5 | | Ch6 | |
| Ch1Byte0 | Ch1Byte1 | Ch2Byte0 | Ch2Byte1 | Ch3Byte0 | Ch3Byte1 | Ch4Byte0 | Ch4Byte1 | Ch5Byte0 | Ch5Byte1 | Ch6Byte0 | Ch6Byte1 |

| Filtered Forces-Torques (6 integers in 24 bytes) | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fx | | | Fy | | | Fz | | | Mx | | | My | | | Mz | | |
| FxByte0 | … | FxByte5 | FyByte0 | … | FyByte5 | FzByte0 | … | FzByte5 | MxByte0 | … | MxByte5 | MyByte0 | … | MyByte5 | MzByte0 | … | MzByte5 |

## A1.2 Load data and Conversion to SI Units

The data returned in the fields *Raw Data with Offset* (selected by POLICY0 bit 0) and *Raw Data* (selected by POLICY0 bit 2) are the outputs of the 16 bit Analogue to Digital Converter in ADC counts before multiplication with the calibration matrix. These data are supplied to allow the possibility for use of the sensor with user generated custom calibration matrix at the Host.
The data returned in the *Forces-Torques* (selected by POLICY0 bit 1) *Filtered Forces-Torques* (selected by POLICY0 bit 6) are in SI units that have been multiplied by 1000000. The user after parsing the UDP data and creating the 6 integers representing the Fx, Fy, FZ, Mx, My, Mz loads must divide them by 1000000 to get the loads in N and Nm.

## A1.3 Poling Mode Code Examples

This example code is taken from a windows example which is not yet released however it demonstrates the technique of building the Poling mode UDP commands for setting up the PCOLICY, for requesting a single packet of data and for biasing the sensor ( SET_SINGLE_UDP_PACKET_POLICY, GET_SINGLE_UDP_PACKET and UDP_CALIBRATE_OFFSETS). It also demonstrates a method for calculating the checksum byte for each command.
Finally a code example demonstrated the process of parsing the received UPD response into the data fields according to the current selected POLICY.

```
void devMeasurementFns_IITFT::UDPPolicy(unsigned char (&UDPpolicy)[2])
{
      char MessageBuf[BufLenS];
      int ucTemp=0;

      //Create the packet of the SET_SINGLE_UDP_PACKET_POLICY Command//
      //Command bytes:  FF    3     3     BoardNum    policyLB    policyHB    CHKSUM
      MessageBuf[0] = 0xFF;
      MessageBuf[1] = 0x03;
      MessageBuf[2] = 0x03;
      MessageBuf[3] = 0x01;//board number
      MessageBuf[4] = UDPpolicy[0];//policy Lower byte
      MessageBuf[5] = UDPpolicy[1];//policy Higher byte

      //Calculate checksum byte
      for( ucTemp = 0, MessageBuf[6]=0; ucTemp < 6; ucTemp++)
        {
            MessageBuf[6] -= MessageBuf[ucTemp];
        }
      UDPcomm_IITFT->UDPSendMessage(MessageBuf);
}
```

```
--------------------------------------------------------------------------
void devMeasurementFns_IITFT::RequestFTsensorData()
{
      char MessageBuf[BufLenS];
      int ucTemp=0;

      //Create the packet of the GET_SINGLE_UDP_PACKET Command//
      //Command bytes:  FF    1     4     BoardNum    CHKSUM
      MessageBuf[0] = 0xFF;
      MessageBuf[1] = 0x01;
      MessageBuf[2] = 0x04;
      MessageBuf[3] = 0x01;//board number

      //Calculate checksum byte
      for( ucTemp = 0, MessageBuf[4]=0; ucTemp < 4; ucTemp++)
        {
            MessageBuf[4] -= MessageBuf[ucTemp];
        }
      UDPcomm_IITFT->UDPSendMessage(MessageBuf);

}
--------------------------------------------------------------------------
void devMeasurementFns_IITFT::BiasFTsensor()
{
      char MessageBuf[BufLenS];
      int ucTemp=0;

      //    Create the packet of the UDP_CALIBRATE_OFFSETS Command//
      //Command bytes:  FF    1     5     BoardNum    CHKSUM
      MessageBuf[0] = 0xFF;
      MessageBuf[1] = 0x01;
      MessageBuf[2] = 0x05;
      MessageBuf[3] = 0x01;//board number
      //Calculate checksum byte
      for( ucTemp = 0, MessageBuf[4]=0; ucTemp < 4; ucTemp++)
        {
            MessageBuf[4] -= MessageBuf[ucTemp];
        }
      //send UDP command packet
```

```
      UDPcomm_IITFT->UDPSendMessage(MessageBuf);
}
-------------------------------------------------------------------------


int devMeasurementFns_IITFT::ReadFTsensorData(FT_Sensor_Poll &FT_Sensor_Data)
{
char RecvBuf[BufLenR];
int COMresult;

RequestFTsensorData(); //Request UDP Data

  if (COMresult=UDPcomm_IITFT->UDPReceieveMessage(RecvBuf)>0) //Receive the UDP message
  {
      if(RecvBuf[2] == (char)0xbc)   // Check if this is a FT sensor boradcast packet
      {
            int boardNo=RecvBuf[3];

            //------------------------   Parse FT sensor Broadcast Data  ---------------------------------
            //-------------------------------------------------------------------------------------------
            //-------------------------------------------------------------------------------------------
            int PcCnt=3;
      if (UDPpolicy&0x1)//check if bit 0 of the BCAST_POLICY lower byte is asserted
            {
            for (int i=0; i<6; i++ )
            {
            if(((RecvBuf[PcCnt+2])& 0x80)>0) //create
            FT_Sensor_Data.Raw_Offs[i]=((unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+2]<<8)& 0xff00)) - 0xffff;
            else
            FT_Sensor_Data.Raw_Offs[i]=((unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+2]<<8)& 0xff00));
            PcCnt=PcCnt+2;
            }
            }
      if (UDPpolicy&0x2)//check if bit 1 of the BCAST_POLICY lower byte is asserted
            {
            for (int i=0; i<6; i++ )
            {
            FT_Sensor_Data.FT[i]= (unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+2]<<8)& 0xff00)
            +((RecvBuf[PcCnt+3]<<16)& 0xff0000)+((RecvBuf[PcCnt+4]<<24)& 0xff000000);
```

```
      PcCnt=PcCnt+4;
      FT_Sensor_Data.ft[i]=(float)(FT_Sensor_Data.FT[i])/1000000;
      }
      }
if (UDPpolicy&0x4)//check if bit 2 of the BCAST_POLICY lower byte is asserted
      {
      for (int i=0; i<6; i++ )
      { //This is a 2 byte unsigned number of the AD converter (value range 0-65535)
      FT_Sensor_Data.Raw[i]=((unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+2]<<8)& 0xff00));
      PcCnt=PcCnt+2;
      }
      }
if (UDPpolicy&0x8)//check if bit 3 of the BCAST_POLICY lower byte is asserted
      {
      FT_Sensor_Data.temp_Vdc=(unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+2]<<8)& 0xff00)
            +((RecvBuf[PcCnt+3]<<16)& 0xff0000)+((RecvBuf[PcCnt+4]<<24)& 0xff000000);
      PcCnt=PcCnt+4;
      }
if (UDPpolicy&0x10)//check if bit 4 of the BCAST_POLICY lower byte is asserted
      {
      FT_Sensor_Data.tStamp=(unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+2]<<8)& 0xff00)
            +((RecvBuf[PcCnt+3]<<16)& 0xff0000)+((RecvBuf[PcCnt+4]<<24)& 0xff000000);
      PcCnt=PcCnt+4;
      }
if (UDPpolicy&0x20)//check if bit 5 of the BCAST_POLICY lower byte is asserted
      {
      FT_Sensor_Data.fault= (unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+1]<<8)& 0xff00);
      PcCnt=PcCnt+2;
      }

if (UDPpolicy&0x40)//check if bit 6 of the BCAST_POLICY lower byte is asserted
      {
      for (int i=0; i<6; i++ )
      {
      FT_Sensor_Data.filt_FT[i]= (unsigned char)RecvBuf[PcCnt+1]+((RecvBuf[PcCnt+2]<<8)& 0xff00)
      +((RecvBuf[PcCnt+3]<<16)& 0xff0000)+((RecvBuf[PcCnt+4]<<24)& 0xff000000);
      PcCnt=PcCnt+4;
      FT_Sensor_Data.filt_ft[i]=(float)(FT_Sensor_Data.filt_FT[i])/1000000;
      }
```

```
      }
  }
}

return COMresult;
}
```