2005-12

Wireless smart shipboard sensor network

Nozik, Andrew B.

Monterey California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

WIRELESS SMART SHIPBOARD SENSOR NETWORK

by

Andrew Benjamin Nozik

December 2005

Thesis Advisor:	Xiaoping Yun
Second Reader:	Robert Hutchins

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Wireless Smart Shipboard Sensor Network			**5. FUNDING NUMBERS**
6. AUTHOR(S) Andrew Benjamin Nozik			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis studies the feasibility of developing a smart shipboard sensor network. The objective of the thesis is to prove that sensors can be made smart by keeping calibration constants and other relevant data such as network information stored on the sensor and a server computer. Study will focus on the design and implementation of an Ipsil IPμ8930 microcontroller, which is then connected, by the standard TCP/IP implementation, to a network where the sensor information can be seen using a web page. The information to make the sensor "smart" will be stored on the Ipsil chip and server computer and can by accessed by a HTML based program. By taking pre-computed calibration constants that minimize the measurement errors and writing them through the web page stored in the Ipsil chip's EEPROM, the calibrated sensor reading can be calculated.

The expected contribution from the research effort would be a reduction in manpower, increased efficiency, and a greater awareness of plant and equipment operation among naval vessels, specifically the DDX. Hardware is relatively inexpensive, reliable, and COTS (Commercial Off the Shelf) available. If implemented, a Smart Shipboard Sensor Network would allow the watch standers, CHENG, OOD, and CO, to all see the same information about the ship's engineering plant and equipment.

A prototype sensor test bed was constructed in the laboratory, which consists of an Ipsil IPμ8930 microcontroller, a Linksys LAN router, and a Dell Inspiron 9300 laptop. The newly developed smart sensor was successfully demonstrated.

14. SUBJECT TERMS Network, Wireless, and Sensor.			**15. NUMBER OF PAGES** 87
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

WIRELESS SMART SHIPBOARD SENSOR NETWORK

Andrew B. Nozik
Lieutenant, Unites States Navy
B.S., United States Naval Academy, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2005

Author: Andrew B. Nozik

Approved by: Xiaoping Yun
Thesis Advisor

Robert Hutchins
Second Reader

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis studies the feasibility of developing a smart shipboard sensor network. The objective of the thesis is to prove that sensors can be made smart by keeping calibration constants and other relevant data such as network information stored on the sensor and a server computer. Study will focus on the design and implementation of an Ipsil IPμ8930 microcontroller, which is then connected, by the standard TCP/IP implementation, to a network where the sensor information can be seen using a web page. The information to make the sensor "smart" will be stored on the Ipsil chip and server computer and can by accessed by a HTML based program. By taking pre-computed calibration constants that minimize the measurement errors and writing them through the web page stored in the Ipsil chip's EEPROM, the calibrated sensor reading can be calculated.

The expected contribution from the research effort would be a reduction in manpower, increased efficiency, and a greater awareness of plant and equipment operation among naval vessels, specifically the DDX. Hardware is relatively inexpensive, reliable, and COTS (Commercial Off the Shelf) available. If implemented, a Smart Shipboard Sensor Network would allow the watch standers, CHENG, OOD, and CO, to all see the same information about the ship's engineering plant and equipment.

A prototype sensor test bed was constructed in the laboratory, which consists of an Ipsil IPμ8930 microcontroller, a Linksys LAN router, and a Dell Inspiron 9300 laptop. The newly developed smart sensor was successfully demonstrated.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge the financial support of SPAWAR San Diego, for their generous Fellowship grant that allowed for the purchase of equipment and travel used in support of this thesis.

The author would like to thank Professor Xiaoping Yun for all his guidance during this thesis. His patience, understanding, and assistance helped me to overcome all the difficulties encountered along the way and was unmatched during the entire time. Thank you sir for your time and the knowledge you have passed on to me.

The author would like to thank the following people for technical support during the course of this thesis effort: James Calusidian, Randy Rupnow, and especially LT Carl Trask for his invaluable help with Java programming.

A very special thank you goes to my beloved wife, Diane, for her constant support during the process of writing this thesis. Her dedication and love has given me the strength I need to face all obstacles, and our marriage has been a continuous source of happiness for me.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The number of sensors onboard naval ships is rapidly increasing, and the accurate reading of these sensors is becoming more crucial to successfully completing the missions they carry out. However, with the reduction in manning, the ability to calibrate, test, and maintain sensors needs to be accomplished with fewer people.

The solution to this problem is to create a sensor network that will allow a single person to observe and maintain all the shipboard sensors. To do this, each sensor must have its own unique IP address as well as have the ability to store information about itself and about the network it is on.

The objective of this thesis was to build and test a "smart" plug and play Ethernet based sensor that can be used on equipment in such a shipboard sensor network. "Smart" means that if a pressure reading from a piece of equipment is desired, the sensor will know what piece of equipment it is being used on, the range of pressure readings required from it, how to connect and send it's information to the network, and most importantly what its calibrations contents are. This research will expand upon previous work done by Professor Yun and others on network-based shipboard sensors and the closed-loop calibration of those sensors on wireless LAN's, however now an Ipsil microcontroller will be used to create a network to be used in the engineering spaces on naval vessels. Ipsil chips are built to be reliable, long-term, and cost-effective.

The main thrust of the study was the design and implementation a smart sensor using Ipsil's IPμ8930 which is then connected to a network where the sensor information can be seen using an html based web page. The information to make the sensor "smart" will be stored on both the EEPROM of the IPμ8930 and on server computer. A web page located on the Ipsil chip interfaces with a user in the form of a java applet program.

A working prototype "smart" sensor has been produced in the laboratory for a shipboard environment. It consists of an Ipsil IPμ8930 microcontroller, a Linksys LAN router, and Dell Inspiron 9300 laptop. However, it was for lab use only and not for a

shipboard environment. The thesis proves that a smart shipboard sensor network is feasible and that it can be constructed using commercial-off-the-shelf (COTS) hardware along with developed software.

I. INTRODUCTION

A. REASON FOR NEW TECHNOLOGY

There are 3,742 hull, mechanical, and electrical (HM&E) sensors on DDG ships. Readings such as temperature, pressure, and flow rate are taken frequently and, with the United States Navy progressing towards lower manning levels, more sensors will be required to monitor shipboard equipment. As the number of sensors increases (it is expected to be around 200,000 for the DDX) and the number of people available to maintain these sensors declines, new technology will be required to modernize the maintenance and repairs of these sensors. Over seventy percent (2,669) require periodic calibration, forty-five percent (1,189) of those that require calibration have standard visual gauge type displays, and the other fifty-five percent (1,480) of those that require periodic calibration send data for display on the ship Machinery Control Systems (MCS) displays. Calibration constants are used to accurately convert the output voltage or current to usable information. Previous efforts have been made to make the calibration of these sensors easier and faster, however there is still a need to store the data from the sensors and their calibration constants as well as making the sensors themselves easier to install, use, and display their valuable information. Accurate sensor data is crucial to the reliable operation and readiness of a ship. In order to meet the future needs in increased sensors and a reduction of manpower shipboard, sensor equipment needs to be designed and integrated into a network capable of meeting these requirements. [Ref. 1]

B. CURRENT PROCESSES

Even though work has been done to wirelessly calibrate HM&E sensors, the new processes have yet to take effect and the current system is still very slow and requires a large amount of labor from a ship's work force when manual paper procedures are used on standard "dumb" sensors. A team of at least two or more technicians from a shore command (SISCAL) must come aboard with heavy (around 100 pounds) portable

equipment and, after the ship's force tags out the equipment from service, the SISCAL team isolates the sensor from the system. At least two people are needed; one to read the calibration standards and instruments at the sensor and the other technician at the MCS console. The calibration constants are changed by hand. The biggest problem to automating this process is that there is no common interface between both hardware and software to the monitoring and/or control systems. [Ref. 2]

C. PROPOSED CONCEPT OF NEW TECHNOLOGY

Conventional, 'dumb' analog and digital sensors provide a simple measured voltage or current output proportional to what they are reading. The calibration constants are stored somewhere away from the sensors themselves. New, 'smart' plug and play sensors will have the capability to store their own adjustable calibration constants as well as being network capable, and have retainable configurations in their RAM or EEPROM.

With the introduction of the Wireless Enhancement of the Integrated Conditioning Assessment System (WEI), shipboard sensor monitoring may be done using PC access through the 802.11b wireless standard. This open architecture system will allow a roam-able notebook computer to capture sensor readings from the smart sensors and compare them to a calibration standard and then re-write the calibration constants. This is possible because each sensor can be assigned its own static IP address.

Already a feasibility demonstration of this concept and a proof of concept aboard the ex-USS FOSTER have been accomplished. Smart plug and play Ethernet based sensors can also be configured to ICAS. Ipsil chips are specifically designed to be used for networking without complicated programming. This technology is also built to be reliable, long-term, and cost-effective.

D. BENEFIT OF CONCEPT TO THE NAVY

The expected contribution from the research efforts would be a reduction in manpower, increased efficiency, and a greater awareness of plant and equipment operation among naval vessels, specifically the DDX. Hardware is relatively inexpensive, reliable, and COTS (Commercial Off the Shelf) available. The plausible benefits to the maintenance of U.S. Navy ships come from the modernization of calibrating, configuring, and networking of shipboard sensors. All of which could be accomplished by trained shipboard personnel following Admiral Vern Clark's Sea Basing portion of Sea Power 21 direction to transfer shore-based capabilities to sea-based systems [Ref. 3]. "As the Navy experiments with optimum manning initiatives to reduce crew size while increasing sustainability, this technology makes sense for the transformation of the Navy to a world of fewer operators, more condition based maintenance and a more precise automated environment" [Ref. 4].

E. DESCRIPTION OF CHAPTERS IN THESIS

This first chapter is an introduction to the thesis. The second chapter covers the problem statement and proposed approaches to the research. The third chapter concentrates on the hardware components. It discusses each component's specifications, operating conditions, and use in the sensor/monitoring system. The fourth chapter investigates the software designed. Each code has its own section to discuss its role in the system, the inputs and outputs, and operating procedures. The operating procedures section also explains each additional code that may be found within the main code being discussed. The fifth chapter presents the results that were found and discusses what worked and what failed. The sixth chapter has the conclusions from the thesis as well as the final analysis, recommendations, and proposed future work. Finally, the appendices contain the HTML and JAVA code that was designed and used for this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. PROBLEM STATEMENT AND PROPOSED APPROACHES

A. PROBLEM STATEMENT

Chapter I reveals that there is a need to have a reduced amount of labor and manning to maintain and calibrate shipboard sensors. Smart sensors will satisfy this need by combining the capability to save and store their own adjustable calibration constants, being network capable, and having retainable configurations in their RAM or EEPROM.

The goal of this thesis is to develop a working prototype smart sensor for a shipboard environment. Limitations of the study will be based on equipment used, and access to a naval vessel for testing of a wireless network in the engineering spaces, and how to incorporate these smart sensors into current shipboard equipment.

B. PROPOSED APPROACHES

Currently the ex-USS FOSTER is being used as a test platform for wireless technology that is designated to be installed and used in future U.S Navy ships. In addition, the USS HOWARD (DDG-83) had 40 wireless Network Capable Application Processors (NCAPs) installed for at-sea testing in January of 2002. The USS MASON (DDG-87) is also slated to be built with wireless NCAPs and Gateways. Figure 1 shows how a wireless environment can work on a ship. [Ref 3. and Ref 5].

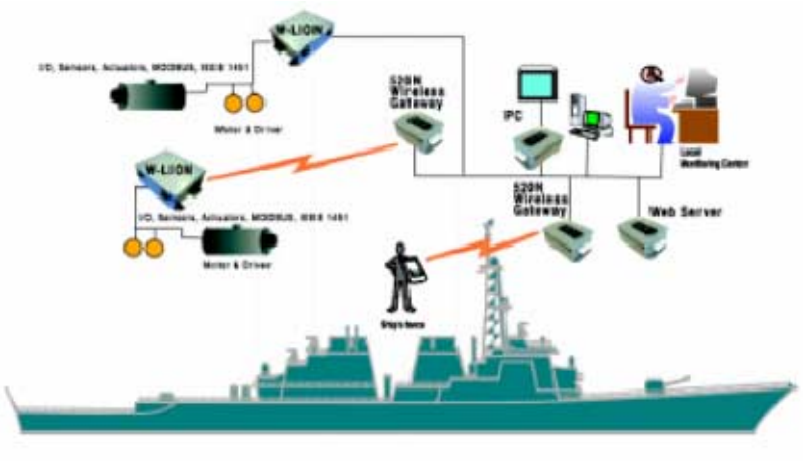

Figure 1. Diagram Showing Use of Wireless NCAP and Gateway in Shipboard Environment Such as DDG 83. [From Ref. 6]

Ipsil chips are specifically designed to be used for networking without complicated programming. This technology is also built to be reliable and cost-effective. The advantage of having a smart sensor network would be to reduce manpower and increase the overall operational awareness of shipboard equipment.

To accomplish the task of creating a smart sensor in the laboratory an Ipsil IPμ8930 with a Linksys LAN router was used. The assembly was then connected to a network where the sensor information can be seen using a html based web page on a Dell Inspiron 9300 laptop. The information to make the sensor "smart" was stored in a server computer where a web page located in the EEPROM of the Ipsil chip can access it by using a java applet program. The Ipsil IPμ8930 has a small built-in webserver that can deliver web pages on request for reporting sensor readings, or making changes to the calibration constants and configuration. In addition, this allows a sensor to be assigned a static IP address making it network capable. [Ref. 7]

C. PREVIOUS WORK

Previous work on creating a smart sensor was done by two previous students at NPS, Steven Joseph Perchalski and Eusébio Pedro da Silva, as well as by Mr. Randy Rupnow of NAVSEA Corona, and Professor Xiaoping Yun. Their work consisted of developing a new calibration system for analog and digital "smart" sensors. The work concentrated on calibrating a sensor using a wireless tablet PC and a NCAP with one or several LabVIEW programs running on both. The LabVIEW program used a least squares fitting method for the calibration process and then wrote the new constants to the sensor's RAM or EEPROM. This was a closed-loop process that wirelessly calibrated shipboard sensors reduced the number of personnel and time required to complete the task.

In a recent test on board the ex-USS FOSTER. Professor Yun and Mr. Rupnow were able to join the previous work using a LabVIEW program with the installed ICAS system. This was a very important step towards demonstrating the feasibility of using a smart shipboard sensor network on navy ships.

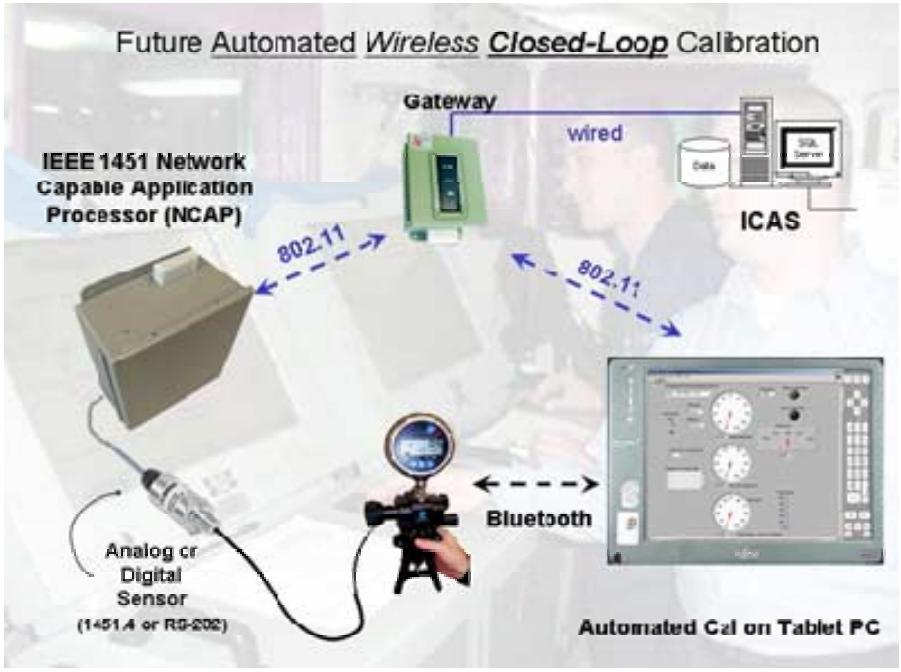

Figure 2. Previous Concept of Operation. [From Ref. 4]

D. SUMMARY

In review, the research problem was to create a smart sensor in the laboratory using an Ipsil IPμ8930 microcontroller, a Linksys LAN router, and a Dell Inspirion 9300 laptop where the sensor information can be seen and accessed using an html based web page and java applet program. Having a wireless smart sensor that is network capable with calibration constants and configurations that are changeable over the network will reduce man-power and increase the overall operational awareness of shipboard equipment. This work continues the process of creating a smart shipboard sensor network by developing a prototype smart sensor using COTS hardware that is relatively inexpensive and reliable. The next chapter more closely looks at the hardware used to accomplish this task.

III. HARDWARE DESCRIPTION AND DISCUSSION

A. INTRODUCTION TO HARDWARE

By the year 2010, 95% of internet connected devices will not be computers. They instead will be network enabled electronic devices. This thesis is proving that with an Ipsil product, shipboard engineering sensors can be network enabled and made smart with the use of both html web pages and java programs. This chapter discusses the different types of hardware used in this thesis. The equipment used includes an Ipsil IPμ8930 microcontroller, a Linksys LAN router, and a Dell Inspirion 9300 laptop. [Ref. 8]

B. IPSIL IPμ8930 CHIP

The IPμ8930 is a compact TCP/IP network controller and web server module designed to rapidly enable and network electronic devices. It combines a TCP/IP controller, HTTP-compliant webserver, MCU peripheral, Modbus TCP node, 10BaseT Ethernet controller, and an A/D converter into a single daughterboard. [From Ref. 11]

Figure 3. IPμ8930. [From Ref. 8]

1. Specifications

The IPμ8930 can be used to monitor and control analog devices, such as sensors, using the eight general-purpose I/O pins. Access to these channels can be done through user-defined web pages, such as the ones created for this thesis. Over 408 kilobytes of space is available for web pages and java programs, or other programs such as a lab view program. Features include: [From Ref. 9]

- TCP/IP and UDP read and write
- HTTP v1.0 compliant embedded webserver
- 8 Analog (10-bit) or digital ports
- MCU serial interface—that provides read/write access to all RAM channels.
- Supports DHCP and ICMP (ping)
- Secure mode with password
- Built-in file system—any file accessible from a browser
- 5.0 Volt operation
- Size: 33.2 mm x 34.5 mm

2. Operating Conditions

The IPμ8930 is the main piece of equipment that is what makes a "smart" sensor smart in this thesis. It is the "brains" behind the operation and is what controls the operation of collecting data from a shipboard engineering sensor, attaching an IP address, and holds the sensor calibration constants, as well as process the sensor data using those constants, and sending the information on to the ship's LAN. Once the information has been put on the ship's LAN it can be accessed by either a web page with a java applet or ICAS (Integrated Conditioning Assessment System).

3. Use in Thesis

The IPμ8930 is the main piece of equipment used in this thesis. All tests and experiments were done with this chip. The first experiment was that when the web page link is clicked, the java program runs and a display of eight LEDs turn on. A user is then able to input a desired number (calibration constant) into the java program on the PC and then the corresponding number in hex is displayed by the LEDs. Hence, the number is now stored with the sensor and can be seen by any user on the network that knows the IP address. Figure 6 shows the IPμ 8930 and LED display.

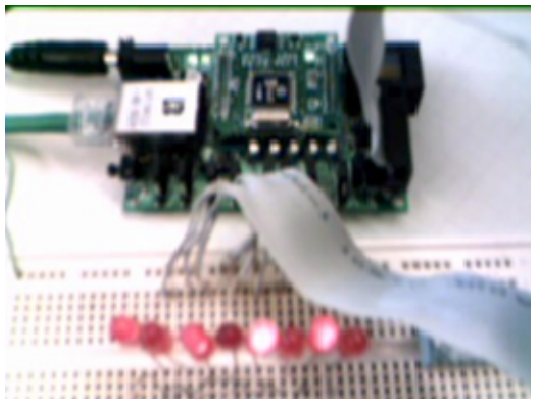

Figure 4. Ipsil IPμ 8930 and 8 LED Display.

C. IPSIL IP / HTML CAPABLE SENSOR BLOCKS

The SensorBlock Wifi has a small built in webserver which delivers web pages on request for reporting sensor readings. It also supports Modbus TCP that allows interfaces to industrial software such as National Instruments LabView™ or users can develop and download their own web pages as in this thesis. [From Ref. 10]

Figure 5. SensorBock WiFi. [From Ref. 8]

1. Specifications of Ipsil SensorBlock WiFi

The Ipsil SensorBlock can support either 4-20mA, 0-5V signal, or a dry contact input signal. The Sensorblock reports the raw values which can then be converted. [From Ref. 10] Some of the specifications are:

- 10-bit resolution
- Sampling Rate: 200 Hz
- Uses dynamic (DHCP) or fixed (user-defined) IP address
- Secure mode requiring a password for all access
- 512KB non-volatile memory available for web pages/web object storage
- Can operate across routers/bridges
- Collision, Transmit, Receive LEDs
- Wireless technology: 802.11b
- Frequency: 2.4 GHz
- Radio output: +18dBm (64 mW)
- Receive sensitivity: -84 dBm
- Typical Range: 300+ feet indoors
- Speed: up to 11 mb/s
- Security: WEP 128-bit Encryption, MAC filtering

2. Operating Conditions

The SensorBlock WiFi collects data from a shipboard engineering sensor attaches an IP address to that sensor and can hold the sensor calibration constants for that specific sensor, process the sensor data using those constants, and then send this information on to the ship's wireless LAN. In effect the Ipsil chip inside of the sensor block is what makes that engineering sensor become a "smart" sensor. Once the information has been put on the ship's LAN it can be accessed by either a National Instruments LabView™ program or ICAS.

3. Use in Thesis

In this thesis, the SensorBlock WiFi was not used because it was not available for purchase in a timely fashion for the completion of this thesis. However, if it had been available, a web page similar to the ones produced could be uploaded to the SensorBlock along with a java program that stores calibration constants and converts the raw data into useable data. More information on the web page and java program will be discussed in chapters four and five. A LINKSYS wireless LAN router would connect the SensorBlock WiFi to the network where a PC could then be used to view the web page.

D. SILICON ON SAPPHIRE PRESSURE TRANSMITTER

This is the PX4200 Series that comes with a Loop Pressure Monitor, and is made by Omega. The pressure monitor displays the pressure right above the connection of the sensor to the IPμ8930. This provides a visual check to coordinate between the actual reading and the reading received by the IPμ8930.

Figure 6. PX4200 Pressure Sensor with Loop Powered Pressure Monitor. [From Ref. 11]

1. Specifications

The Omega silicon on sapphaire pressure transmitter has the following specifications: [From Ref. 11]

- Excitation: 13 to 36 Vdc with PM1000
- 18 to 36 Vdc reverse polarity protected
- Load Driving Capacity: 1150Ω@
- 36 Vdc, R max = (Vs-13)/20 with
- PM1000 Rmax = (Vs-18)/20
- Output: 4 to 20 mA (2-Wire) with zero
- and span adjustments
- Accuracy: ±0.25% FS includes
- linearity, and hysteresis
- Repeatability: ±0.1% FS
- Long Term Stability (1yr): ±0.2% FS

- Media Temperature:
- -50 to 120°C (-58 to 248°F)
- Ambient Temperature:
- -40 to 80°C (-40 to 175°F)
- Thermal Zero Effect: <±0.016% FS/°F
- Thermal Span Effect: <±0.016% FS/°F
- Humidity: 95% RH non-condensing
- Maximum Overpressure: 200% FS
- Sensor: Silicon on Sapphire
- Wetted Parts: Titanium
- Case Material: Stainless Steel, IP65
- Pressure Port: 1⁄4" NPT male
- Electrical Connection: Din 43650
- plug mating connector supplied
- Response Time: 10 ms
- Weight: 90 g (3.2 oz)

2. Use in Thesis

The PX4200 pressure transmitter could be used to model a typical shipboard pressure sensor. However, due to time constraints the programming to incorporate data from the PX4200 was not completed. If it had been, the sensor would send a 4 to 20 mA current to the IPμ8930. The IPμ8930 then receives the signal as an input and reads it into memory where a PC could then access the sensor reading.

E. LINKSYS LAN ROUTER

The LINKSYS LAN router was used to connect IPμ8930 to a network capable computer.

1. Specifications

[From Ref. 12]

- IEEE specification: 10/100 fast ethernet
- Supports Dynamic and Static IP Addresses
- Creates a Firewall to Protect Your PCs From Outside Intruders

2. Operating Conditions

This is COTS available and has the ability to create a firewall for security purposes. For ease of use in this thesis the firewall was not used. However, the firewall and 802.11b standard provide for any security that might be required onboard a ship.

3. Use in Thesis

On board a ship the LINKSYS LAN Router would be connected to the Ship's LAN. It acts as the connection between the sensor with its IP address and the network capable computer.

F. NETWORK CAPABLE COMPUTER

In this thesis the Dell Inspiron 9300 laptop PC acts as the user interface to the system. In previous projects, it contained the wireless calibration software enabling the operating to calibrate the sensor wirelessly. Now that information is held on a server computer and accessed by the IPμ8930 with the sensor.

Figure 7. Dell Inspiron 9300 [From Ref. 13]

1. Specifications

[From Ref. 13]

- Processor: 1.73 GHz
- Operating system: Windows XP Professional
- RAM: 1.00 GB
- Networking cards: Internal wireless 802.11 a/b/g, 54Mbps

2. Use in Thesis

The PC was also used to write, store, and upload the html and java programs used on the IPμ8930.

G. SUMMARY

In this chapter, all the hardware that was used in this thesis has been described. The equipment included an Ipsil IPμ8930, a Linksys LAN router, and a Dell Inspirion 9300 laptop. The next chapter discusses and describes the software that these components use, as well as the designed html and java programs used to make a smart shipboard sensor network.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SOFTWARE

A. INTRODUCTION TO SOFTWARE

In this thesis, two sets of programs were developed. The first set consisted of one java program and one web page where the output of the Ipsil IPμ8930 was controlled through an input to the java program running inside of the web page. The second set of programs consisted of three java programs and three web pages. One of the web pages held a java applet program inside of it and could update and retrieve calibration constants from a server computer to the internal EEPROM of the IPμ8930 as well as calculate the value of the sensor reading after using the calibration constants. The programs are explained in detail later in the chapter, and are also shown in the Appendices.

B. IPSIL CONFIGURATION UTILITY

The Ipsil Configuration Utility or ICU is provided by the manufacture and is the primary interface between the network capable computer and the IPμ8930.

1. Role in Thesis

The ICU is used to configure the IPμ8930. Parameters such as IP Address, device password, and a DHCP (Dynamic Host Configuration Protocol) are able to be configured. For this thesis, the main purpose for the ICU was to upload the java programs, web pages, and any other associated files needed, such as pictures that the web pages use to the IPμ8930. Once a file is uploaded to the device into a memory location, it can then be accessed through a web browser such as Internet Explorer to access that file and the ICU is no longer needed. It is the ability of the IPμ8930 of allowing for the use of COTS software that is partly what makes using it so attractive in making a smart sensor. Another very important feature is the network connection; the IPμ8930 can be connected to a network of computers. The ICU allows for the ability to assign or change the IP

address of the IPμ8930. The IPμ8930 hardware along with the ICU software is what "net-enables" a sensor in this thesis.

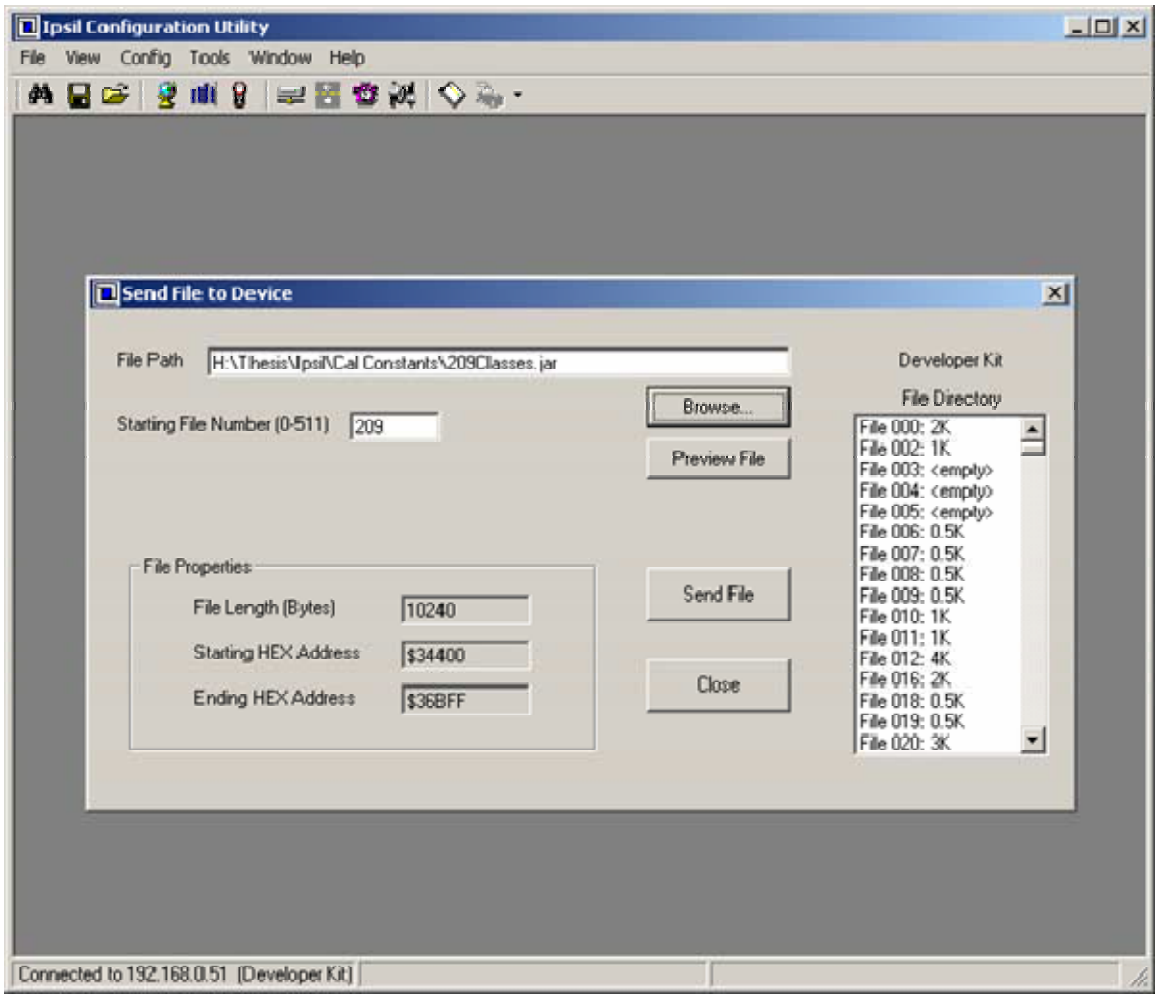

Figure 8. Ipsil Configuration Utility – Uploading a File.

2. Inputs and Outputs

The ICU can search for and connect to any Ipsil device that it finds on the network. Alternatively, a specific IP address may be entered for a connection to be made. When searching however, only devices that are within the same subnet network as the computer will be found, but if the exact IP address is known, a connection may be made outside of the local subnet. Once a connection has been established, configuration changes may be made. The ICU also has the ability to upload files to a specific location

in memory and download files to a specified directory on a computer. A java program will carry out all other inputs and outputs for this thesis. [Ref. 14]

3. Operating Procedures

When the ICU software is started the connection window automatically pops up. Here either a search for a device can be preformed or an IP address may be entered as well as a password for the device. After clicking on the, "OK" button the ICU access the device and forms a connection.

To upload a file, the command from the file menu is chosen and another dialog box appears on the screen as in Figure 8. The file path may be entered or the file chosen after clicking on the browse button. Files are stored on 512KB memory. This fact is important and will be explained in further detail later on in this chapter on how it affects the uploading and usage of web pages with java programs. A specific file location may be chosen (between 0 and 511 for a total of 512), however if a file name starts with the same number that its location will be, the Starting File Number input box will automatically be filled in. This feature makes it easy to remember where files are located in memory without having to either download from a memory location or use a web browser to see what is there. [Ref. 14]

To access a file location using a web browser the IP address along with the file number must be entered into the address location of the web browser. For example if a user wanted to see what the file in memory location 42 was, a user would enter http://192.168.0.51/042. If memory location 42 held a web page, then the web page would show up in the browser.

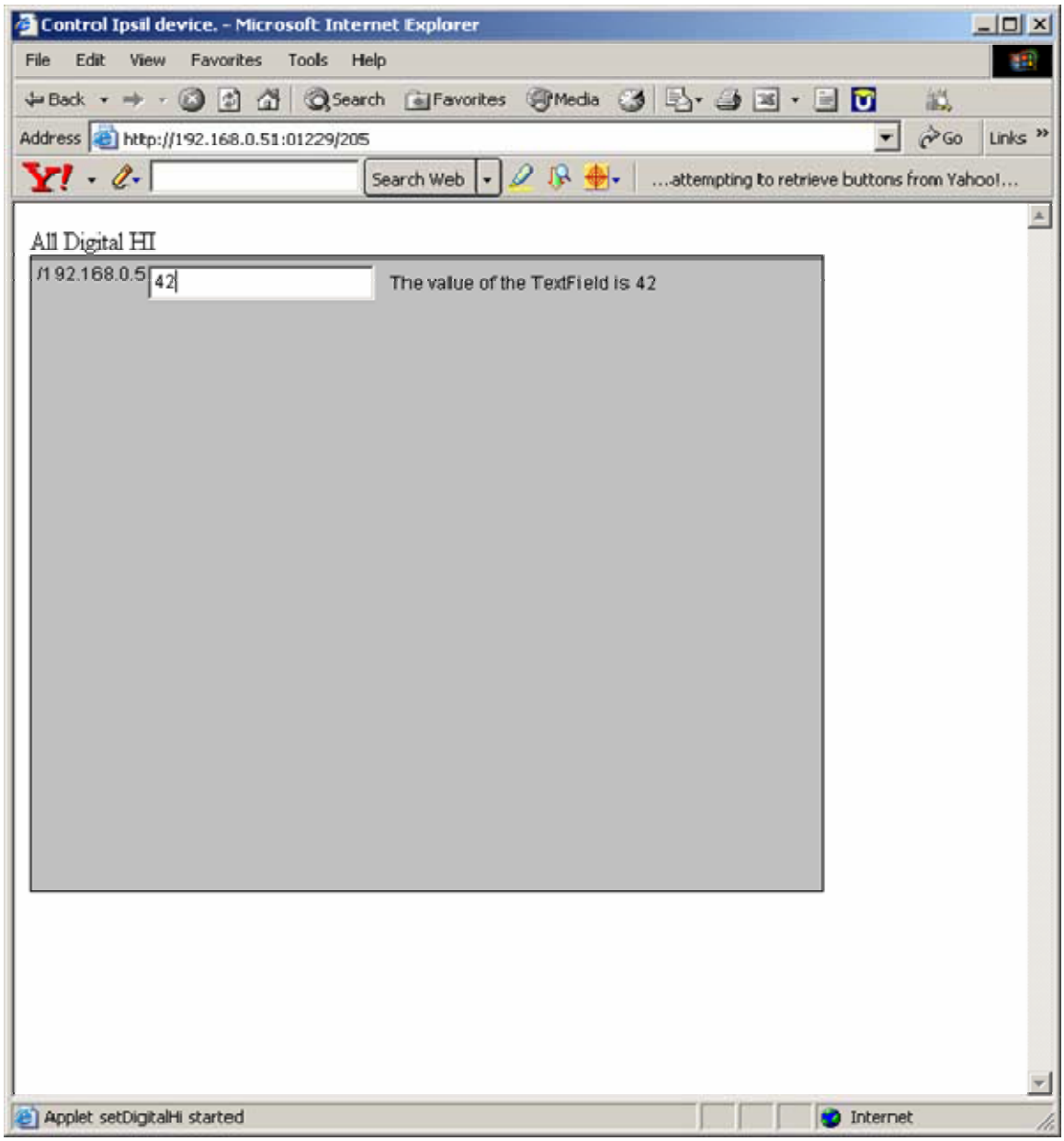

Figure 9. Web Page Showing Modified Version of setDigitalHi.java Applet.

C. USE OF JAVA

Java is an object-oriented programming language based on C and C++. It is described as a "web programming language" because it can be used to write programs, called "applets," that run within a web browser. This means that a web browser is required to execute Java applets. Applets permit more dynamic and flexible distribution

of information on a network. The language has what programmers call a clean design because only features that were essential were included. [Ref. 15]

1. Role in Thesis

The use of a java program or applet in this thesis is the principal software method for making a smart sensor smart. The applet takes input from a user and can either save this information to a memory location in the IPμ8930 or send it to the output/input pins as appropriate.

The development of the java applet programs in this thesis used the incremental software development process. In each cycle there was a design, code, and test step. Ipsil did the initial cycle and an example code provided from which this thesis used as a starting point and then further developed. This thesis does not take any credit for the development of the initial code, only for the changes made therein.

2. Inputs and Outputs

The command for creating an input for the first java applet was: sIn.read() and the command for and creating an output was: sOut.write(). The second java applet used a standard server-client relationship. This was accomplished using two java programs discussed later in this chapter. These two processes formed the basis for reading, writing, and storing data to and from the IPμ8930.

3. Java Codes

a. First Program

The first java program written for this thesis was based on a program developed by Ipsil. The java applet developed by Ipsil first set all the input/output pins,

shown on the bottom part of the IPμ8930 in Figure 3, as digital, then it made them all high. This program, called *SetDigitalHi.java*, did not require any input from a user in the java applet.

The first step taken in this thesis was to create a user input in the java applet so that the output could be changed to a number the user chose instead of making all the output pins high as in the example applet from Ipsil. The code for the java applet was very similar but now there was a textbox in the applet that had a place for a number to be inserted. The user could insert a number between 0 and 255 (for a total of 256) and the corresponding output display (seen in Figure 4) would show the hex value of the number the user entered. This was done by adding an object called changeLights(). Then the previously mentioned commands for sIn.read() and sOut.write() were called. Doing this allowed for the byte of input that the user entered to be sent to the output of the IPμ8930 and the user could see the result on the display board of LEDs built specifically for this thesis.

The display board seen in Figure 4 consisted of a 16 pin connector that attached to the input/output pins of the IPμ8930. The other end of the connector was split into 16 individual wires where there were eight pairs. The 16 pins on the IPμ8930 were paired to form one input/output pin and one ground pin. Each of these pairs was then connected to a small prototype board where each ground pin was connected to a ground line and each input/output wire connected to a LED. Next the LED connected to a resistor (to keep the LED from burning out) and then back to ground. In this way each LED would represent the on (high) or off (low) state of each input/output pin of the IPμ8930. When a user entered a number the LEDs would turn on according the the HEX value of that number. This java program is shown in Appendix A. Figure 10 is a block Diagram of the LED set-up, and Figure 10 shows the java applet running in the associated web page.

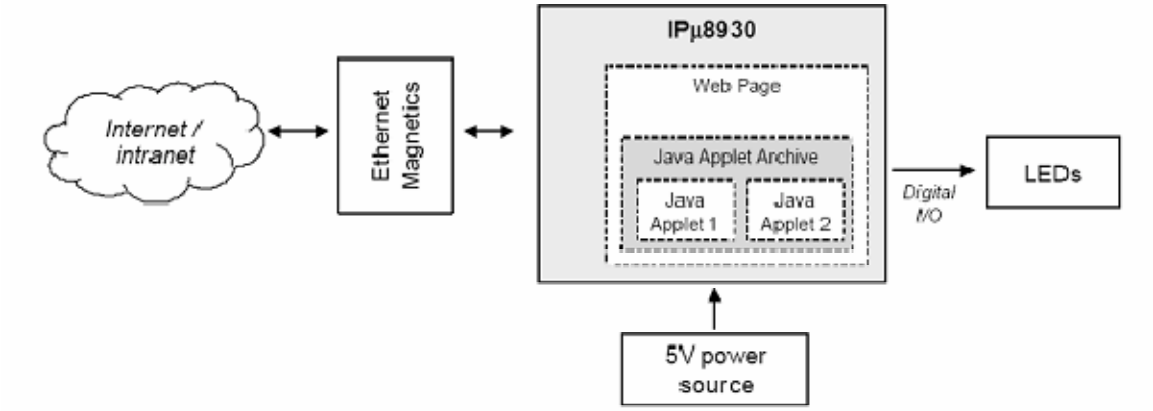

Figure 10. Block Diagram of LED Set-up. [From Ref. 16]

b. Second Program

The second java program developed for this thesis further modified the first java program but with a few key differences. The program, called *insertCalConstants.java*, did not write to an output display as the modified setDigitalHi program did. Instead, there were three places for inputs from a user. These were the calibration constants previously determined for a sensor and the value of the sensor reading itself. Once entered the values were stored in a text file located on a server and accessed by the applet. The calibrated value is calculated and displayed using the equation:

$$y = m * x + b$$

where the values of m and b were the calibration constants and x was the sensor reading value and y the calibrated sensor value. The values for m and b are updated by the IPμ8930 and displayed each time the applet is run. It is this key information being stored on the server and accessed by the client web page (discussed later in the chapter) in the EEPROM of the IPμ8930, combined with a sensor, which creates a smart sensor. The sensor now has its own IP address and the calibration constants for that sensor may be seen and accessed through any computer on the network through a web page by anyone

who knows the IP address of the sensor/IPμ8930 combination. This java program is shown in Appendix A and Figure 11 shows the java applet running in the associated web page.

c. ServerExample.java and ClientExample.java

These two programs, written by LT Carl Trask, set up a typical client-server relationship. When the SeverExample.java is running on the computer that is connected to the IPμ8930, it allows a connection to be made either by telnet or through the web page shown in Figure 11. The ClientExample.java is the program that access the server and either uploads or downloads the calibration constants m and b to it. The ClientExample.java is called by the insertCalConts.java and is part of the JAR archive package discussed later on in this chapter. Both java programs are shown in Appendix A.

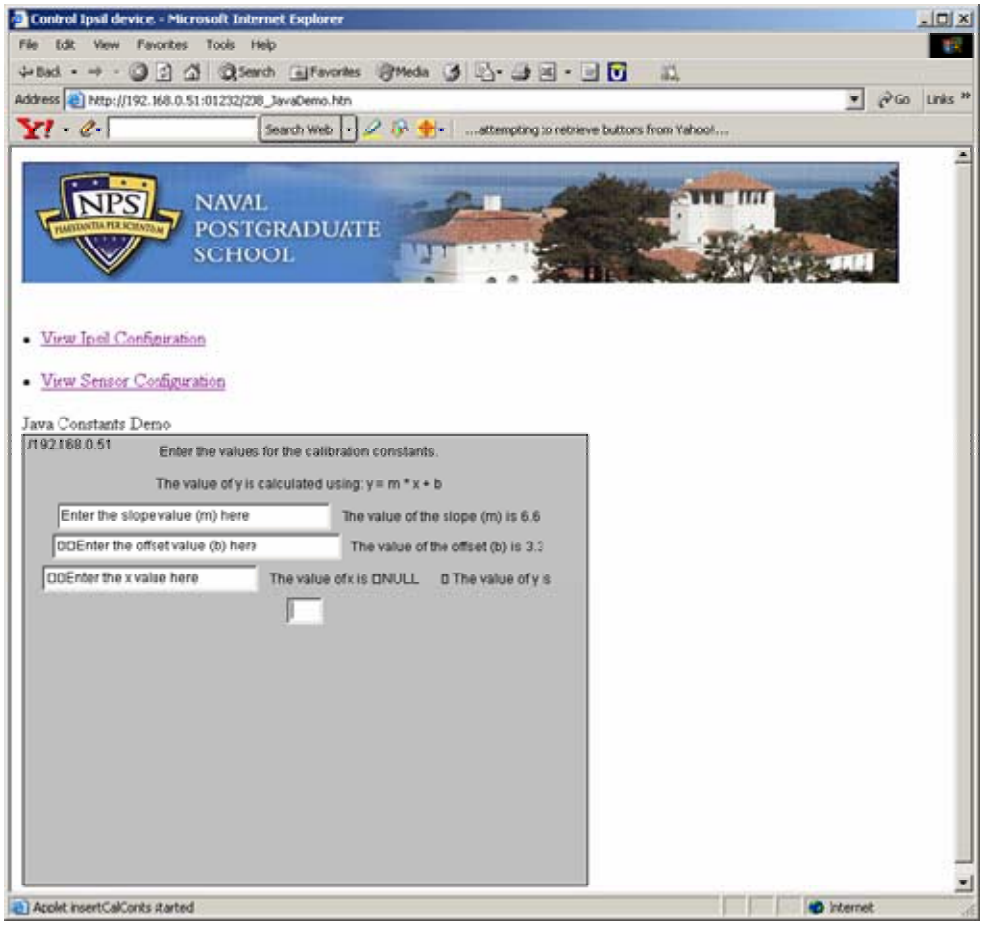

Figure 11. Web Page Showing insertCalConstants.java Applet.

D. USE OF HTML CODE

HTML or Hyper Text Mark-up Language was the programming code used to create the web pages for this thesis. The web pages themselves play an essential role in accessing, gathering, and displaying information. They are necessary for executing the java applets for without them the applets would not run, and provide a COTS programming and networking solution.

1. Role in Thesis

In this thesis the HTML based web pages are used to run the java applets, display, and gather information about the IPμ8930 and associated sensor. They also provide a means to access the IPμ8930's EEPROM.

2. Inputs and Outputs

Almost all of the inputs and outputs for this thesis are completed through java applets even though Ipsil has provided a way to use only html code and "web holes." However, this process is complicated and time consuming to develop. An example of using the html code with web holes is shown in Appendix B. It is a web page that Ipsil created which reads from the IPμ8930 how long it has been turned on for.

3. Web Pages

a. First Web Page

The first web page used with the modified *setDigitalHi.java* was the same web page that Ipsil developed with minor modifications to the references it used. The code is listed in Appendix B. There is no special or extraordinary html code used. It is simple and written for functionality. This does not mean that the web pages cannot be

more complex; however, for demonstration purposes in the lab there was no reason to use more than a functional code. The code calls for the use of javascript and gives the web page a name, "Control 8930. Set Digital Outputs Hi." Also, the html code writes an applet code that called, "SetDigitalHi.class" and calls the archive, "207Classes.jar" as well as setting the size of the applet area. More information on the *.jar archive will be presented in the next section of this chapter.

b. Second Set of Web Pages

1. Web Page 208. This web page is modeled after the one by Ipsil. However, it has been changed to show a few of the simple capabilities that html can provide. The first change was to incorporate a picture at the top of the web page. This picture is a jpeg file borrowed from the Naval Postgraduate School's web page and can be seen in Figure 11. It was uploaded into memory location 165 in the IPμ8930's EEPROM. To make this easier it was named, "065nps.jpg" and this can be seen in the code shown in Appendix B. The next major modification was to add two links under the picture. The links were to the following two web pages and can also be seen in Figure 11.

2. Web Page 020. This web page, shown in Figure 12, was developed by Ipsil and shows the status of the IPμ8930. The most important feature however is that it can read from the IPμ8930 the current running time and display it straight to the web page without having to use a java applet. This feature is called using "Web Holes" by Ipsil and was thoroughly investigated in the course of this thesis. It was determined that for simple processes using Web Holes is easier than using a java applet. However, if more than one read or write needs to be completed using Web Holes can become very complex and time consuming for a programmer. Using a java applet would then be recommended.

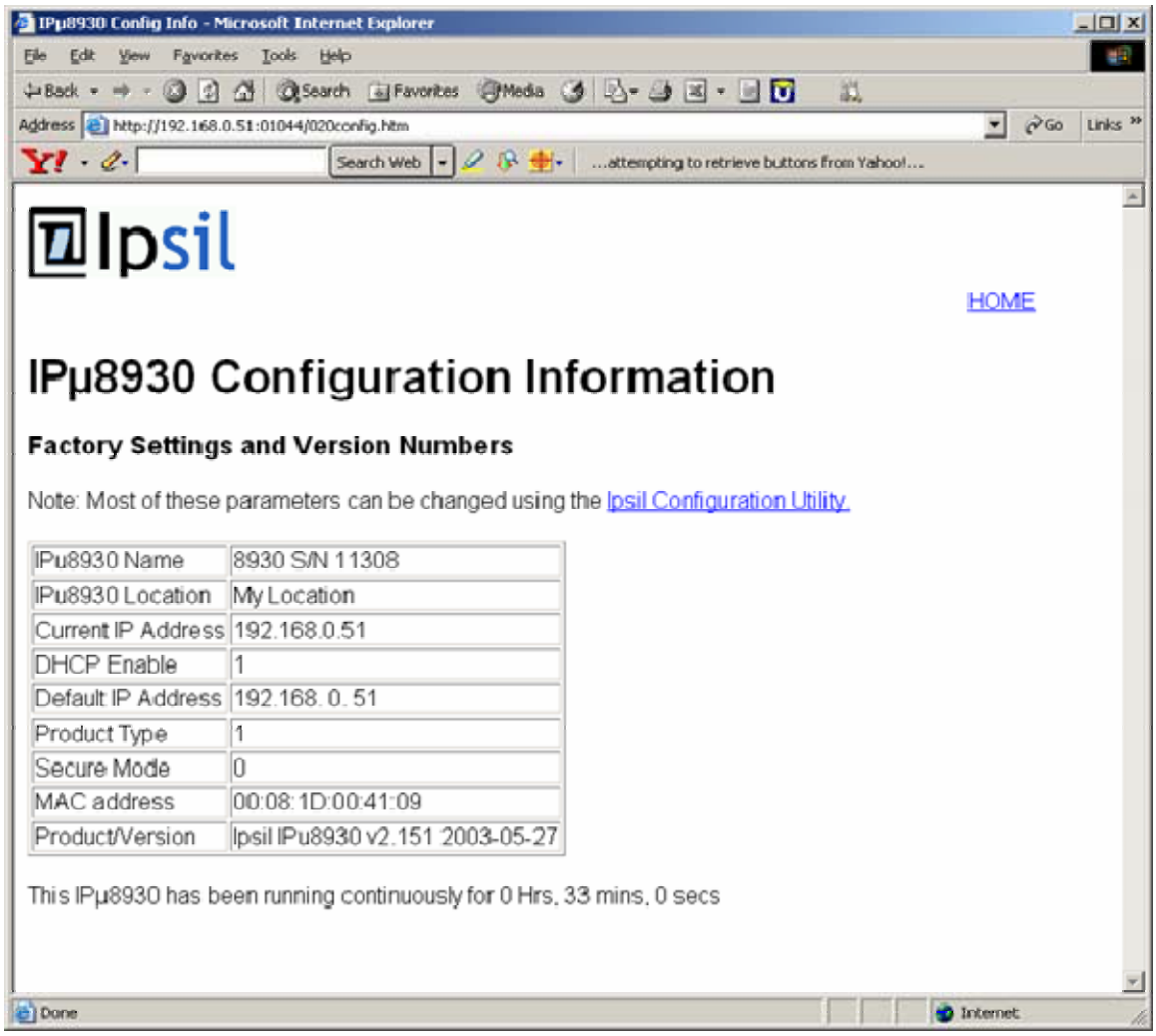

Figure 12. Web Page 020.

3. Web Page 160. The last web page, shown in Figure 13, created for this thesis was a modified version of Ipsil's web page in file 020 of the IPµ8930's EEPROM. It is shows information on the sensor that is connected to the input/output pins of the IPµ8930. This web page allows a user to not only read the current sensor reading but also shows other information that might be of importance to the user such as: sensor name, location, current IP address, default IP address, product type, secure mode, MAC address, and product/version. Having a web page such as this plays a key role in the development of a shipboard sensor network because it allows for information access flow to anyone on the network that may require it.

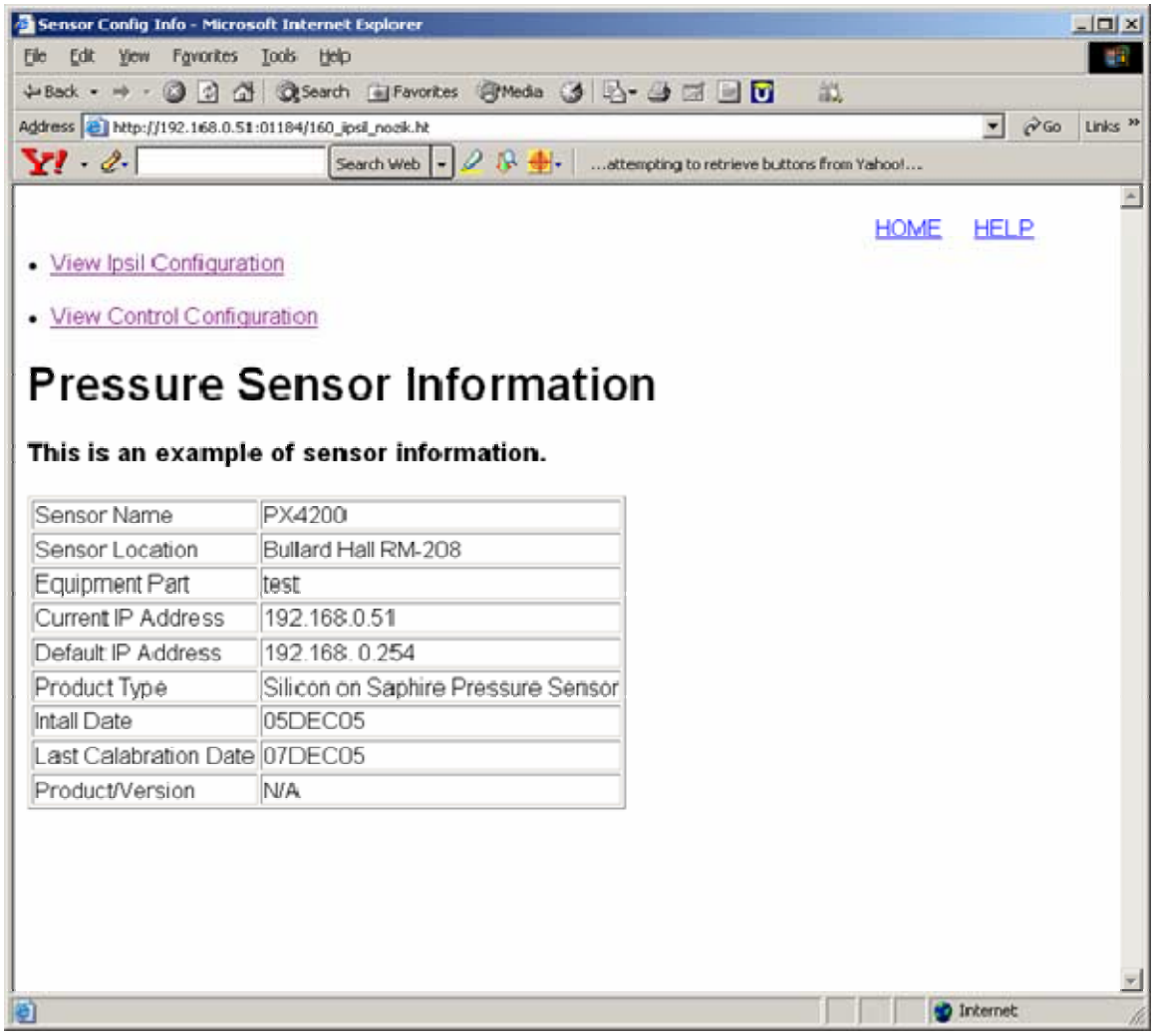

Figure 13. Web Page 160.

E. UPLOADING AND ARCHIVING: THE JAR COMMAND

Unlike the process for uploading a web page the process for uploading a Java program is a little more complex. The use of a java applet archive file (JAR) must be included and uploaded separately in order for the Java applet to work correctly with the input/output resources of the IPμ8930. Java applets can only establish a TCP/UDP connection with the same device that the applet is being run on in order to access read and write commands. For this thesis this process is done between the IPμ8930 and the computer running the server program.

A JAR file is a compressed archive of Java applets and other files. It is used for two reasons. The first is that it takes up less space in memory, and the second is that since class names cannot start with a number, the use of a JAR file allows the programmer to use an associated number in the file name to memory location as discussed in the operating procedures of the ICU.

There were three steps taken in order to create the JAR files used for this thesis. The first step was to change the computer's path environmental variable so a JAR command could be used in any directory while using a command prompt. The second step was to use the command, "jar cfv 209Classes.jar *.java *.class *.htm save.txt." This command created a JAR file in the current directory that held all of the java programs, class files, web pages, and text files. The actual files in 209Classes.jar were inseretCalConts.java, ClientExamle.java, insertCalConts.class, ClientExample.class, 208_JavaDemo, and save.txt.

The size of the newly created JAR file must be a multiple of 512 bytes in order for the file to store correctly in the IPμ8930. This is because the file sytem clusters in the IPμ8930 are 512 byte blocks. When a JAR file is served to a client web browser, a data stream that is a multiple of 512 bytes is returned. If the JAR file is not exactly a multiple of 512 bytes there will be extraneous trailing bytes that will also be sent to the client web browser. This may cause a problem in that the JAR file could be seen as a potential virus by the client web browsers. The answer to this problem is to create a JAR files that is exactly a multiple of 512 bytes in size. This is called adding padding to the JAR file and is done by adding an uncompressed text file that contains as many spaces as needed to make the JAR file to a 512 byte multiple. [Ref. 16]

Adding the padding to the JAR file is the third and final step taken to create the JAR files used for this thesis. The command used was, "jar ufv0 209Classes.jar trailer.txt." This command attached the trailer text file that held a previously determined number of spaces in order to have the 209Classes.jar file be a multiple of 512 bytes. Figure 14 shows the relationship between the web pages, java code, and JAR file.

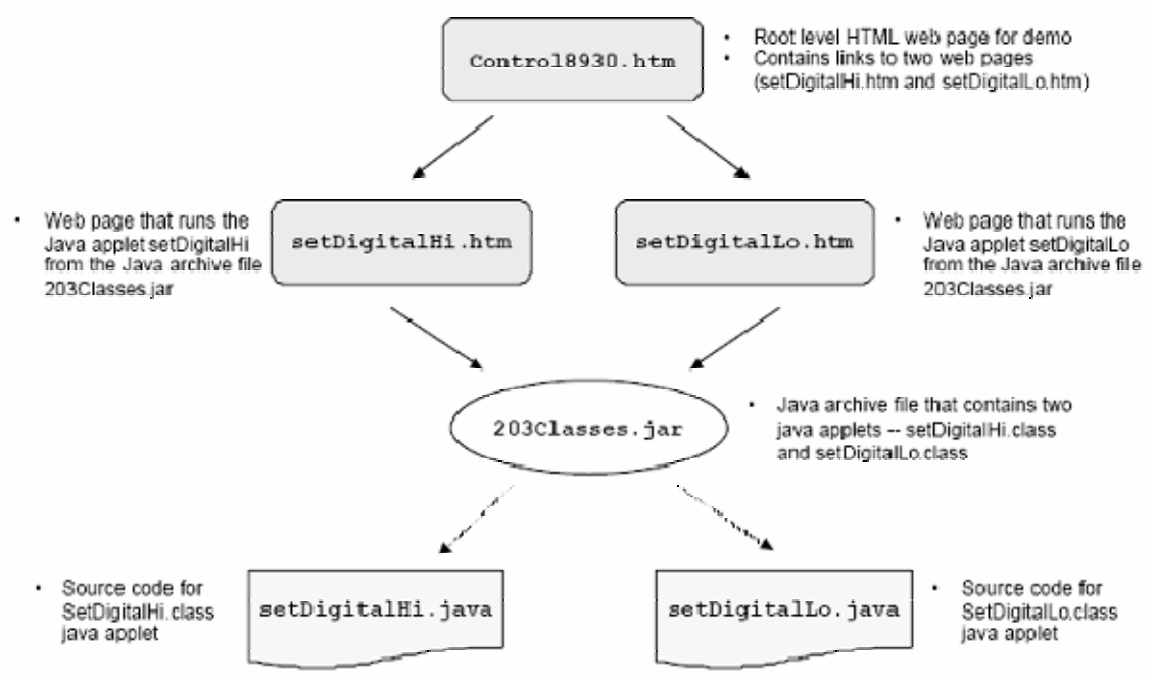

Figure 14. Summary of Relationship Between Various Demo Files. [From Ref. 16]

F. SUMMARY

This chapter discussed in detail the different Java programs and web pages that were used or developed for this thesis. The insertCalConts.java program is the applet located on the IPμ8930. The ServerExample.java program is the primary data acquisition program located on the network computer and enables the operator to write or read the calibration constants from the web page with the java applet. The JAR file is a special archive that allows the java programs to be run from the IPμ8930 web server. All of these programs coordinate in order for a user to send and receive the calibration constants to and from the IPμ8930/sensor combination.

The next chapter focuses on the results that were found in the development of the java programs and associated web pages. The chapter discusses the lessons learned and talks about how suitable and capable the IPμ8930 is.

V. RESULTS AND LESSONS LEARNED

A. INTRODUCTION

Previous chapters have covered the theory, software, and hardware used to design and implement a smart sensor. This chapter sequentially analyzes the final product and design process from start to finish.

B. THE CONCEPT

From the start of this thesis project, the center of attention was on how to build a "smart "sensor. The idea of what a "smart" sensor was first needed to be defined. After a few days of deliberation and research it was decided that a smart sensor should enable users and systems integrators to automatically configure their measurement automation and network systems. More specifically for this thesis, it would be the ability for a sensor to store its own calibration constants as well as any applicable network information. Next was to choose the hardware that could accomplish this task. The Ipsil IPμ8930 is a small TCP controller module that contains everything needed to add internet-based monitoring and control to an application. Connecting an IPμ8930 to a sensor along with the appropriate software would create a "smart" sensor. [Ref. 8 and 17]

C. THE BEGINNING

At first, it was thought possible to use an Ipsil IPμ8930 chip and a single web page that would hold both the network information and the sensor calibration constants. This process would use, "web holes" (discussed in the previous chapter). The sensor calibration constants would reside in the EEPROM of the IPμ8930 and could be accessed by a user who knew the IP address of the IPμ8930/sensor combination thereby creating a

"smart" sensor. However, it soon became apparent that the use of web holes was complicated and time consuming. Another way needed to be found if the use of the IPμ8930 was to be continued.

Fortunately, Ipsil had also provided an example of using Java with the IPμ8930. Their example was simple and showed how a Java applet could be written and uploaded in a JAR file to be used inside of another uploaded web page. The Java applet could control the input/output pins of the IPμ8930 by either turning them all high (on) or low (off).

The Java example provided by Ipsil was then taken one step further by having a user input a number into the java applet on the web page and then set the input/output pins to the HEX value of that number. This required a modification to the java applet but not the web page it was being used on. This program is listed in Appendix A as, *setDigitalHi.java*. The process of developing and uploading the "First Java Program" had a high learning curve, however once the process was completed, repeating it became much easier.

After understanding the process for creating a JAR file and the interaction between it and the IPμ8930, the next step was to develop the Java applet that could store sensor calibration constants, make calculations using them, and display the result. It was thought at first that this could be done just using a text file that would be included in the JAR file that the Java applet could read/write from. In this way the calibration constants would reside in the IPμ8930's EEPROM. Although this is possible, it will be shown in the next section why this was not done. The *insertCalConts.java* is the final program designed in this thesis for making a "smart" sensor and is shown in Appendix A.

D. THE ADJUSTMENT PHASE

As mentioned in the previous section, using a JAR file with an included text file to store calibration constants by itself was not done. The reason for this was because in taking another look at the objective of this thesis and its benefit to the Navy, it made more sense to have the calibration constants stored on a server computer as well the text

file. This way if power is lost to the IPμ8930/sensor combination, the calibration constants are saved on the server computer and can be accessed by the IPμ8930/sensor when the java applet is run. In addition, it is now possible to keep a record of what the sensor calibration constants are and send this information offboard the ship for evaluation.

The method for implementing this was to create two new java programs. With the invaluable help of LT Carl Trask, the *ClientExample.java* and *ServerExample.java* programs were written. They are shown in Appendix A. The combination of both of these programs allows for the java applet in *insertCalConts.java* to send the calibration constants to the computer running the server program. Now, the calibration constants are held in both the text file in the JAR file that resides in the IPμ8930's EEPROM and on a server computer. Figure 15 is an information flow diagram showing how information from the java applet is sent to the IPμ8930/sensor combination and then on to the server computer.

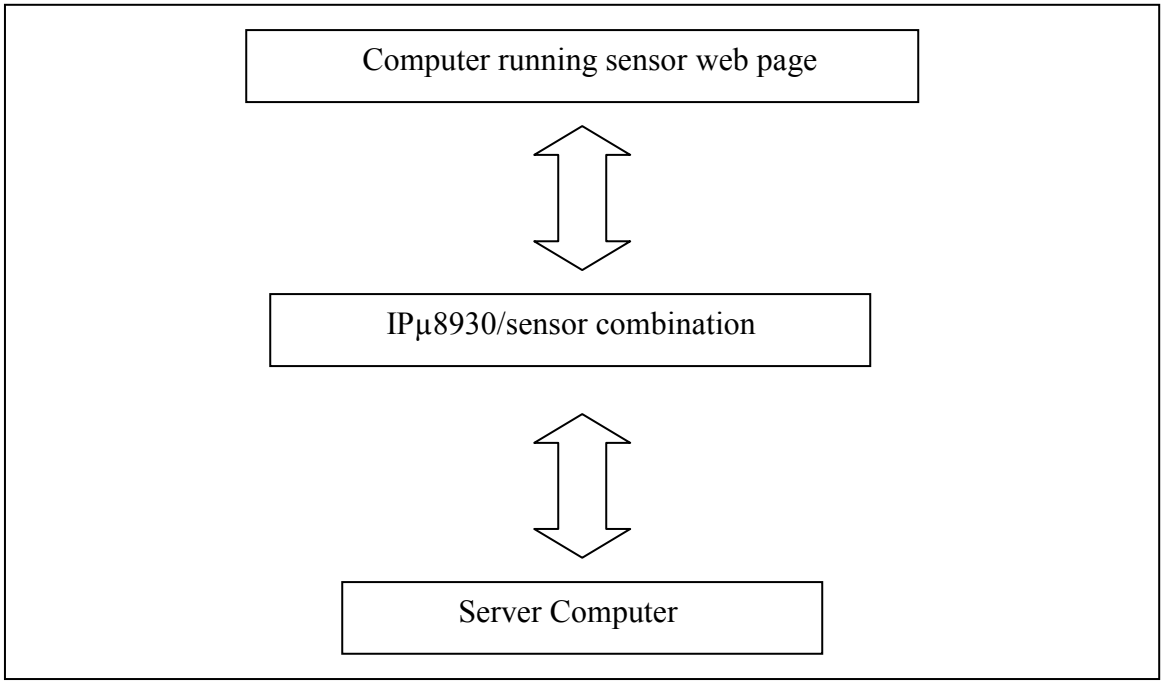

Figure 15. Information Flow Diagram.

The server/client relationship intent is such that the java applet is the client, however it is also possible to telnet into the server computer if the IP address is known along with the correct port and sever ID. Doing this allows a user to either retrieve the calibration constants or update them. When the java applet is restarted by either refreshing the web page or switching from another web page and back, it automatically retrieves the current calibration constants from the server computer and displays them.

E. THE FINISHING TOUCHES

Most of the effort for this thesis went to making the Java programs and JAR archive files work correctly with the IPμ8930. The web pages created were roughly designed and for demonstration purposes only. Final finishing touches completed to the web pages included making them more user friendly and more professional in appearance. Lastly, the text boxes inside the Java applet were adjusted so that they appeared in rows. Figure 16 is a picture of the IPμ8930 microcontroller attached to the PX4200 pressure sensor.

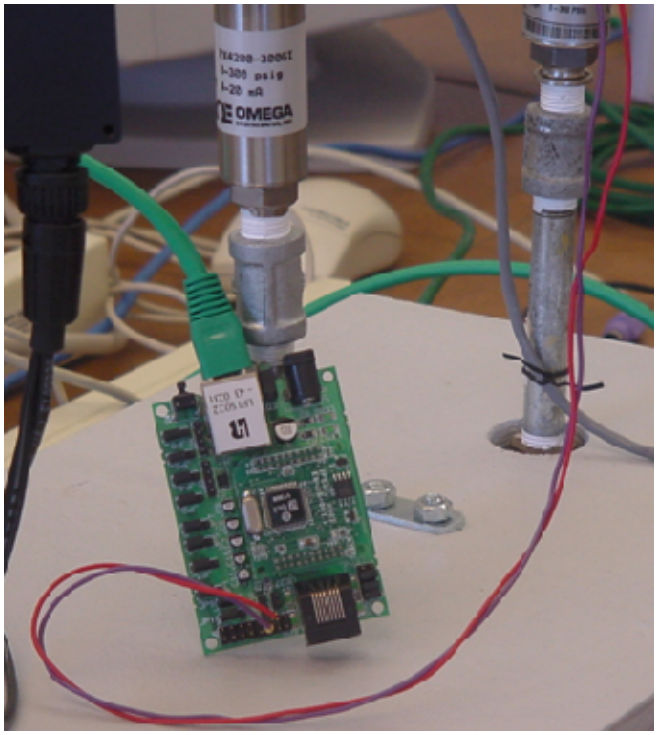

Figure 16. IPμ8930 With Sensor and Network Connections.

F. LESSONS LEARNED

There were five main lessons learned from this thesis project.

- Using Ipsil's Web Holes is time consuming and tedious.
- Web pages are simple to write and can be made user friendly.
- Java is a very adaptable and useful programming language.
- Different Java versions would not work together.
- Newer versions of Java would not work with the IPμ8930

1. Using Web Holes

Ipsil has developed a method for integrating HTML straight into the IPμ8930's EEPROM. This method while at first seemed to be an easy and reliable way to incorporate information from a sensor was in fact complicated and time consuming to implement. The reason for this is because the exact space number of the "hole" in the HTML code being used needs to be known in order to have the web hole work correctly. If the space number is even off by one not only will that hole not work, but any holes following it will not work as well. Therefore it was found that the use of a Java applet was much easier to implement with the HTML code.

2. Using HTML

The internet has exploded over the past decade and along with it the programming language that made it happen. HTML has become the primary means to writing web pages, is easy to learn, and has the ability to be very adaptable to many types of applications. In the case of this thesis it is used as a shell to hold information and implement Java applets. The focus on this thesis was the Java programming and not the HTML, however future work should focus more on the use of HTML as will be discussed in the next chapter.

3. Using JAVA

Java is an extremely flexible programming language that can adapt to multiple hardware devices. The fact that it is an object oriented programming language makes it very versatile and because it is a COTS available makes it perfect for developing a smart sensor without the use of proprietary software.

4. Different Versions of JAVA

Java, however, is not without its downsides as with all programs. As great as it is as a programming language, it gets more complicated because a run time environment is required that is separate from the developer platform. There were problems at first with using an older version of the developer platform with a newer addition of the runtime environment. However, when both were switched to either new or old there were no more problems.

5. IPμ8930 and JAVA

As discussed above there were a few problems encountered with the different versions of Java. Part of the reason those problems were encountered is because the IPμ9030 would only work with older versions of Java when compiling and creating JAR files. Once uploaded a computer with a newer version of the runtime environment could access and use the Java applet stored on the IPμ8930, but the process of compiling the Java programs and JAR files required an older version. However, the older versions were available from www.java.sun.com at no cost and it is believed that newer models of the IPμ8930 from Ipsil will be able to work with the never versions of Java.

G. SUMMARY

The entire process from start to finish has been presented in this chapter. The next chapter discusses the conclusions that were developed from the research and any suggested future work that for completion.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

So far this thesis has covered the theory, hardware, and software for a smart shipboard sensor network. This chapter contains conclusions of the research and recommendations for future work.

B. CONCLUSIONS

As previously stated the goal of this thesis was to develop a working prototype smart sensor for a shipboard environment. This goal was partially achieved in that a working prototype smart sensor was developed. However, it was for lab use only and not for a shipboard environment. That part of the goal was not accomplished because the equipment needed to achieve it was not available for purchase in a timely fashion for the completion of this thesis.

C. RECOMMENDATIONS FOR FUTURE WORK

The product of this research is a prototype only. It is only intended to be used in a laboratory environment. Future work should center on using Ipsil's Wireless Sensor Block which hardens the IPμ8930 to be used in a shipboard environment and makes it wireless. Also, the software should be improved so it is more user friendly and can integrate with the ICAS technology. Lastly, a better method for storing the calibration constants on the server computer should be developed. This will make the process of transferring the calibration information off-board ship easier and more organized so that examination of the data can provide useful results for improving the sensors accuracy and the associated equipment behavior.

1. Using Wireless SensorBlock

The use of Ipsil's Wireless SensorBlock will allow for the ability to not have extensive wiring in shipboard engineering spaces. The SensorBlock uses the IEEE 802.11b wireless standard and can be used in the same way that this thesis has developed for the IPμ8930. They are low-cost and especially designed to monitor a wide range of sensors.

2. Improving Software

There are several improvements that should be implemented in reference to the developed software for this thesis. The first improvement recommended is to make the web page more user friendly. Java applets are very versatile and may be applied to a wide variety of applications. Future versions of the software should include options that will make the software more user friendly. An example would be a "submit" button that could be clicked after the calibration constants have been entered. Also, it should be possible to incorporate the information provided by several sets of IPμ8930/sensor combinations into one central user friendly location. This would also serve as a database showing where all the installed IPμ8930/sensor combinations are located and on which piece of equipment.

3. Integrating With ICAS

Future naval ships will include a single Integrated Conditioning Assessment System (ICAS) that will encompass their entire suite of engineering systems. Integrating the technology derived from this thesis into ICAS will improve the overall efficiency of the engineering plant and awareness.

4. Improving Information Storage

The ability to transmit sensor calibration data off the ship to a shore command such as SPAWAR or NAVSEA would be an important step towards the ability to make speedy improvements to the sensors and equipment that naval ships use. Fleet maintenance is a very important process towards keeping ships ready to accomplish their missions. In order to transmit sensor calibration data off the ship it needs to be stored in an organized fashion that is useful for analysis. This analysis would help in making better decisions in the type of shipboard equipment used as well as sensor types.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. JAVA CODES

A. FIRST JAVA PROGRAM: SETDIGITATHI.JAVA

```
import java.applet.Applet;
import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;

 public class setDigitalHi extends Applet implements ActionListener
{
// Constants
   private int IPSIL_PORT = 8930;          // Ipsil device port number
   private String tfString = "NULL";
   private TextField tf;
   private Label lb;
   private byte by;

// Variables

   InetAddress       ia      = null;
   Socket            sock    = null;       // Socket
   InputStream       sIn     = null;       // Input data stream
   OutputStream      sOut    = null;       // Output data stream instance

   StringBuffer buffer = new StringBuffer();

// Arrays
// Write RAM variable ICP command
// Configures all channels as Digital Outputs
// Operation code = 0x06, address in RAM 0x113, bytes to write 2
   byte [] setAllDigitalOutput =
   { 0x00, 0x00, 0x00, 0x00, 0x00,  // Password                    (5 bytes len)
     0x06,                          // "Write EEPROM" cmd          (1 byte len)
     0x00, 0x00, 0x01, 0x13,                // Start location      (4 bytes len)
     (byte)0x86,                    // All channels          Digital (1 byte len)
     0x00,                          // All channels as Outputs (1 byte len)
     };

// Set all digital outputs to HI level
// Write RAM variable ICP command
// Operation code = 0x06, address in RAM 0x117, 1111 1111 - all channels "1"
output
```

```
// Each bit of the last byte represents each channel state accordinly
  byte [] setAllDigitalHi =
  { 0x00, 0x00, 0x00, 0x00, 0x00,  // Password                  (5 bytes len)
    0x06,                          // "Write EEPROM" cmd        (1 byte len)
    0x00, 0x00, 0x01, 0x17,        // Start location            (4 bytes len)
    (byte)0xff,                    // All channels HI           (1 byte len)
    };

  byte [] resp = new byte[1024];    // Network response buffer

   public void init()
   {
     try
     {

     //ia = InetAddress.getByName("192.168.1.101");
     //ia = InetAddress.getLocalHost();
     //addItem(ia.getHostAddress());
       ia = InetAddress.getByName( getCodeBase().getHost() );

     // Init TCP socket. Connect to the device itself on Ipsil port
       sock = new Socket(ia, IPSIL_PORT);
       addItem(ia.toString());

     // Create input data stream
       sIn  = sock.getInputStream();

     // Create output data stream
       sOut = sock.getOutputStream();

       tf = new TextField("Enter Number Here");
       lb = new Label (" The value of the TextField is " + tfString);
       add(tf);
       add(lb);
       tf.addActionListener(this);
       repaint();

     }

       catch (Exception e)
       {
         e.printStackTrace();
         addItem(" Init " );
       }

   }
```

```
 private void addItem(String newWord)
{
  System.out.println(newWord);
  buffer.append(newWord);
  repaint();
}

 public void paint(Graphics g)
{
//Draw a Rectangle around the applet's display area.
  g.drawRect(0, 0, getSize().width - 1, getSize().height - 1);

    //Draw the current string inside the rectangle.
  g.drawString(buffer.toString(), 5, 15);
}

 public void actionPerformed (ActionEvent e){

  try{
    by = (byte) Integer.parseInt(tf.getText());
    lb.setText(" The value of the TextField is " + by );
    setAllDigitalHi[10]=by;
    changeLights();
  }
     catch (NumberFormatException ne){
       lb.setText(" Please insert an Int ");
     }

  repaint();
}


 public void start()
{
    changeLights();
}

 private void changeLights() {
  try
  {
  // Send ICP command - Configure all channels as digital outputs
    sOut.write(setAllDigitalOutput);

  // Read and ignore replay byte from the Ipsil device
```

```
            sIn.read(resp);

          // Send ICP command - Set HI all on all of the channels
            sOut.write(setAllDigitalHi);

          // Read and ignore replay byte from the Ipsil device
            sIn.read (resp);
          }

             catch (Exception e)
            {
              e.printStackTrace();
              addItem(" Start ");
            }
        }
      } //end of the applet
```

B. SECOND JAVA PROGRAM: INSERTCALCONTS.JAVA

```
import java.applet.Applet;
import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;

 public class insertCalConts extends Applet implements ActionListener
{
// Constants
   private int IPSIL_PORT = 8930;          // Ipsil device port number

                private String tfString = "NULL";
                private String tffString = "NULL";
                private String tfffString = "NULL";

                private TextField tf, tff, tfff, tf_y;

                private Label comment_one, comment_two, lb, lbb, lbbb, lb_y;

                private byte by, byy, byyy;

                private int m,b,x,y;

// Variables
   InetAddress          ia     = null;
   Socket               sock   = null;          // Socket
```

```
  InputStream       sIn     = null;          // Input data stream
  OutputStream      sOut    = null;          // Output data stream instance

  StringBuffer buffer = new StringBuffer();


// Arrays
// Write RAM variable ICP command
// Configures all channels as Digital Outputs
// Operation code = 0x06, address in RAM 0x113, bytes to write 2
  byte [] setAllDigitalInput =
  { 0x00, 0x00, 0x00, 0x00, 0x00,  // Password                          (5 bytes len)
    0x06,                          // "Write EEPROM" cmd                (1 byte len)
    0x00, 0x00, 0x01, 0x13,        // Start location                    (4 bytes len)
    (byte)0x86,          // All channels Digital (1 byte len)
    0x00,                // All channels as Outputs (1 byte len)
    };

// Set all digital outputs to HI level
// Write RAM variable ICP command
// Operation code = 0x06, address in RAM 0x117, 1111 1111 - all channels "1" output
// Each bit of the last byte represents each channel state accordinly
  byte [] setAllDigitalHi =
  { 0x00, 0x00, 0x00, 0x00, 0x00,     // Password                          (5 bytes len)
    0x06,                             // "Write EEPROM" cmd                (1 byte len)
    0x00, 0x00, 0x01, 0x17,           // Start location                    (4 bytes len)
    (byte)0xff,                       // All channels HI                   (1 byte len)
    };

  byte [] resp = new byte[1024];     // Network response buffer

   public void init()
   {
     try
     {

     //ia = InetAddress.getByName("192.168.1.101");
     //ia = InetAddress.getLocalHost();
     //addItem(ia.getHostAddress());

        ia = InetAddress.getByName( getCodeBase().getHost() );

     // Init TCP socket. Connect to the device itself on Ipsil port
        sock = new Socket(ia, IPSIL_PORT);
        addItem(ia.toString());
```

```
// Create input data stream
   sIn  = sock.getInputStream();

// Create output data stream
   sOut = sock.getOutputStream();

   comment_one = new Label ("Enter the values for the calibration constants.");
   comment_two = new Label ("The value of y is calculated using: y = m * x + b");
   add(comment_one);
   add(comment_two);
   repaint();

   tf = new TextField("Enter the slope value (m) here\n");
   lb = new Label (" The value of the slope (m) is " + tfString);
   add(tf);
   add(lb);
   tf.addActionListener(this);
   repaint();

   tff = new TextField("\nEnter the offset value (b) here\n");
   lbb = new Label (" The value of the offset (b) is " + tffString);
   add(tff);
   add(lbb);
   tff.addActionListener(this);
   repaint();

   tfff = new TextField("\nEnter the x value here\n");
   lbbb = new Label (" The value of x is \n" + tfffString);
   add(tfff);
   add(lbbb);
   tfff.addActionListener(this);
   repaint();

   lb_y = new Label ("\n The value of y is ");
   tf_y = new TextField("\n");
   add(lb_y);
   add(tf_y);
   repaint();
}

    catch (Exception e)
   {
     e.printStackTrace();
     addItem(" Init " );
   }
```

```
}


 private void addItem(String newWord)
{
  System.out.println(newWord);
  buffer.append(newWord);
  repaint();
}

 public void paint(Graphics g)
{
//Draw a Rectangle around the applet's display area.
  g.drawRect(0, 0, getSize().width - 1, getSize().height - 1);

    //Draw the current string inside the rectangle.
  g.drawString(buffer.toString(), 5, 15);
}

 public void actionPerformed (ActionEvent e){

  try{
    by = (byte) Integer.parseInt(tf.getText());
    lb.setText(" The value of the slope (m) is " + by );
    setAllDigitalHi[10]=by;
    changeLights();
  }
     catch (NumberFormatException ne){
       lb.setText(" Please insert an Int ");
     }

    try{

      byy = (byte) Integer.parseInt(tff.getText());
      bb.setText(" The value of the offset (b) is " + byy );
      setAllDigitalHi[10]=byy;
      changeLights();
  }
     catch (NumberFormatException ne){
    lbb.setText(" Please insert an Int ");
     }
  try{

    byyy = (byte) Integer.parseInt(tfff.getText());
    lbbb.setText(" The value of x is " + byyy );
    setAllDigitalHi[10]=byyy;
```

```
      changeLights();
    }
       catch (NumberFormatException ne){
       lbbb.setText(" Please insert an Int ");
      }

    repaint();

      m = (int) Integer.parseInt(tf.getText());
      b = (int) Integer.parseInt(tff.getText());
      x = (int) Integer.parseInt(tfff.getText());

      y = m * x + b;

      tf_y.setText("\n " + y);
      tf_y.setEditable(false);

}


 public void start()
{
     changeLights();
}

 private void changeLights() {
   try
   {
   // Send ICP command - Configure all channels as digital outputs
    sOut.write(setAllDigitalInput);

   // Read and ignore replay byte from the Ipsil device
    sIn.read(resp);

   // Send ICP command - Set HI all on all of the channels
    sOut.write(setAllDigitalHi);

   // Read and ignore replay byte from the Ipsil device
    sIn.read (resp);
   }

       catch (Exception e)
      {
        e.printStackTrace();
        addItem(" Start ");
      }
```

```
    }
  } //end of the applet
```

C. SERVEREXAMPLE.JAVA

```
import java.net.*;
import java.io.*;

public class ServerExample {

   private ServerSocket    serverSocket;
   private Socket          clientSocket;
   private PrintWriter     out;
   private BufferedReader  in;

   private String          inputLine;
   private String          outputLine;

   private double          m;
   private double          b;

   public static final int M = 1;
   public static final int B = 2;
   public static final int ERROR = 0;
   public static final int RETRIEVE = 3;
   public static final int UPDATE = 4;


   public ServerExample(){
      try{
         init();
         doStartComms();
         getClientID();

         boolean flag = true;
         double send = 0;

         while(flag){
            int command = getCommand();
            switch(command){
               case UPDATE:
                  doGetVariableToUpdate();
                  flag = !doUpdate();
                  break;
```

```
            case RETRIEVE:
               doRetrieve();
               flag = false;
               break;
            case ERROR:
               flag = true;
               continue;
         }
      }
      doSendDone();
      exit();
   }catch (IOException io){
      System.out.println(io);
   }
}

public static void main(String[] args) throws IOException {

   while (true){
      ServerExample se = new ServerExample();
   }
}

private void init() throws IOException{

   serverSocket = null;
   try {
      serverSocket = new ServerSocket(1234);
   } catch (IOException e) {
      System.err.println("<SERVER> Could not listen on port: 1234.\n");
      System.err.println(e);
      System.exit(1);
   }

   clientSocket = null;

   try {
      clientSocket = serverSocket.accept();
   } catch (IOException e) {
      System.err.println("Accept failed.");
      System.exit(1);
   }
   out = new PrintWriter(clientSocket.getOutputStream(), true);
   in  = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

}
```

```
private void exit() throws IOException{
   out.close();
   in.close();
   clientSocket.close();
   serverSocket.close();

}

private String readData() throws IOException{
   inputLine = "";
   String tmp;
   try{
      Thread.sleep(25);
   }catch (Exception e){

   }

   while (!in.ready()) {
      try{
         Thread.sleep(25);
      }catch (Exception e){

      }
   }

   while (in.ready()) {
      tmp = in.readLine();
      System.out.println("<SERVER Received> " + tmp);
      inputLine += tmp;
   }
   return inputLine;
}

private void writeDataString(String outString){
   System.out.println("<SERVER Sent>" + outString);
   out.println(outString);
   out.flush();
}

private void writeDataDouble(double outDouble){
   System.out.println("<SERVER Sent>" + outDouble);
   out.print(outDouble);
   out.flush();
}
private void doStartComms(){
```

```
    writeDataString("Welcome to the custom variable server");
}

private int getClientID() throws IOException{
    writeDataString("Enter the server ID");
    String tmp = readData();
    return Integer.parseInt(tmp.trim());
}

private int doGetVariableName() throws IOException{
    writeDataString("Enter the Variable.");
    String tmp = readData();
    if (tmp.trim().equalsIgnoreCase("M")){
        return M;
    }else if (tmp.trim().equalsIgnoreCase("B")){
        return B;
    }else{
        return ERROR;
    }
}

private void doSendVariable(double dataOut){
    writeDataDouble(dataOut);
}
private int getCommand()throws IOException{
    writeDataString("Press U for Update and R for Retrieve.");
    String tmp = readData();
    if (tmp.equalsIgnoreCase("U")){
        return UPDATE;
    }else if (tmp.equalsIgnoreCase("R")){
        return RETRIEVE;
    }else
        return ERROR;
}

private boolean doUpdate()throws IOException{
    try{
        writeDataString("Send the value for M");
        String mstr = readData();
        double m = Double.parseDouble(mstr);

        writeDataString("Send the value for B");
        String bstr = readData();
        double b = Double.parseDouble(bstr);

        writeFile(m,b);
```

```
            return true;
        }catch (NumberFormatException e){
            System.out.println(e);
            return false;
        }
    }

    private void doRetrieve()throws IOException{
        readFile();
        writeDataString("Standby for values of M then B.");
        doSendVariable(m);
        writeDataString("\n");
        doSendVariable(b);
        writeDataString("\n");
    }

    private void readFile(){
        try{
            File infile = new File("c:/Documents and Settings/Andrew/My
Documents/NPS/Thesis/Ipsil/Cal Constants/save.txt");
            FileReader fins = new FileReader(infile);
            System.out.println("In file reader");
            BufferedReader bufRead = new BufferedReader(fins);
            System.out.println("In buff reader");

            String tmp = bufRead.readLine();
            m = Double.parseDouble(tmp);
            tmp= bufRead.readLine();
            b = Double.parseDouble(tmp);
            bufRead.close();
            fins.close();
        }catch (Exception e){
            System.out.println(e);
        }
    }

    private void writeFile(double m, double b){
        try{
            File outfile = new File("c:/Documents and Settings/Andrew/My
Documents/NPS/Thesis/Ipsil/Cal Constants/save.txt");
            FileOutputStream fos = new FileOutputStream(outfile);
            PrintWriter pw = new PrintWriter(fos);
            pw.println(""+m);
            pw.println(""+b);
            pw.close();
            fos.close();
```

```
        }catch (Exception e){
            System.out.println(e);
        }
    }

    private void doGetVariableToUpdate(){
    }

    private void doSendDone(){
        writeDataString("Thanks Goodby!");
    }
}
```

D. CLIENTEXAMPLE.JAVA

```
import java.io.*;
import java.net.*;
/*
 * ClientExample.java
 *
 * Created on October 19, 2005, 1:00 AM
 */

/**
 *
 * @author  carltrask
 */
public class ClientExample {

    Socket skt;
    BufferedReader in;
    PrintWriter out;
    public double m;
    public double b;
    /** Creates a new instance of ClientExample */
    public ClientExample() {
        init();
        getWelcomeMsg();
        getServerQuestion();
        sendServerID(1);
        getServerQuestion();
    }
    public void doUpdate(double m, double b){
            sendResponse("U");
            getServerQuestion();
```

```
        sendResponse(Double.toString(m));
        getServerQuestion();
        sendResponse(Double.toString(b));
        getServerQuestion();
        close();
}
public void doRetrieve(){
        sendResponse("R");
        getServerQuestion();
        m = Double.parseDouble(getData());
        getServerQuestion();
        b = Double.parseDouble(getData());
        getServerQuestion();
        getServerQuestion();
        close();

}
private void getWelcomeMsg(){
   getData();
}

private void getServerQuestion(){
   getData();
}

private void sendServerID(int id){
   sendData(Integer.toString(id));
}
private void sendVariableName(String name){
   sendData(name);
}
private void getVariableVal(){
   getData();
}
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
   try {
       ClientExample ce = new ClientExample();
       ce.doUpdate(12.89, .00023);
       ce = new ClientExample();
       ce.doRetrieve();
       System.out.println("M is " + ce.m +"\n" );
       System.out.println("B is " + ce.b +"\n" );
       System.exit(0);
```

```
    }
    catch(Exception e) {
        System.out.print("<CLIENT> Whoops! It didn't work!\n");
        System.out.println(e);
    }
}
private void init(){
    try {

        skt = new Socket("localhost", 1234);
        in = new BufferedReader(new InputStreamReader(skt.getInputStream()));
        out = new PrintWriter(new OutputStreamWriter(skt.getOutputStream()));
    }catch(Exception e) {
        System.out.print("Whoops! It didn't work!\n");
        System.out.println(e);
    }
}
private void close() {
    try {
        in.close();
        out.close();
        skt.close();
    }
    catch(Exception e) {
        System.out.print("Whoops! It didn't work!\n");
        System.out.println(e);
    }
}

private String getData(){
    try{
        String tmp;
        while (!in.ready()) {
            Thread.sleep(25);
        }

        tmp = in.readLine();
        System.out.println("<CLIENT Received>" +  tmp);
        Thread.sleep(50);
        return tmp;
    }catch (Exception e){
        System.out.println(e);
        return "ERROR\n";
    }
```

```
    }

    private void sendData(String output){
        try {
            Thread.sleep(25);
            System.out.println("<CLIENT Sent>" + output);
            out.println(output);
            out.flush();
            Thread.sleep(25);
        }catch (Exception e){
            System.out.println(e);
        }
    }

    private void sendResponse(String output){
        sendData(output);
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. WEB PAGES

A. FIRST WEB PAGE: SETDIGITALHI.HTM

```
<HTML>
<SCRIPT LANGUAGE="Javascript">
<!--
function U() {return "http://"+location.hostname}
// -->
</SCRIPT>

<HEAD>
<META NAME="Control 8930. Set Digital Outputs HI." Content="Ipsil, Inc.">
<TITLE>
  Control Ipsil device.
</TITLE>
</HEAD>
<BODY>

<SCRIPT>
document.write ("All Digital HI<br>")
document.write("<APPLET   CODE = setDigitalHi.class" + " "
          +" CODEBASE = "+U( )+" "
          +" NAME = setDigitalHi" + " "
          +"ARCHIVE = 207Classes.jar" + " "
          +"WIDTH = 500 HEIGHT =400" + " "
          +"VIEWASTEXT >")
</SCRIPT>

</BODY>
</HTML>
```

B. SECOND WEB PAGE: 208

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Control Ipsil device.</TITLE>
<META http-equiv=Content-Type content="text/html; charset=windows-1252">
<META content="MSHTML 6.00.2900.2668" name=GENERATOR></HEAD>
<BODY><IMG src="165nps.jpg" border=0></IMG><BR>
<SCRIPT language=Javascript>
<!--
function U() {return "http://"+location.hostname}
// -->
</SCRIPT>
```

```
<META content="Ipsil, Inc." name="Insert Calibration Constants."><BR><BR>
<LI><A
href="020config.htm">View Ipsil Configuration</A> <BR><BR>
<LI><A
href="160_ipsil_nozik.htm">View Sensor Configuration</A> <BR><BR>
<SCRIPT>
document.write ("Java Constants Demo<br>")
document.write("<APPLET   CODE = insertCalConts.class" + " "
        +" CODEBASE = "+U( )+" "
        +" NAME = insertCalConts" + " "
        +"ARCHIVE = 209Classes.jar" + " "
        +"WIDTH = 500 HEIGHT =400" + " "
        +"VIEWASTEXT >")
</SCRIPT>
</LI></BODY></HTML>
```

C. THIRD WEB PAGE: 020

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- saved from url=(0029)http://192.168.0.51:01044/020 -->
<HTML><HEAD><TITLE>IPμ8930 Config Info</TITLE>
<META http-equiv=Content-Type content="text/html; charset=windows-1252">
<SCRIPT language=Javascript>
<!--
function mod(divisee,base) {
 return Math.round(divisee - (Math.floor(divisee/base)*base));
}
function TimeOn(tmsw,tlsw){
t = tmsw*65536 + tlsw;
t = t/125;
hrs = Math.floor(t/3600);
t = mod(t,3600);
min = Math.floor(t/60);
t = mod(t,60);
sec = t;
return hrs+" Hrs, "+min+" mins, "+sec+" secs";
}
// -->
</SCRIPT>

<META content="MSHTML 6.00.2800.1523" name=GENERATOR></HEAD>
<BODY><FONT face=Arial><IMG src="IPμ8930 Config
Info_files/011_ipsillogo.gif"
border=0></IMG><BR>
```

```
<TABLE width=707>
 <TBODY>
 <TR>
  <TD>
   <TABLE align=right>
    <TBODY>
    <TR align=right>
     <TD width=60><A
```

```
href="http://192.168.0.51:01044/">HOME</A></TD></TR></TBODY></TABLE></T
D></TR></TBODY></TABLE>
<H1>IPμ8930 Configuration Information</H1>
<H3>Factory Settings and Version Numbers</H3>
<P>Note: Most of these parameters can be changed using the <A
href="http://www.ipsil.com/products/icu.exe">Ipsil Configuration
Utility.</A></P>
<TABLE border=1>
 <TBODY>
 <TR>
  <TD>IPu8930 Name</TD>
  <TD>8930 S/N 11308 </TD>
 <TR>
  <TD>IPu8930 Location</TD>
  <TD>My Location </TD>
 <TR>
  <TD>Current IP Address</TD>
  <TD>192.168.0.51 </TD></TR>
 <TR>
  <TD>DHCP Enable </TD>
  <TD>1</TD>
 <TR>
  <TD>Default IP Address</TD>
  <TD>192.168. 0. 51</TD></TR>
 <TR>
  <TD>Product Type </TD>
  <TD>1</TD>
 <TR>
  <TD>Secure Mode</TD>
  <TD>0</TD>
 <TR>
  <TD>MAC address </TD>
  <TD>00:08:1D:00:41:09</TD></TR>
 <TR>
  <TD>Product/Version </TD>
  <TD>Ipsil IPu8930 v2.151 2003-05-27
</TD></TR></TBODY></TABLE><BR>This IPμ8930
```

has been running continuously for
<SCRIPT>document.write(TimeOn(22,16894))</SCRIPT>

</FONT></BODY></HTML>

D. FOURTH WEB PAGE: 160

```
<HEAD><SCRIPT LANGUAGE="Javascript">
<!--
// -->
</SCRIPT>
<TITLE>Sensor Config Info</TITLE></HEAD>
<BODY>
<FONT FACE="Arial">
<TABLE WIDTH="707"><TR><TD>
<TABLE ALIGN="right"><TR ALIGN="right">
<TD WIDTH="60"><A HREF="/">HOME</A></TD>
<TD WIDTH="60"><A HREF="050help#2.htm">HELP</A></TD>
</TR></TABLE>
</TD></TR></TABLE>
<LI><A
href="020config.htm">View Ipsil Configuration</A> <BR><BR>
<LI><A
href="208_JavaDemo.htm">View Control Configuration</A> <BR><BR>
<H1>Pressure Sensor Information</H1>
<H3>This is an example of sensor information.</H3>

<TABLE BORDER="1">
<TR><TD>Sensor Name</TD><TD>PX4200</TD>
<TR><TD>Sensor Location</TD><TD>Bullard Hall RM-208</TD>
<TR><TD>Equipment Part</TD><TD>test</TD>
<TR><TD>Current IP Address</TD><TD>192.168.0.51</TD></TR>
<TR><TD>Default IP Address</TD><TD>192.168.  0.254</TD></TR>
<TR><TD>Product Type</TD><TD>Silicon on Saphire Pressure Sensor</TD>
<TR><TD>Intall Date</TD><TD>05DEC05</TD>
<TR><TD>Last Calabration Date</TD><TD>07DEC05</TD></TR>
<TR><TD>Product/Version</TD><TD>N/A</TD>
</TABLE>
<BR>
<BR>
<BR>
</FONT>
</BODY>
</HTML>
```

LIST OF REFERENCES

1. Hogan, R. J. T, Cesarone and C. Savage, "*Wireless expansion and enhancements of ICAS*," Proceedings of the Thirteenth International Ship Control Systems Symposium (SCSS), Orlando, FL, April 2003.

2. Noguchi, Yuki, Rockville Firm Hoping to Help the Navy Go Wireless. Washington Post, 7 May 2002.

3. Vern, Clark, "*Sea Power 21, Projecting decisive joint capabilities*," Naval Institute
Proceedings, October 2002.

4. Perchalski, S. J., "*Shipboard sensor closed-loop calibration using wireless LANS and DataSocket transport protocols*," Master's Thesis, Naval Postgraduate School, Monterey, CA, June 2003.

5. D-LINK, Rev.2.0, *DBT-120 USB Bluetooth Adaptor*, D-LINK, Taiwan, August 2002.

6. Rupnow, R., Perchalski, S., Walden, J., Yun, X., Greaves, D., Glick, H., *New Calibration Standings for Next Generation Ship.s Monitoring Systems*, paper presented at the Thirteenth International Ship's Control Systems Symposium (SCSS). Orlando, FL. 7 to 9 April 2003.

7. Ipsil, *Monitor Remote Sensors In Real-Time Via a Browser*, Ipsil Inc., Cambridge, MA, 2003. Page 1.

8. Web Site: *www.ipsil.com*,August, 2005.

9. Ipsil, *IPµ8930 Product Brief*, Ipsil, Inc., Cambridge, MA, 2003.

10. Ipsil, *SensorBlock Product Brief*, Ipsil, Inc., Cambridge, MA, 2003.

11. Web Site: www.*omega.com/Pressure/pdf/PX4200-I.pdf2005*. September, 2005.

12. Web Site: *www.linksys.com* August, 2005.

13. Web Site: *www.dell.com* August, 2005.

14. Ipsil, *Ipsil IPµ8930 Develper Guide*, Ipsil, Inc., Cambridge, MA, 2004.

15. Wu, Thomas, C., *An Introduction to Object-Oriented Programming with Java*^{TM}, McGraw-Hill, New York, NY, 2004.

16. Ipsil, *Using Java Applets With The IPμ8930*, Ipsil, Inc., Cambridge, MA, 2003.

17. Web site: *www.ni.com*, March, 2005.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA

4. Professor Xiaoping Yun, Code EC/Yx
   Department of Electrical and Computer Engineering
   Monterey, CA

5. Professor Robert Hutchins, Code EC/Hu
   Department of Electrical and Computer Engineering
   Monterey, CA

6. Rita Painter
   SPAWAR San Diego
   San Diego, CA

7. Randy Rupnow
   NSWC Corona
   Corona, CA

8. Andrew B. Nozik
   Santa Cruz, CA