Web Enabling Your Device Server

Version 4 09-16-2010

This document and supporting files are available here:

http://ts.lantronix.com/faqs/943/Web.zip

Overview

Serial devices fit typically into just three basic categories. Input only devices, output only devices and input / output devices.

Input only devices include measurement and monitor devices like temperature gauges, weather stations, heart monitors, etc. Output devices include displays like sign boards.
Input and output devices include controls and interactive devices like robotics, PLCs, management equipment, terminal sessions.

A human operator or technician interacted with these devices typically from another 'hard wired' serial device like a computer or controller.

With the technological advances of recent years, it is now possible to interact with these devices from a remote location using a web browser over an Ethernet network. Low cost hardware exists to convert the RS-232, RS-422, or RS-485 serial port into an Ethernet interface, which can be accessed by any IP, based application (like a web browser) over an IP network from any place in the world. Getting application access to the serial port over a network is called serial tunneling. The serial data is simply encapsulated into TCP or UDP packets, which can travel through any IP based network. The hardware performing this function is called a Device Server.

Since Device Servers are inherently network aware, the ability to add functionality like web services, e-mail, network diagnostics are easily within reach. In this paper we'll cover the detailed requirements of adding web services to your serial device.

First we need to get the serial device connected to the network. For this task you'll need a Device Server such as the Lantronix UDS-1100 and the Lantronix Device Installer application to ease the configuration requirements. Follow the documentation to configure the network and serial parameters of the device server. Connect the serial port of the Device Server to your serial equipment using the appropriate cable and connectors, and connect the Device Server to the Ethernet network using an appropriate cable.

Now we need to consider the desired web functionality of the serial device. Of course, we can't change the functionality of the device, but we can enhance the user interface. Regardless of the device functionality, we need a method to query and or control the device over the network. To perform the required programming we'll use a browser-supported language like Java. If you don't already have a Java development kit, one can be downloaded from http://www.oracle.com/technetwork/java/index.html web site. This is required to develop the web-based application. All of the basic functionality and sample code is listed in this paper. (Simply cut and paste as needed.)

Web Servers Background

The CoBox and Evolution family of products supports an internal web server that may be utilized by the web programmer for storage and retrieval of documents, images, and Java applets. This paper will review the concepts and present the methods utilized to program the CoBox and Evolution web servers.

Web browsers, like Microsoft's Internet Explorer or Netscape's Navigator, request information from web servers by using a protocol known as Hypertext Transfer Protocol or HTTP. HTTP was developed in 1990 and allows the transfer of information between two different computers (the browser computer or client, and the HTTP server computer, or server). A client supporting HTTP 1.0, has three methods to interact with the server. These methods (GET, HEAD, and POST) are very basic. The GET method retrieves a particular document or file. The HEAD method retrieves only the "header" of the document. A client will use HEAD to check whether a cached copy of the file has been changed since the last access. The last method POST is typically used with "forms" to send information to the server. When the server receives data from a POST method, it typically passes the data to another application for processing by a mechanism referred to as CGI, or the Common Gateway Interface.

Hypertext Markup Language (HTML) documents are static in nature. CGI, PHP and other programs, are typically used to perform functions on the server, which dynamically build HTML documents. For example, querying a database for the lowest mortgage rate.

The Lantronix CoBox and Evolution Family, which includes the UDS, XPort, MatchPort and EDS, will simply be called a Device Server in this discussion. The Lantronix Device Servers support a HTTP server. This service is used to transfer static documents or files to the requesting web browser. The typical use of the HTTP server is for unit configuration. Simply connecting to the CoBox or Evolution Device Server will bring up the unit's configuration home page. The Device Server may then be configured by this simple "user friendly" interface.

Implementation

The Device Server's HTTP server has support for GET, HEAD and POST methods. This support is adequate for configuration, and file service of help files. It does not support any "server side" required processing. (No user defined CGI support.)

Customers may add their own web pages to the Device Server's HTTP server via a file transfer protocol. Customers that need access to the serial port(s) are encouraged to use a Java applet.

Device Server products that contain 512KB or more of flash memory support the HTTP server. On CoBOS based products, the flash memory area is divided into 64KB pages, which are available for "web sites". These areas are called WEB1 – WEB6 (for 512K flash) and WEB19 on the UDS-1100. In Evolution based products, the flash look like a flat file system with directory support.

When a client makes a GET or HEAD request for a file, the HTTP server process will search the flash for the file. On CoBOS, the service will first look in WEB1 for the file. If the file is not found in WEB1, the process will look in WEB2, then WEB3 and so on. When the file is located, it is sent to the client.

Currently, the CoBOS configuration web pages are loaded into WEB1 and span multiple sections. Since this is the first place the server will look for a file, customers who want to override the Lantronix default page, will need to load their own configuration pages into WEB1 and reload the Lantronix pages following. This way, when a file is requested, the customer's files will be searched first.

The Evolution OS has the configuration pages built into the ROM image. To override these pages, new pages can be loaded into the /http directory.

On CoBOS, the HTTP server process needs to be able to determine if the WEB area contains a valid "web site". The web2cob.exe program is designed to create an archive of HTML documents and other web files. Web2cob.exe also inserts a magic number into the archive. The server process will test the magic number to determine if a valid "web site" is loaded into that section of flash memory. Since the flash memory is divided into pages, each "web site" or .COB file cannot be larger than 64KB. It is the responsibility of the programmer not to create files larger than 64KB. (In the later versions of the OS, .COB files can be larger than 64KB. However, no single file within the archive can be larger than 64KB.)

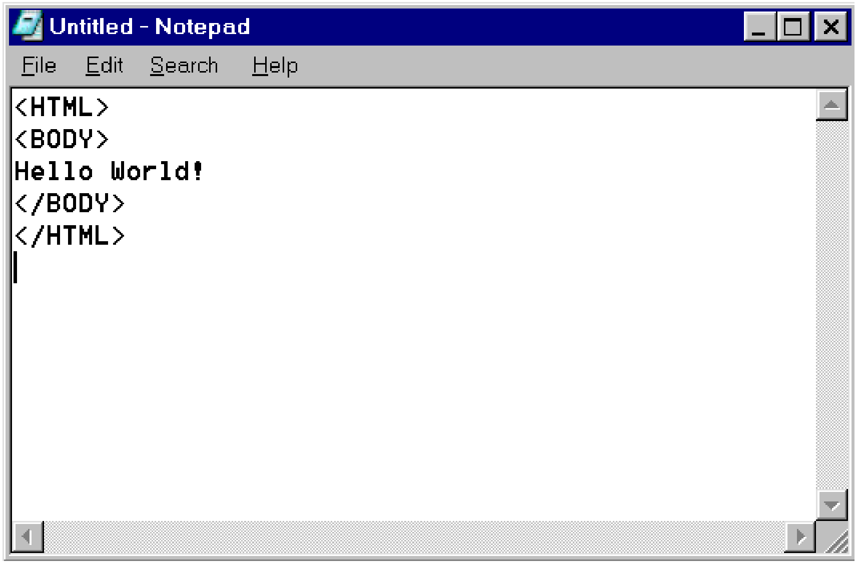
On Evolution, the files should not be converted into a cob archive and there are no restrictions on the file size.

Creating Web Pages

Several books and programs exist to help in the creation of HTML documents and Java applets. The details of this subject are beyond the scope of this document. However, an example may help clarify the subject.

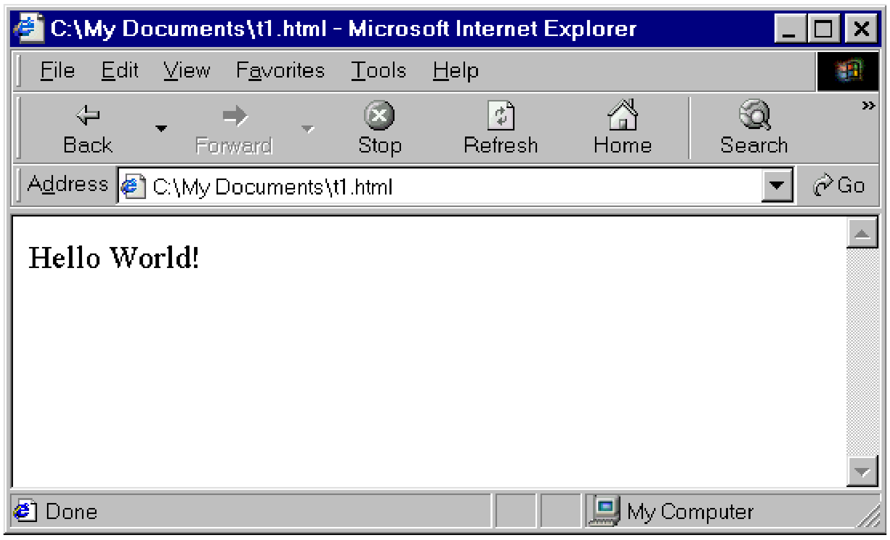
To create a simple HTML document, you can use any text editor. Insert the following text into the editor, and store the file as t1.html.

```
<HTML>
<BODY>
Hello World!
</BODY>
</HTML>
```



```
<HTML>
<BODY>
Hello World!
</BODY>
</HTML>
```

To test your new creation, open this file using your browser.


C:\My Documents\t1.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Address
Go
Done
My Computer

C:\My Documents\t1.html

Hello World!

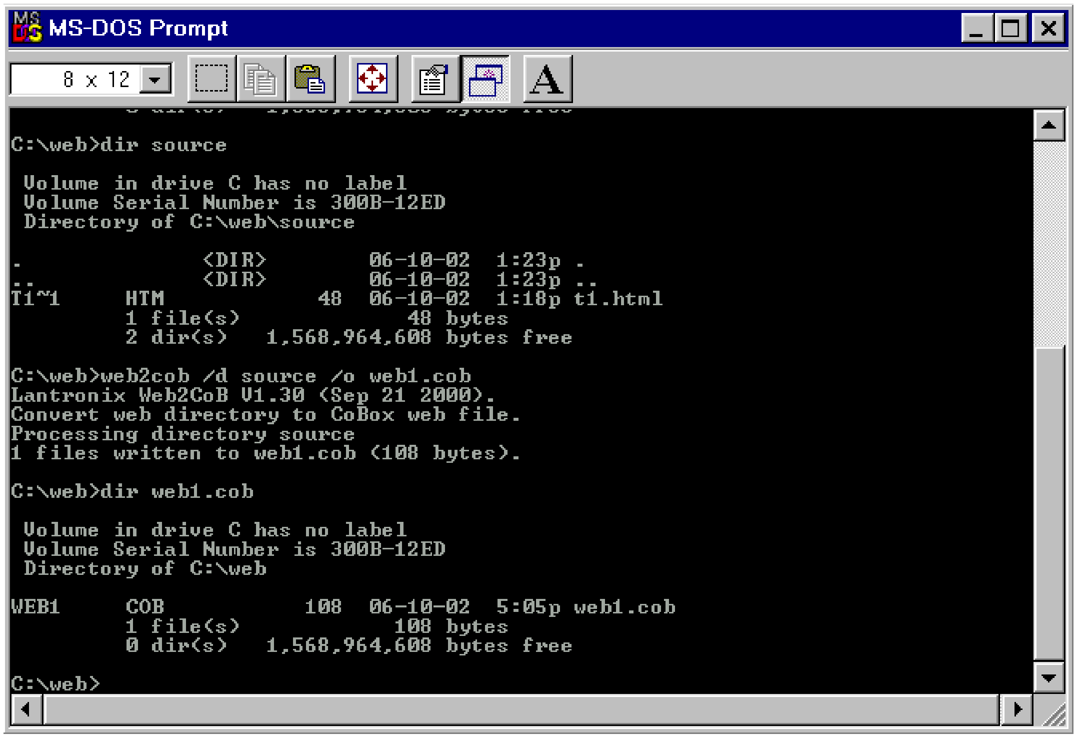
Creating .COB Files (CoBOS based devices only)

In order to store your files on the Device Server's flash card, they must be in the correct format. Lantronix provides a utility called "web2cob.exe" to create the required the archive file for the "web site".

Syntax: Web2cob.exe /d <directory> /o <output_file_name>
Where: "directory" is the directory that contains the files to be archived
"output_file_name" is the name of the archive to be created
Example:
Web2cob /d myfiles /o web3.cob

To create your own .COB file, follow this process:

1. From the Command or DOS prompt, create a new directory.
    a. C:> mkdir \web
2. Copy the two supplied Lantronix files into the new directory
    a. C:> copy web2cob.exe \web
    b. C:> copy mimetype.ini \web
3. Create a new directory under \web.
    a. C:> mkdir \web\source
4. Copy your HTML documents into \web\source.
    a. C:> copy t1.html \web\source
5. Create a .COB file.
    a. C:> cd \web
    b. C:> web2cob /d source /o web1.cob
6. Verify that the new file is smaller than 64KB.
    a. C:> dir web1.cob



```
C:\web>dir source

 Volume in drive C has no label
 Volume Serial Number is 300B-12ED
 Directory of C:\web\source

.              <DIR>        06-10-02  1:23p .
..             <DIR>        06-10-02  1:23p ..
T1~1      HTM          48   06-10-02  1:18p t1.html
          1 file(s)            48 bytes
          2 dir(s)     1,568,964,608 bytes free

C:\web>web2cob /d source /o web1.cob
Lantronix Web2CoB V1.30 (Sep 21 2000).
Convert web directory to CoBox web file.
Processing directory source
1 files written to web1.cob (108 bytes).

C:\web>dir web1.cob

 Volume in drive C has no label
 Volume Serial Number is 300B-12ED
 Directory of C:\web

WEB1      COB         108   06-10-02  5:05p web1.cob
          1 file(s)           108 bytes
          0 dir(s)     1,568,964,608 bytes free

C:\web>
```

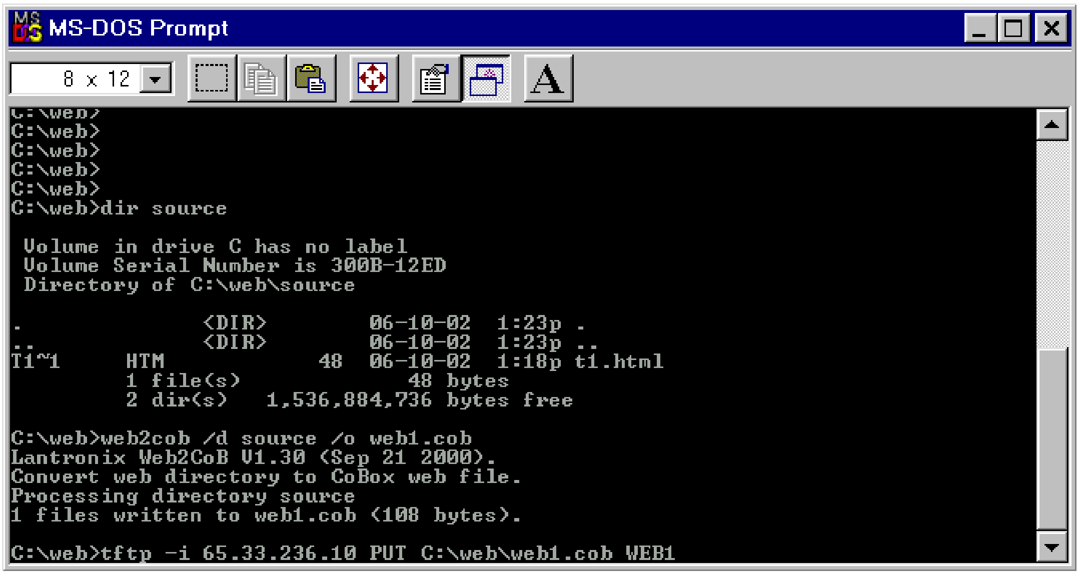
Downloading .COB Files to the CoBOS Device Server

To load a .COB file into the Device Server, you will need a tftp client program. Several tftp programs are available that run on Microsoft Windows platforms. For our example we will use the tftp client built into Windows 2000, and Windows NT.

To load the .COB file into the Device Server, follow this process:

1. From a command or DOS prompt on Windows 2000 or NT
    a. C:> tftp -i xxx.xxx.xxx.xxx PUT \web\web1.cob WEB1

Note: Substitute your Device Server's IP address for the xxx.xxx.xxx.xxx above.



```
C:\web>
C:\web>
C:\web>
C:\web>
C:\web>
C:\web>dir source

 Volume in drive C has no label
 Volume Serial Number is 300B-12ED
 Directory of C:\web\source

.              <DIR>        06-10-02  1:23p .
..             <DIR>        06-10-02  1:23p ..
TI~1      HTM          48  06-10-02  1:18p ti.html
         1 file(s)             48 bytes
         2 dir(s)   1,536,884,736 bytes free

C:\web>web2cob /d source /o web1.cob
Lantronix Web2CoB V1.30 (Sep 21 2000).
Convert web directory to CoBox web file.
Processing directory source
1 files written to web1.cob (108 bytes).

C:\web>tftp -i 65.33.236.10 PUT C:\web\web1.cob WEB1
```

Downloading Files to the Evolution Device Server

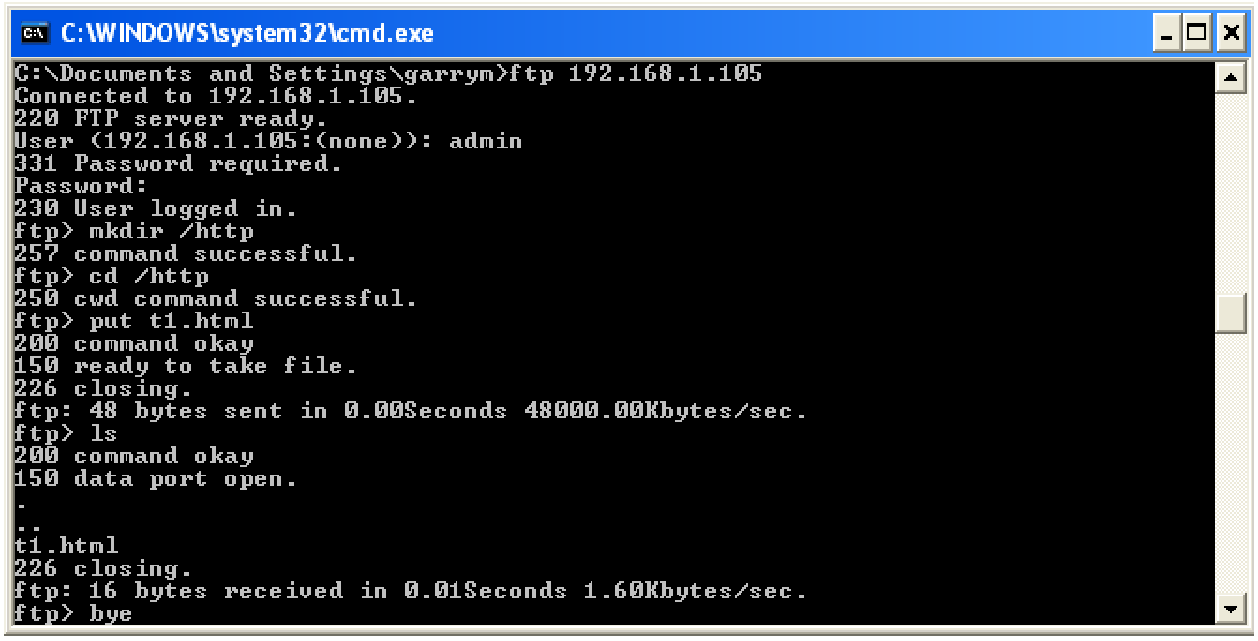
To load a file into the Device Server, you will need a ftp client program. Several ftp programs are available that run on Microsoft Windows platforms. For our example we will use the ftp client built into Windows 2000, and Windows NT.

To load the t1.html file into the Device Server, follow this process:

1. From a command or DOS prompt on Windows 2000 or NT
    a. C:> ftp xxx.xxx.xxx.xxx

Note: Substitute your Device Server's IP address for the xxx.xxx.xxx.xxx above.

    b. User: admin
    c. Password: PASS
    d. mkdir /http
    e. cd /http
    f. put t1.html
    g. bye

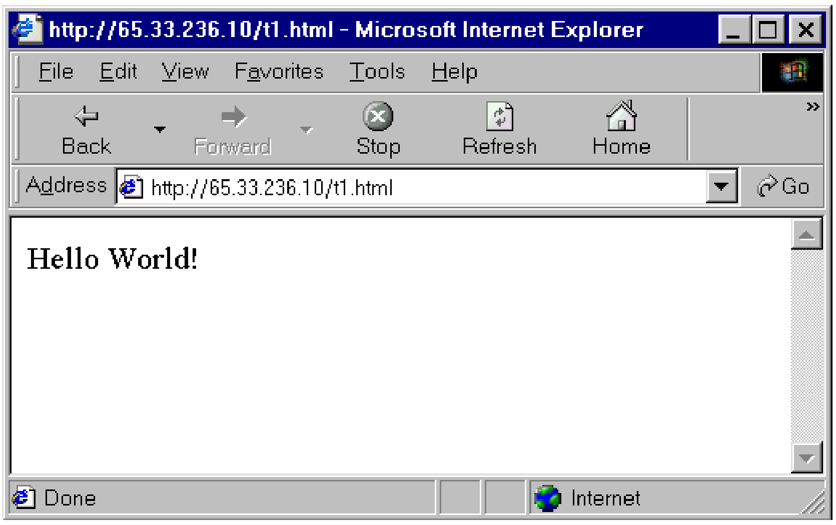


```
C:\Documents and Settings\garrym>ftp 192.168.1.105
Connected to 192.168.1.105.
220 FTP server ready.
User (192.168.1.105:(none)): admin
331 Password required.
Password:
230 User logged in.
ftp> mkdir /http
257 command successful.
ftp> cd /http
250 cwd command successful.
ftp> put t1.html
200 command okay
150 ready to take file.
226 closing.
ftp: 48 bytes sent in 0.00Seconds 48000.00Kbytes/sec.
ftp> ls
200 command okay
150 data port open.
.
..
t1.html
226 closing.
ftp: 16 bytes received in 0.01Seconds 1.60Kbytes/sec.
ftp> bye
```

You can now access your new web page from any browser using the new URL:

http://xxx.xxx.xxx.xxx/t1.html

Where xxx.xxx.xxx.xxx is the IP address of the Device Server.


http://65.33.236.10/t1.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Address
Go
Done
Internet

http://65.33.236.10/t1.html

Hello World!

Connecting to the Device Server with Java

Writing a Java applet to communicate with a serial device attached to the Device Server is very straightforward. However, you will need to be familiar with Java programming, and need a Java compiler.

Like any network application, you must open a communication channel to the remote device. In our example, we open a socket, and create two data streams to perform the actual sending and receiving of data. The following example will use a new Java Class called “tcpip”. Simply copy the following code into a file called tcpip.java.

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

/*
 * This class opens a TCP connection, and allows reading and writing of byte arrays.
 */

public class tcpip
{
        protected Socket s = null;
        public DataInputStream dis = null;
        protected DataOutputStream dos = null;

        public tcpip(InetAddress ipa, int port)
        {
                Socket s1 = null;
          try {                                 // Open the socket
                        s1 = new Socket(ipa.getHostAddress(), port);
                }
          catch (IOException e) {
                        System.out.println("Error opening socket");
                        return;
                }
                s = s1;
                try {                           // Create an input stream
                        dis = new DataInputStream(new BufferedInputStream(s.getInputStream()));
                }
                catch(Exception ex) {
                        System.out.println("Error creating input stream");
                }
                try     {                       // Create an output stream
                        dos = new DataOutputStream(new BufferedOutputStream(s.getOutputStream()));
                }
                catch(Exception ex) {
                   System.out.println("Error creating output stream");
                }
        }

        public synchronized void disconnect()
        {
                if (s != null) {
                        try {
                                s.close();
                        }
                        catch (IOException e){}
                }
        }

        public synchronized void send(byte[] temp)
        {
                try {
                        dos.write(temp, 0, temp.length);
                        dos.flush();
                }
                catch(Exception ex) {
                        System.out.println("Error sending data : " + ex.toString());
                }
        }

        public synchronized void send(byte[] temp, int len)
        {
                try {
                        dos.write(temp, 0, len);
                        dos.flush();
                }
                catch(Exception ex) {
                        System.out.println("Error sending data : " + ex.toString());
                }
        }

        public synchronized void send(String given)
        {
                                // WARNING: this routine may not properly convert Strings to bytes
                int length = given.length();
                byte[] retvalue = new byte[length];
                char[] c = new char[length];
                given.getChars(0, length, c, 0);
                for (int i = 0; i < length; i++) {
                        retvalue[i] = (byte)c[i];
                }
                send(retvalue);
        }

        public synchronized byte[] receive()
        {
                byte[] retval = new byte[0];

                try {
                        while(dis.available() == 0);  /* Wait for data */
                }
                catch (IOException e){}
                try {
                        retval = new byte[dis.available()];
                }
                catch (IOException e){}
                try {
                        dis.read(retval);
                }
                catch (IOException e){}
                return(retval);
        }

        public int available()
        {
                int avail;

                avail = 0;
                try {
                        avail = dis.available();
                }
                catch (IOException e) {}

                return(avail);
        }
}
```

Next, create the Text_io class as the application. Copy the following code into a file called "Text_io.java".

```
import java.awt.*;
import java.awt.event.*;
import java.lang.*;

public class Text_io extends Panel implements Runnable, TextListener {
        private tcpip gtp;
        String oldmessage = new String("");
        TextArea input_box  = new TextArea("", 10, 60, 3);
TextArea output_box = new TextArea("", 10, 60, 3);
Thread timer;

public Text_io(tcpip tp) {
        gtp = tp;

        setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.insets =  new Insets(5,5,5,5);
        setBackground(java.awt.Color.lightGray);
        setSize(561,380);

        c.gridx = 0; c.gridy = 2; c.gridwidth = 1; c.gridheight = 1;
        c.weightx = 0.0; c.weighty = 0.0; c.anchor = GridBagConstraints.WEST;
        c.fill = GridBagConstraints.NONE;
        add((new Label("To Device Server:  (Type Here--->)")), c);

        //input_box
        input_box.addTextListener(this);
        c.gridx = 1; c.gridy = 2; c.gridwidth = 3; c.gridheight = 1;
        c.weightx = 0.5; c.weighty = 0.0; c.anchor = GridBagConstraints.CENTER;
        c.fill = GridBagConstraints.BOTH;
        add(input_box,c);

        c.gridx = 0; c.gridy = 4; c.gridwidth = 1; c.gridheight = 1;
        c.weightx = 0.0; c.weighty = 0.0; c.anchor = GridBagConstraints.WEST;
        c.fill = GridBagConstraints.NONE;
        add((new Label("From Device Server:")), c);

        c.gridx = 1; c.gridy = 4; c.gridwidth = 3; c.gridheight = 1;
        c.weightx = 0.5; c.weighty = 0.0; c.anchor = GridBagConstraints.CENTER;
        c.fill = GridBagConstraints.BOTH;
        add(output_box,c);
        output_box.setEditable(false);
        timer = new Thread(this);
        timer.start();
}

public void run() {
        int i;
        byte[] in;
        Thread me = Thread.currentThread();
        while (timer == me) {
                try {
                        Thread.currentThread().sleep(200);
                }
                catch (InterruptedException e) { }
                if ( (gtp != null) && ((i = gtp.available()) > 0) ) {
                        in = gtp.receive();
                                        /* remove non-printing bytes */
                        for (i = 0; i < in.length; i++) {
                                if (in[i] < 0x20)
                                        in[i] = 0x20;
                        }
                        output_box.append((new String(in)));
                }
        }
}

public void textValueChanged(TextEvent e) {
        int len, i;
        String str = new String("");
        String message = input_box.getText();
        len = message.length() - oldmessage.length();
        if (len < 0) {
                for (i = 0; i < -len; i++)
                        str += "\b";
                //System.out.println("Backspace");
        }
        else if (len > 0) {
                str = message.substring(oldmessage.length());
                //System.out.println("len = "+str.length()+" str = "+str);
        }
        oldmessage = message;
        if ( (len != 0) && (gtp != null) )
```

```
                                        gtp.send(str);
                }
}
```

Next, create the actual applet that uses the tcpip and Text_io classes. Copy the following code into a file called “Test.java”.

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


public class Test extends Applet {
        static private boolean isapplet = true;
        static private InetAddress arg_ip = null;
        static private int arg_port = 0;
        public tcpip gtp = null;;
        InetAddress reader_ip = null;
        int port = 10001;

        public void init()
        {
                gtp = null;
                reader_ip = null;
                port = 10001;
        }

        public void start()
        {
                String st = new String("TCP/IP connection status: ");
                setFont(new Font("Dialog",Font.BOLD,16));
                setLayout(new GridBagLayout());
                GridBagConstraints c = new GridBagConstraints();
                c.gridx = 0; c.gridy = 0; c.gridwidth = 1; c.gridheight = 1;
                c.anchor = GridBagConstraints.CENTER;
                c.fill = GridBagConstraints.BOTH;
                c.insets =  new Insets(5,5,5,5);
                setBackground(Color.yellow);
                setSize(600,500);

                                /* Either get the IP address from the HTTP server if we're
                                   an applet, or from the commandline (if passed). */
                if (isapplet) {
                        try{
                                reader_ip = InetAddress.getByName(getCodeBase().getHost());
                        }
                        catch (UnknownHostException e){}
                }
                else {
                        reader_ip = arg_ip;
                        if (arg_port != 0) {
                                port = arg_port;
                        }
                }
                                /* Open a socket to the Device Server's serial port */
                if (reader_ip != null) {
                        if (gtp == null) {
                                gtp = new tcpip(reader_ip, port);
                                if (gtp.s == null) {
                                        st += "Connection FAILED! ";
                                        gtp = null;
                                }
                        }
                }
                if (gtp == null) {
                        st += "Not Connected";
                        add((new Label(st)), c);
                        return;
```

```
			}
			st += "Connected";
			add((new Label(st)), c);
					/* You may now perform IO with the Device Server via
					 *			gtp.send(byte[] data_out);
					 *			byte[] data_in = gtp.receive();
					 * functions.
					 * In our example we'll use two TextBoxes which have been extended
					 * to handle IO to the Device Server.  Data typed in the upper
					 * text box will be sent to the Device Server, and data received
					 * will be displayed in the lower text box.
					 */
		/*******************************************************/
		/*	Start of custom application code	*/
		/*	ADD YOUR CODE HERE			*/
		/*******************************************************/
			c.gridx = 0; c.gridy = 2; c.gridwidth = 3; c.gridheight = 1;
			c.anchor = GridBagConstraints.WEST;
			add((new Text_io(gtp)), c);
		/*******************************************************/
		/*	End of custom application code	*/
		/*******************************************************/

		}

		public void destroy()
		{
			if (gtp != null)
				gtp.disconnect();
			gtp = null;
		}

		public void stop() {
		}

		public static void main(String[] args) {
			Frame frame = new Frame("TCP/IP Test");
			frame.addWindowListener(new WindowAdapter() {
				public void windowClosing(WindowEvent e) {
					System.exit(0);
				}
			});
			if (args.length > 0) {
				try{
					arg_ip = InetAddress.getByName(args[0]);
				}
				catch (UnknownHostException e){}
				if (args.length > 1) {
					try {
						arg_port = Integer.valueOf(args[1]).intValue();
					}
					catch (NumberFormatException e) {}
				}
			}
			Test ap = new Test();
			frame.add(ap);
			ap.init();
			isapplet = false;
			ap.start();
			frame.pack();
			// frame.show(); /* deprecated */
			frame.setVisible(true);
		}
}
```

In our example we use two TextBoxes, which are in the Text_io class to handle IO to and from the Device Server. Data typed in the upper text box will be sent to the Device Server, and data received will be displayed in the lower text box.

If you want to use your own code, you'll need to add your application code at the tag /* ADD YOUR CODE HERE */ which is in the start() routine. You'll also need to replace the section of code between the ‘custom code’ fences. The syntax for the send and receive functions are listed below. Use gtp.send() to send data to the serial device, and gtp.receive() to read data from the serial device.

Syntax:

```
gtp.send(byte[] array)
byte[] array = gtp.receive()
```

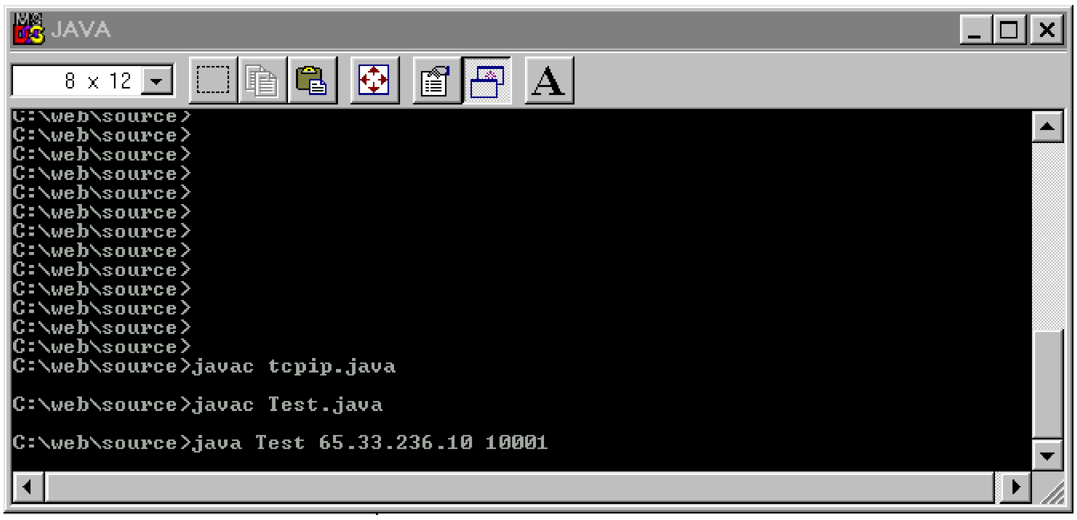
Now you need to compile the Java source code into class files.

```
C:> javac tcpip.java
C:> javac Text_io.java
C:> javac Test.java
```

To test your Java application:

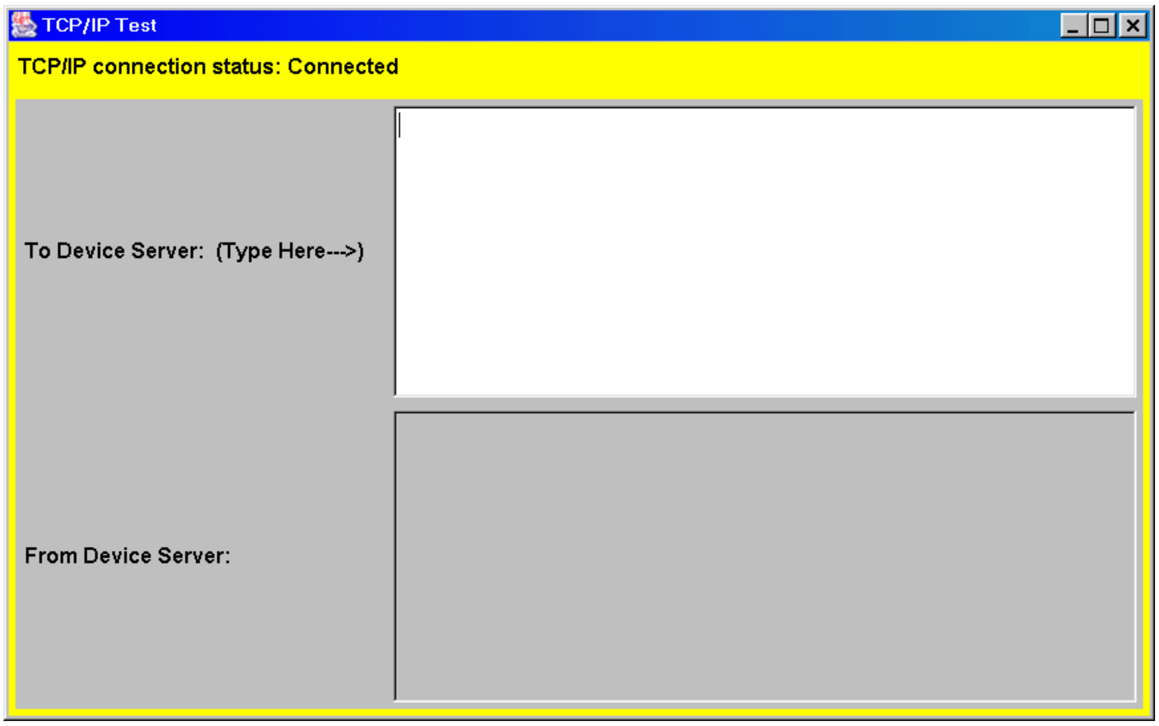
```
C:> java Test xxx.xxx.xxx.xxx 10001
```

Where: xxx.xxx.xxx.xxx is the IP address of the CoBox or UDS, and 10001 is the TCP port number the CoBox or UDS is listening on. (10001 is the default port number.)



```
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>
C:\web\source>javac tcpip.java

C:\web\source>javac Test.java

C:\web\source>java Test 65.33.236.10 10001
```

A large window should appear that says “TCP/IP connection status: Connected”.



TCP/IP connection status: Connected

To Device Server: (Type Here--->)

From Device Server:

If the response is “FAILED”, then the actual TCP connection failed. This could be lack of communication with the device, wrong IP address or wrong port number. If the response is “Not Connected”, you did not pass a valid IP address on the command line.

Once you’ve successfully connected to the Device Server, and completely debugged your Java application, you can then create a HTML document, which includes your Java application in the form of an Applet. Copy the text below into a file called “test.html”.

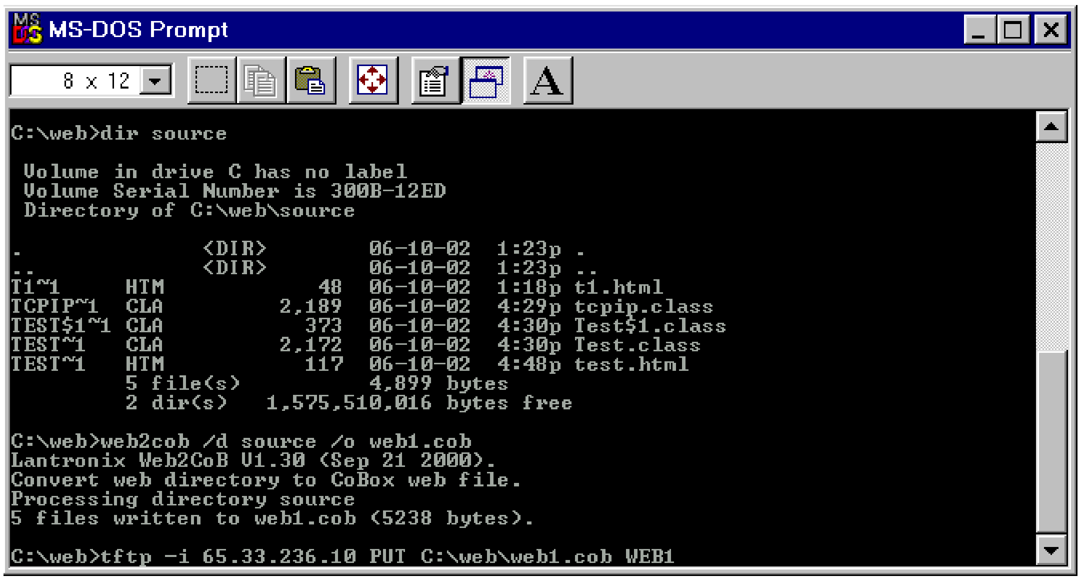
```
<HTML>
<BODY>
<CENTER>
<APPLET CODE="Test.class" WIDTH="862" HEIGHT="512"></APPLET>
</CENTER>
</BODY>
</HTML>
```

Now copy the Java classes and new HTML file into the \web\source directory.

```
C:> copy *.class \web\source
C:> copy test.html \web\source
```

For CoBOS based devices, create a new .COB file, as described above, and load it into the Device Server via tftp.

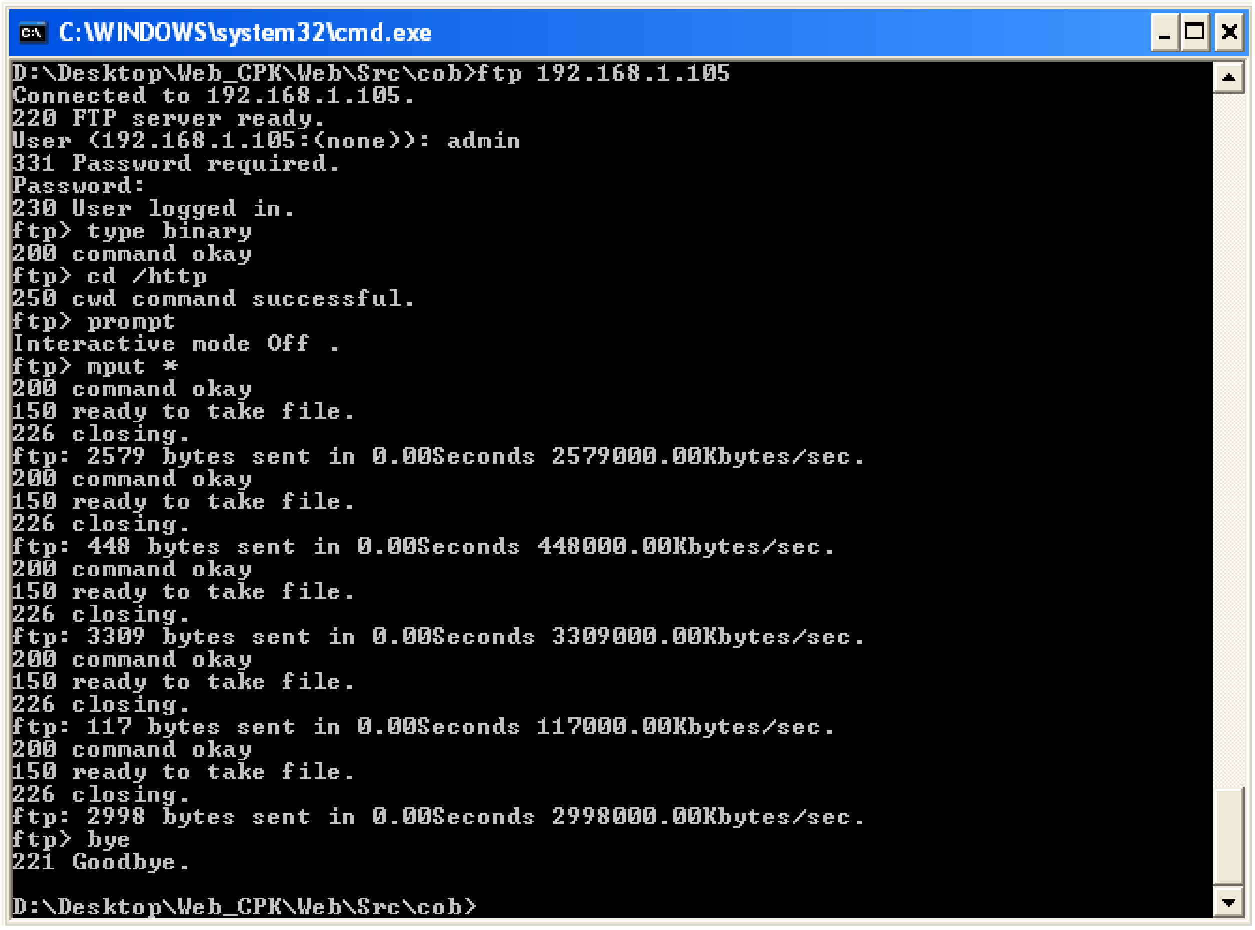
```
C:> cd \web
C:> web2cob /d source /o web1.cob
C:> tftp -i xxx.xxx.xxx.xxx PUT \web\web1.cob WEB1
```



```
C:\web>dir source

 Volume in drive C has no label
 Volume Serial Number is 300B-12ED
 Directory of C:\web\source

.              <DIR>        06-10-02  1:23p .
..             <DIR>        06-10-02  1:23p ..
T1~1     HTM            48  06-10-02  1:18p t1.html
TCPIP~1  CLA         2,189  06-10-02  4:29p tcpip.class
TEST$1~1 CLA           373  06-10-02  4:30p Test$1.class
TEST~1   CLA         2,172  06-10-02  4:30p Test.class
TEST~1   HTM           117  06-10-02  4:48p test.html
         5 file(s)          4,899 bytes
         2 dir(s)   1,575,510,016 bytes free

C:\web>web2cob /d source /o web1.cob
Lantronix Web2CoB V1.30 (Sep 21 2000).
Convert web directory to CoBox web file.
Processing directory source
5 files written to web1.cob (5238 bytes).

C:\web>tftp -i 65.33.236.10 PUT C:\web\web1.cob WEB1
```

For Evolution based devices, transfer the files to the Device Server via ftp.

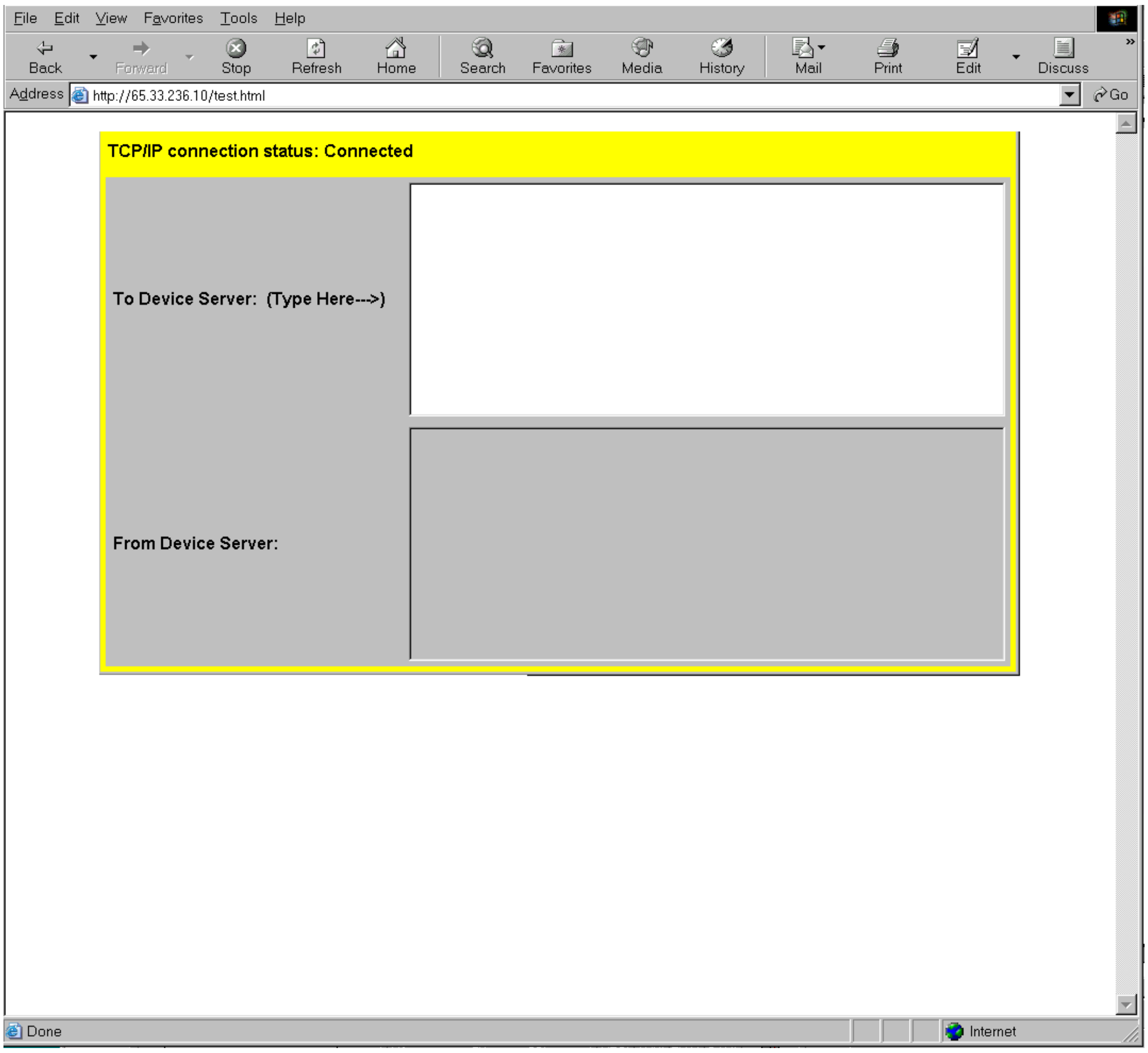
```
C:> cd \web
C:> ftp xxx.xxx.xxx.xxx
        User: admin
        Password: PASS
        ftp> type binary
        ftp> cd /http
        fpt> prompt
        ftp> mput *
        ftp> bye
```



C:\WINDOWS\system32\cmd.exe

```
D:\Desktop\Web_CPK\Web\Src\cob>ftp 192.168.1.105
Connected to 192.168.1.105.
220 FTP server ready.
User (192.168.1.105:(none)): admin
331 Password required.
Password:
230 User logged in.
ftp> type binary
200 command okay
ftp> cd /http
250 cwd command successful.
ftp> prompt
Interactive mode Off .
ftp> mput *
200 command okay
150 ready to take file.
226 closing.
ftp: 2579 bytes sent in 0.00Seconds 2579000.00Kbytes/sec.
200 command okay
150 ready to take file.
226 closing.
ftp: 448 bytes sent in 0.00Seconds 448000.00Kbytes/sec.
200 command okay
150 ready to take file.
226 closing.
ftp: 3309 bytes sent in 0.00Seconds 3309000.00Kbytes/sec.
200 command okay
150 ready to take file.
226 closing.
ftp: 117 bytes sent in 0.00Seconds 117000.00Kbytes/sec.
200 command okay
150 ready to take file.
226 closing.
ftp: 2998 bytes sent in 0.00Seconds 2998000.00Kbytes/sec.
ftp> bye
221 Goodbye.

D:\Desktop\Web_CPK\Web\Src\cob>
```

You can now access your new web page from any browser using the new URL:
http://xxx.xxx.xxx.xxx/test.html
Where xxx.xxx.xxx.xxx is the IP address of the Device Server.


File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss
Address http://65.33.236.10/test.html
Go
Done
Internet

TCP/IP connection status: Connected

To Device Server: (Type Here--->)

From Device Server:

Summary

The CoBox and Evolution family supports an internal web server that may be utilized for storage and retrieval of documents, images, and Java applets. With a little effort, the experienced web programmer can create custom web pages to interact with any serial device and present the information to a user in a very familiar and friendly user interface.