Web Enabling Your Device Server

Part Number 900-326
Revision B July 2004

Copyright and Trademark

Contacts

Lantronix Corporate Headquarters
15353 Barranca Parkway
Irvine, CA 92618, USA
Phone: 949-453-3990
Fax: 949-453-3995

Technical Support
Phone: 800-422-7044 or 949-453-7198
Fax: 949-450-7226
Online: www.lantronix.com/support
Email mailto:support@lantronix.com

Sales Offices
For a current list of our domestic and international sales offices, go to the Lantronix web site at http://www.lantronix.com/about/contact/index.html

Contents

1: 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 such as temperature gauges, weather stations, heart monitors, etc. Output devices include displays (e.g. sign boards).

Input and output devices include controls and interactive devices such as robotics, PLCs, management equipment, terminal sessions.

Typically, an operator or technician interacts with these devices from another hard-wired serial device (a computer or a controller).

With the technological advances of recent years, it is 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 is accessible by an IP-based application (i.e. a web browser) over an IP network from any place in the world. Obtaining access via an application to the serial port over a network is called *serial tunneling*. The serial data is encapsulated into TCP or UDP packets; these 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 (web services, e-mail, network diagnostics) are easily within reach. This paper covers the detailed requirements of adding web services to your serial device.

First, connect the serial device to the network. This requires a Device Server such as the Lantronix UDS-100 and the Lantronix Device Installer to ease configuration. 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 a suitable cable.

Next, consider the desired web functionality of the serial device. Device functionality cannot be changed but you can enhance the user interface. Regardless of the device functionality, a method is needed to query or control the device over the network. To perform the required programming, use a browser-supported language like Java. To obtain a Java development kit, download one from the Sun Microsystems web site. This is required to develop the web-based application.

All of the basic functionality and sample code is listed in this paper. (Copy and paste as needed.)

2: Background on Web Servers

The CoBox family of products supports an internal web server used by the web programmer for storage and retrieval of documents, images, and Java applets. This article reviews the concepts and present the methods utilized to program the CoBox web server.

Web browsers, such as Microsoft Internet Explorer or Netscape Navigator, request information from web servers by using a protocol known as Hypertext Transfer Protocol (HTTP). HTTP allows the transfer of information between two different computers: the browser computer (client) and the HTTP computer 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. 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 the Common Gateway Interface (CGI).

Hypertext Markup Language (HTML) documents are static in nature. Programs such as CGI and PHP typically perform functions on the server to dynamically build HTML documents (e.g. querying a database for the lowest mortgage rate).

The Lantronix CoBox Family (includes the UDS, XPort, and WiPort) will be referred to as the Device Server in this paper. The Lantronix Device Servers support an 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. Connecting to the CoBox Device Server brings up the unit’s configuration home page. Then, the Device Server may be configured by this user-friendly interface.

3: Implementation

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

Customers may add their own web pages to the Device Server HTTP server using a tool called *web2cob* available from Lantronix (discussed later). Customers requiring access to the serial port(s) are encouraged to use a Java applet.

Device Server products containing 512KB or more of flash memory support the HTTP server. 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 WEB14 on the UDS-100.

When a client makes a GET or HEAD request for a file, the HTTP server process first looks 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.

The Lantronix configuration web pages are loaded into the highest numbered WEB section. Since this is the last place the server will look for a file, users may add their own configuration pages in any lower numbered web area. This way, when a file is requested, users' files are searched first.

The HTTP server process must 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 unique number into the archive. The server process will test the unique number to determine if a valid web site is loaded into that section of flash memory.

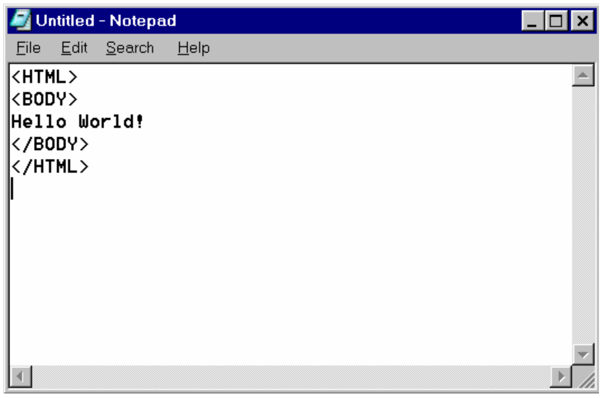
Note: *Since the flash memory is divided into pages, each web site or .COB file cannot be larger than 64KB. It is the responsibility of the programmer to not create files larger than 64KB.*

4: 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.

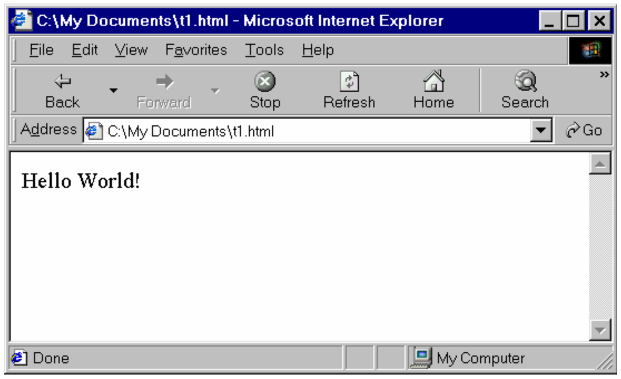
To create a simple HTML document, use any text editor. Insert the following text into the editor, and store the file as **t1.html**.

```
<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
C:\My Documents\t1.html
Go
Done
My Computer

Hello World!

5: Creating .COB Files

In order to store files on the Device Server's flash card, they have 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 and "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:

1. From the Command or DOS prompt, create a new directory.

   ```
   C:> mkdir \web
   ```

2. Copy the two supplied Lantronix files into the new directory

   ```
   C:> copy web2cob.exe \web
   C:> copy mimetype.ini \web
   ```

3. Create a new directory under \web.

   ```
   C:> mkdir \web\source
   ```

4. Copy your HTML documents into \web\source.

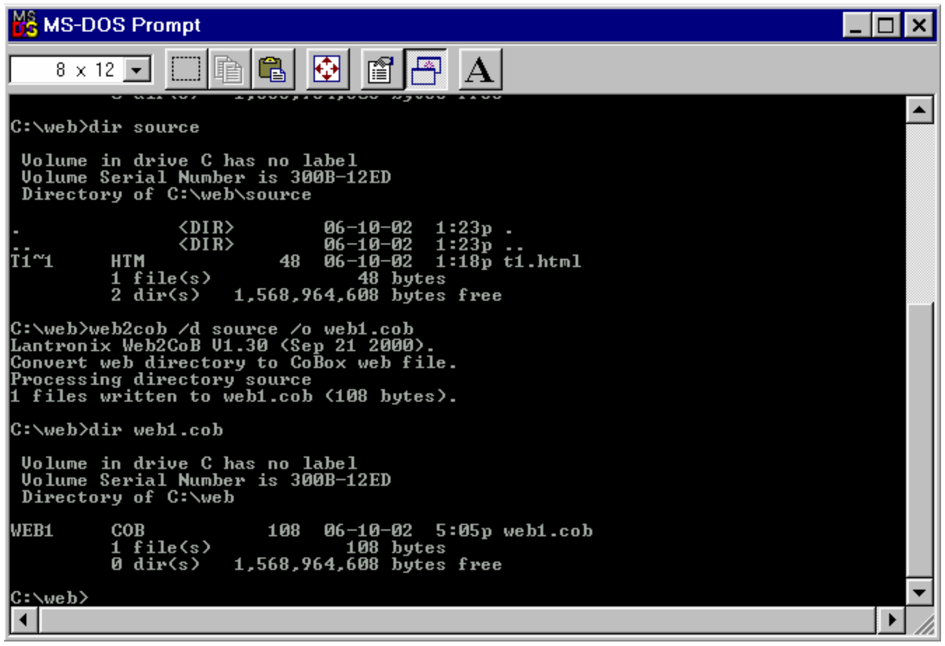
   ```
   C:> copy t1.html \web\source
   ```

5. Create a .COB file.

   ```
   C:> cd \web
   C:> web2cob /d source /o web1.cob
   ```

6. Verify that the new file is smaller than 64KB.

   ```
   C:> dir web1.cob
   ```



MS-DOS Prompt

```
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>
```

6: Downloading .COB Files to the Device Server

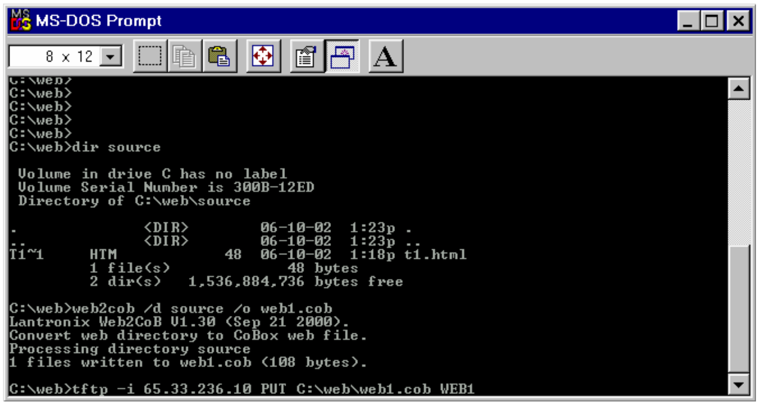
To load a .COB file into the Device Server, a TFTP client program is required. Several TFTP programs are available that run on Microsoft Windows platforms. For our example we will use the TFTP client built into Windows 2000, and Windows NT.

To load the .COB file into the Device Server:

1. From a command or DOS prompt on Windows 2000 or NT, enter:

   ```
   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.*

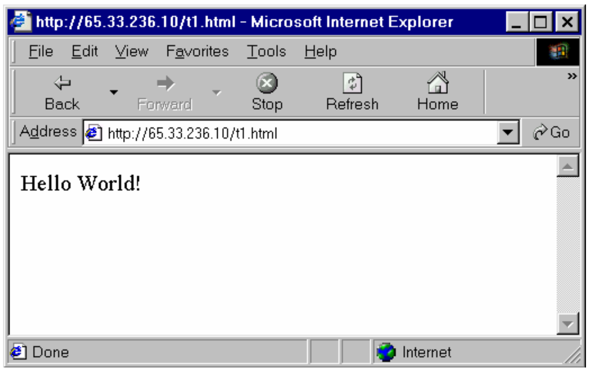


```
C:\web>
C:\web>
C:\web>
C:\web>
C:\web>
C:\web>dir source

 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 ..
T1~1      HTM           48  06-10-02  1:18p t1.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
```

2. Access your new web page from any browser using the new URL:

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

   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
http://65.33.236.10/t1.html
Go
Done
Internet

Hello World!

7: 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, familiarity with Java programming and a Java compiler are required.

As with any network application, open a communication channel to the remote device. In our example, a socket is opened and and two data streams are created to perform the actual sending and receiving of data.

The following example uses a new Java Class called *tcpip*. 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();
      }
   }
```

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()
```

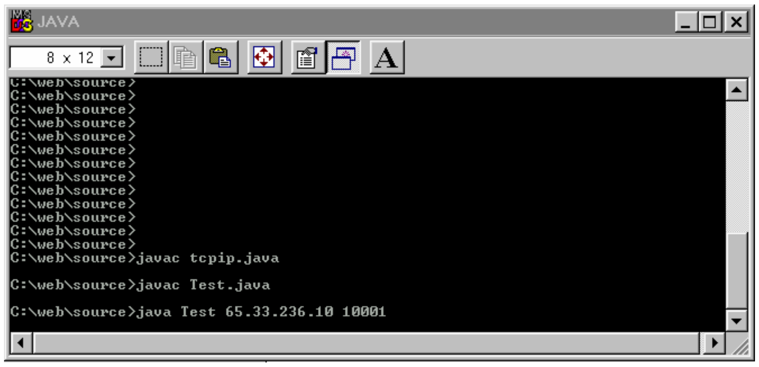
Now you need to compile the Java source code into class files.

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

To test your Java application:

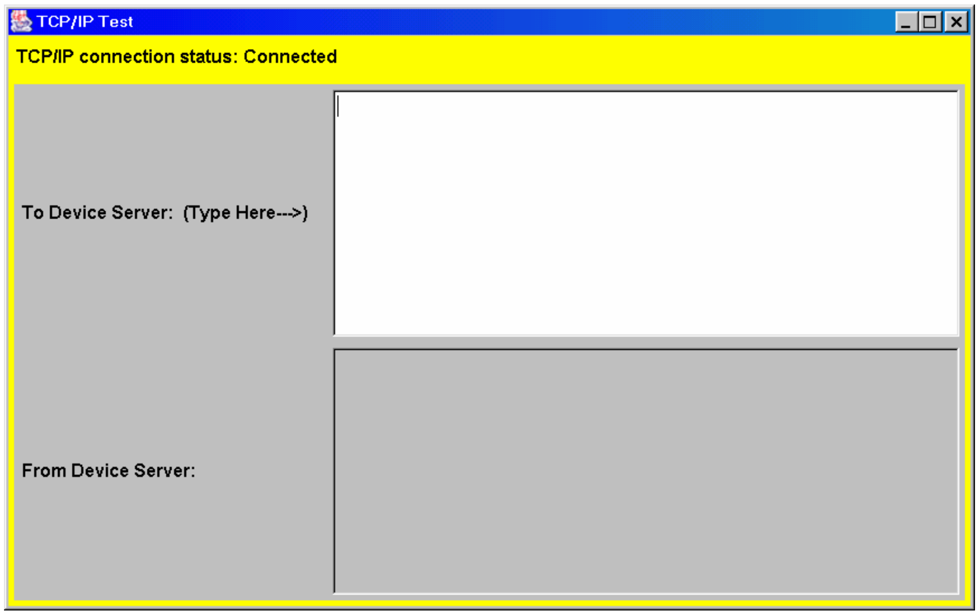
```
C:> java Test xxx.xxx.xxx.xxx 10001
```

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>
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 appears with the message: TCP/IP connection status: Connected.



TCP/IP connection status: Connected

If the response reads FAILED, then the actual TCP connection failed. This could be a lack of communication with the device, a wrong IP address, or a wrong port number. If the response is Not Connected, an invalid IP address was entered on the command line.

After connecting to the Device Server and completely debugging the Java application, create an HTML document (which includes the Java application in the form of an Applet).

1. Copy the text below into a file called test.html.

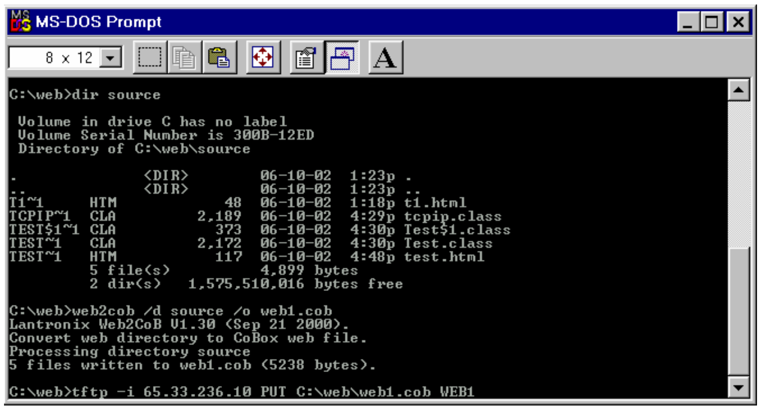
```
<HTML>
<BODY>
<CENTER>
<APPLET CODE="Test.class" WIDTH="862" HEIGHT="512"></APPLET>
</CENTER>
</BODY>
</HTML>
```

2. Copy the Java classes and new HTML file into the \web\source directory:

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

3. 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
```

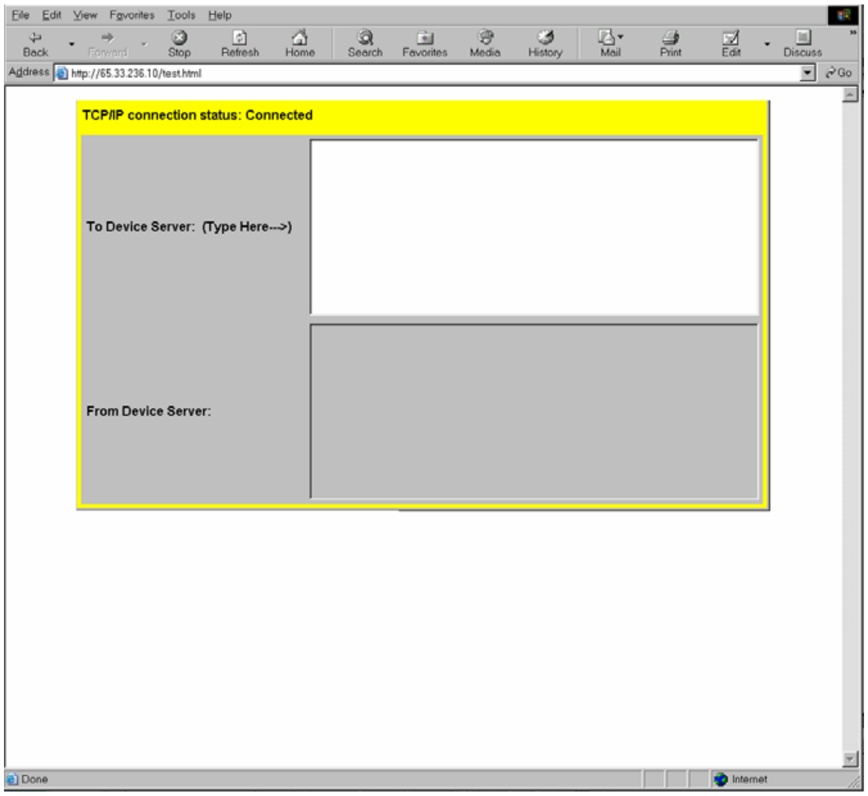


```
C:\web>dir source

 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
```

4. 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.



Address http://65.33.236.10/test.html

TCP/IP connection status: Connected

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

From Device Server:

File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss
Go
Done
Internet

8: Summary

The CoBox family supports an internal web server that is used for storage and retrieval of documents, images, and Java applets. An experienced web programmer can create custom web pages to interact with any serial device and present the information to a user in a user friendly web interface.