Application Note 199
Designing a Networked On/Off Switch Using TINI

www.maxim-ic.com

INTRODUCTION

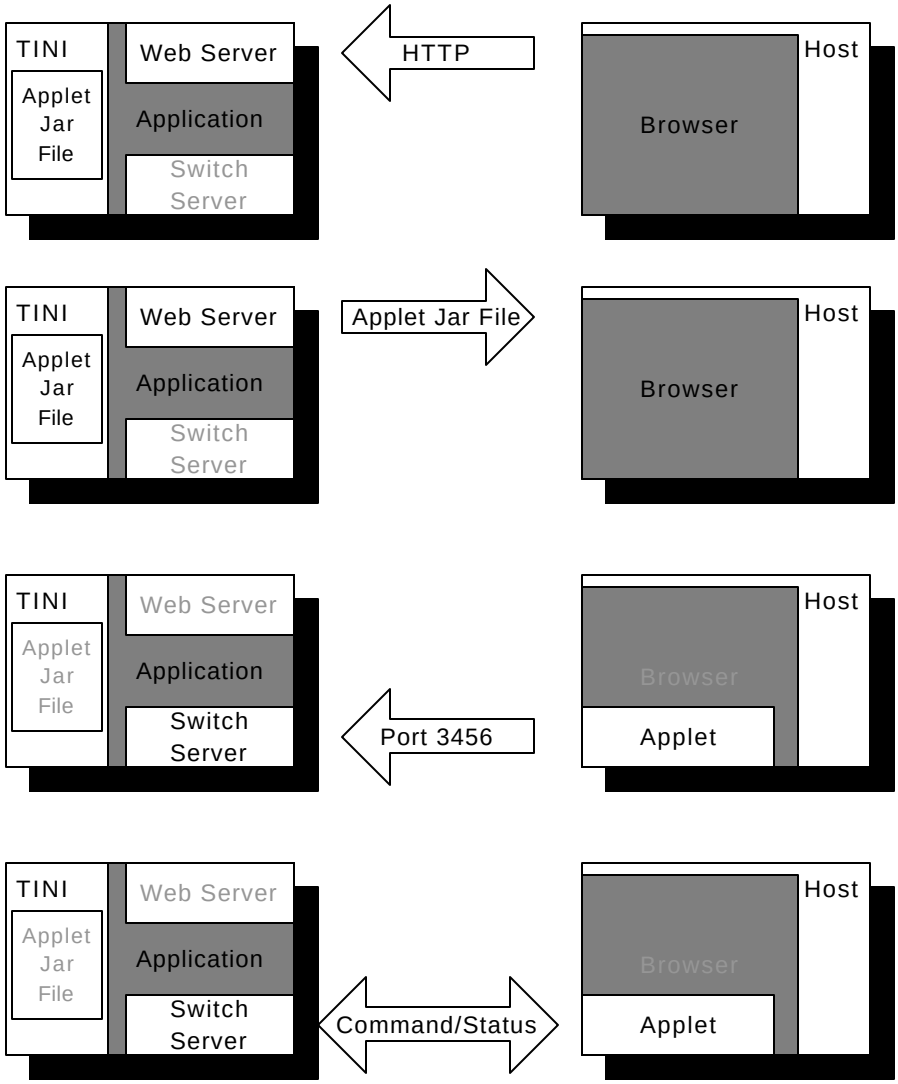
The Tiny InterNet Interfaces (TINI®) platform provides the TCP/IP network stack and local control capabilities needed to design IP networked switches. The addition of a Java™ runtime environment greatly reduces the complexity with which small sensors and actuators can be accessed and controlled remotely over a network. This application note presents an IP On/Off switch constructed with a TINIm390 Verification Module and a simple relay circuit. The circuit is controlled using a slightly modified version of the TINIWebServer example that is executed directly by the TINI Runtime Environment and an applet served to the host machine. The applet opens a two-way communication channel back to TINI for commands and status and displays a graphical user interface to give the user simple remote control of the relay.

SYSTEM OVERVIEW, SOFTWARE

The switch control application on TINI uses the com.dalsemi.tininet.http.HTTPServer class to implement a simple web server. The web server is used only as a means to transfer an applet to a remote host, which is executed within the host's browser. The applet establishes a two-way TCP connection (over port 3456) with TINI for the purpose of transferring commands and status. The applet also provides a graphical user interface to display the controls and status.

TINI is a registered trademark of Dallas Semiconductor.
Java is a trademark of Sun Microsystems.

Figure 1. SOFTWARE BLOCK DIAGRAM



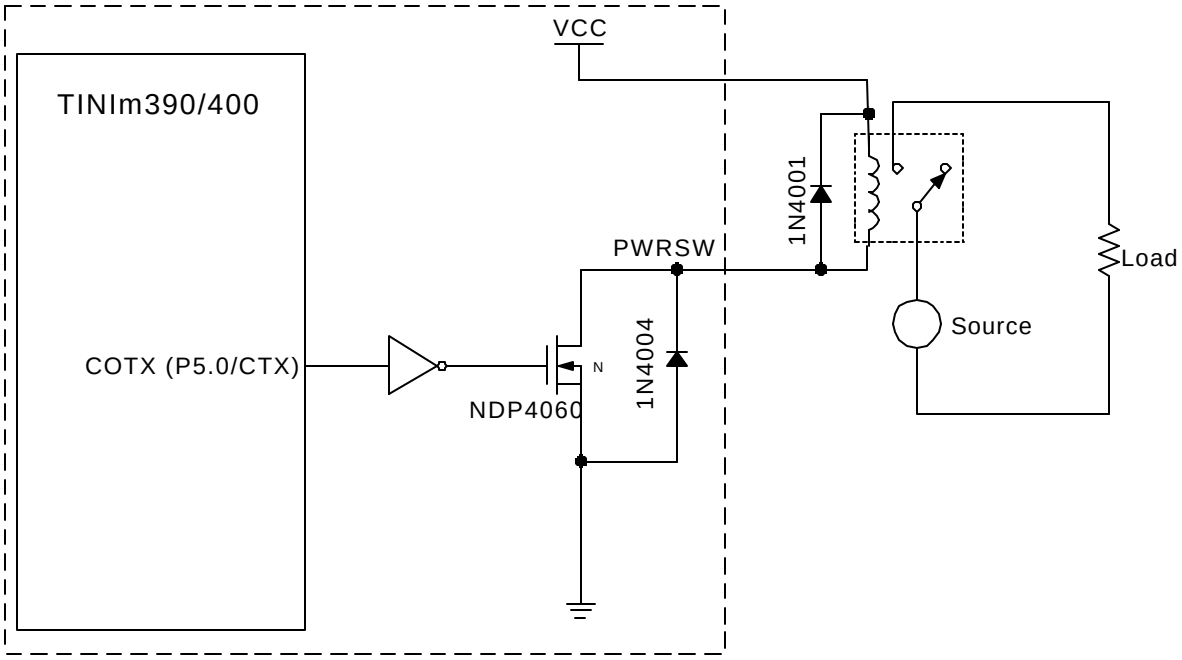
SYSTEM OVERVIEW, HARDWARE

Figure 2 shows a diagram of the On/Off control circuit interfacing a TINIm390[1] Verification Module. The TINIm390 provides Ethernet network connectivity and controls the switch through the port pin P5.0. Any available port pin will work equally well.

In the configuration shown below, an N-channel power FET controls the relay by switching current from the relay to ground. The relay and FET can be resized to match circuits with various current and voltage requirements. The relay can be omitted altogether if the external circuit does not need to be isolated from TINI's power supply. The diodes protect against voltage spikes from the relay's coil while the switch is changing state. The portion of the circuit inside the dashed box is actually part of the socket board design on some revisions of the TINI E-series socket.

[1] Details of the TINIm390 are contained in *Application Note 195*.

Figure 2. HARDWARE BLOCK DIAGRAM



TINI'S SWITCH CONTROL APPLICATION

The application consists of four classes that make up the switch control and the web server interface portions of the application. The PowerSwitch class interfaces directly to the hardware using TINI's com.dalsemi.system.BitPort API class. The WebWorker class comes directly from the TINIWebServer example and is responsible for servicing HTTP connections that arrive. The SwitchWorker class manages all command and status communications between the applet and TINI. The TINIWebServer class drives the application, binding together the operation of the individual classes.

The PowerSwitch class is the application's interface to the hardware. It creates a BitPort object for the processor port pin P5.0 in the constructor. There are two methods implemented in this class. The on method sets the port pin state that is used to apply voltage to the relay's coil. The off method clears the port pin state removing the voltage from the coil. The relay in Figure 1 is SPDT (Single Pole, Double Throw) and can be used in the normally closed position, the normally open position or can be used to switch the external voltage source between the two positions. The on and off methods would need to be modified to accommodate the normally closed circuit. In fact, an extra `boolean` variable, "invert," could be added to this class to indicate whether on/off corresponds to BitPort's set or clear method. An extra method setInvert would be needed to initialize the invert variable. The hardware diagram shows the normally open circuit being used.

The WebWorker class is the application's interface to the network. It simply sets up and drives an com.dalsemi.tininet.http.HTTPServer object which spawns a thread to service each incoming connection. This class is essentially unchanged from the TINIWebServer example in the TINI SDK (Software Developer's Kit) distribution.

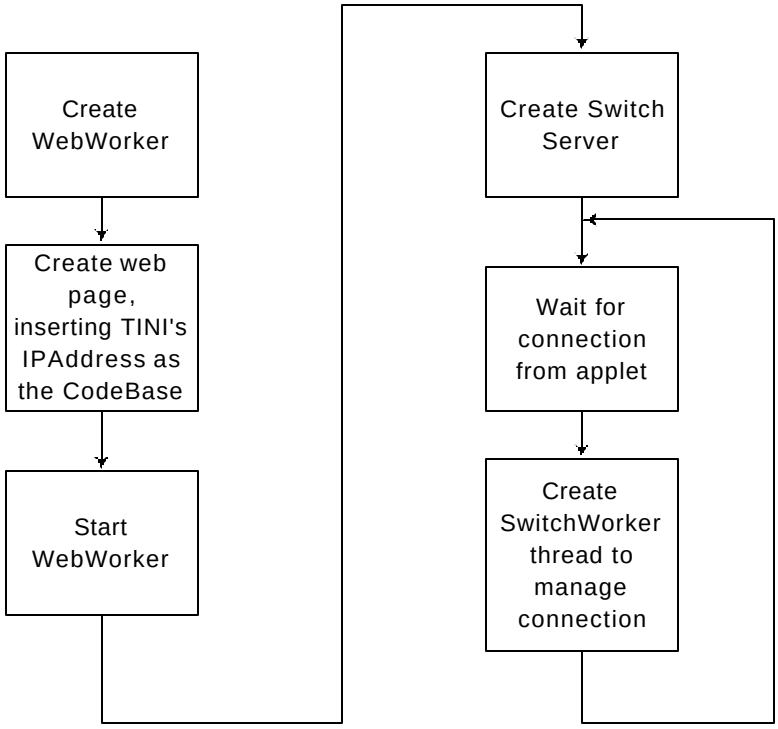
The TINIWebServer class ties the network and the hardware classes together to allow control of a remote switch. The drive() method sets up the web server by creating a WebWorker thread and creating the web page "index.html." The primary use of the web page is to download and run the applet on the host workstation. If the index page contained static information the application would not have to create this page. It could have simply been copied to the web server's root along with the jar file containing the applet. The one parameter in

this page that changes on every TINI is CODEBASE. The applet uses this information to connect back to TINI on a separate server socket. A custom web page could be created and uploaded to each TINI that is installed in the field but it is much easier to have the application on each TINI create it. The createIndexPage() method creates index.html, inserting TINI's IP address into the CODEBASE section using three writes

```
index.write(indexTop.getBytes(), 0, indexTop.length());
index.write(InetAddress.getLocalHost().getHostAddress().getBytes());
index.write(indexBottom.getBytes(), 0, indexBottom.length());
```

The first and third write simply copy the static portions of the web page into the file. The second write copies TINI's IP address to the CODEBASE section of the file. After the web server is set up and the page is created the web server is started. A server socket is created to handle incoming connections from the downloaded applets and the serviceConnection method is called each time an applet connects with TINI. The serviceConnection method creates a new instance of SwitchWorker and passes the socket to this class. The SwitchWorker constructor creates a new thread to manage the connection between the host applet and TINI. Handling the next incoming connection is done using the `serviceConnection` method which transfers control to the `drive` method.

Figure 3. TINIWEBSERVER



The SwitchWorker class manages all communications between the applet and TINI. It loops forever until the connection is dropped, performing the following steps:

1) Block on read(), waiting for a command byte from the applet.
2) If the command byte is 0 then turn the switch off. If the byte is 1 then turn the switch on.
3) Read the current switch state and send it back to the applet.

This algorithm can be extended to multiple switches by using each of the seven lower bits of the command byte to represent the state of separate switches. The most significant bit is reserved to indicate a read only operation.

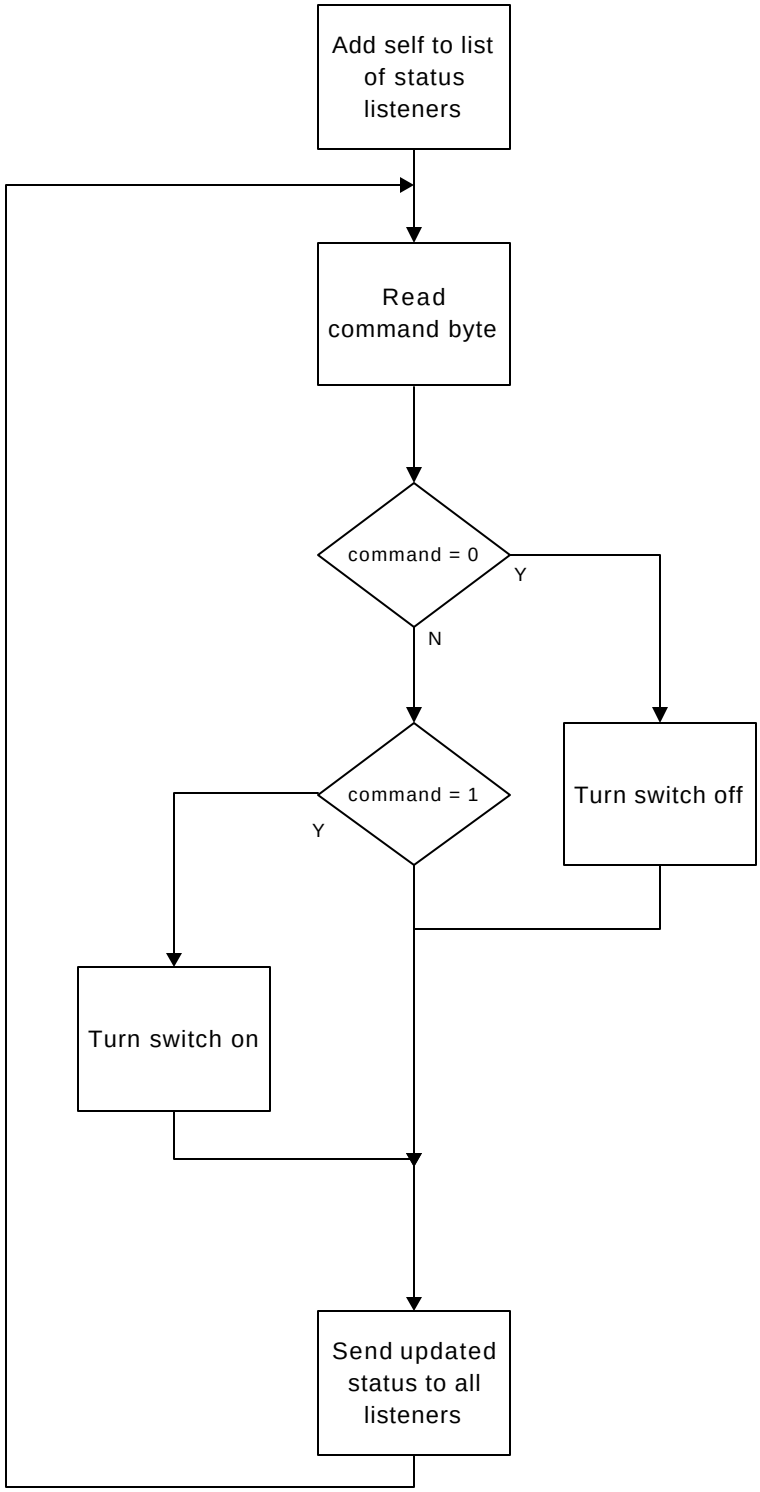
Figure 4. COMMAND BYTE FOR MULTIPLE SWITCHES

R/W	Sw 6 state	Sw 5 state	Sw 4 state	Sw 3 state	Sw 2 state	Sw 1 state	Sw 0 state

Modified Command Byte

The algorithm can also be extended to allow multiple applets to connect to a single TINI at the same time. The SwitchWorker simply needs to keep a Vector of listeners. Each time a single applet sends a command to change the state of a switch; TINI sends the status back to all of the applets that are currently connected.

Figure 5. SWITCHWORKER



THE HOST APPLET

An applet is used on the host for several reasons. Applets provide a rich set of graphical objects to display controls and status.

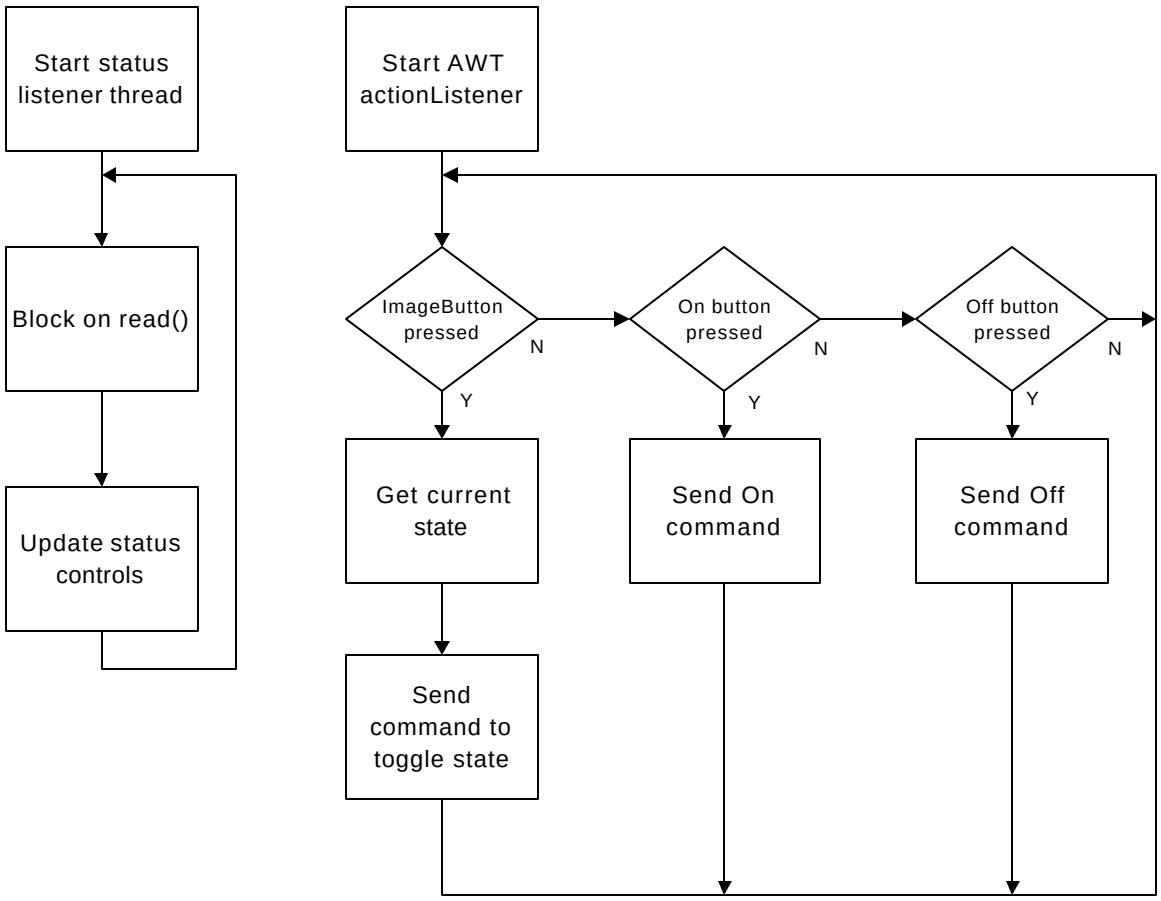
TINI's com.dalsemi.tininet.http.HTTPServer class is fast and small. However, it only supports the HTTP GET operation. This means that data can only flow in one direction—from TINI to the host. This application requires data flow in both directions. Commands are sent from the host to TINI and status is sent from TINI to *all* connected hosts.

There is no protocol overhead involved in the communication between the host and TINI. Commands are one byte and the status is one byte, allowing for very quick control and status responses.

There are two classes in the host applet. The SwitchControl class is the primary class in the applet. It creates all of the graphical elements displayed on the web page and handles the host side network communications. The ImageButton class creates a graphical toggle button that displays one of two bitmaps depending on the status of the switch. This should be the only control that is needed for control and status of the TINI switch control, but the behavior of applets varies from one browser to the next. Additional simple On and Off buttons and a simple text window for status were added to allow for wider browser coverage. The `ImageButton` and the On/Off buttons perform the same control function. The bitmaps and status window perform the same status function.

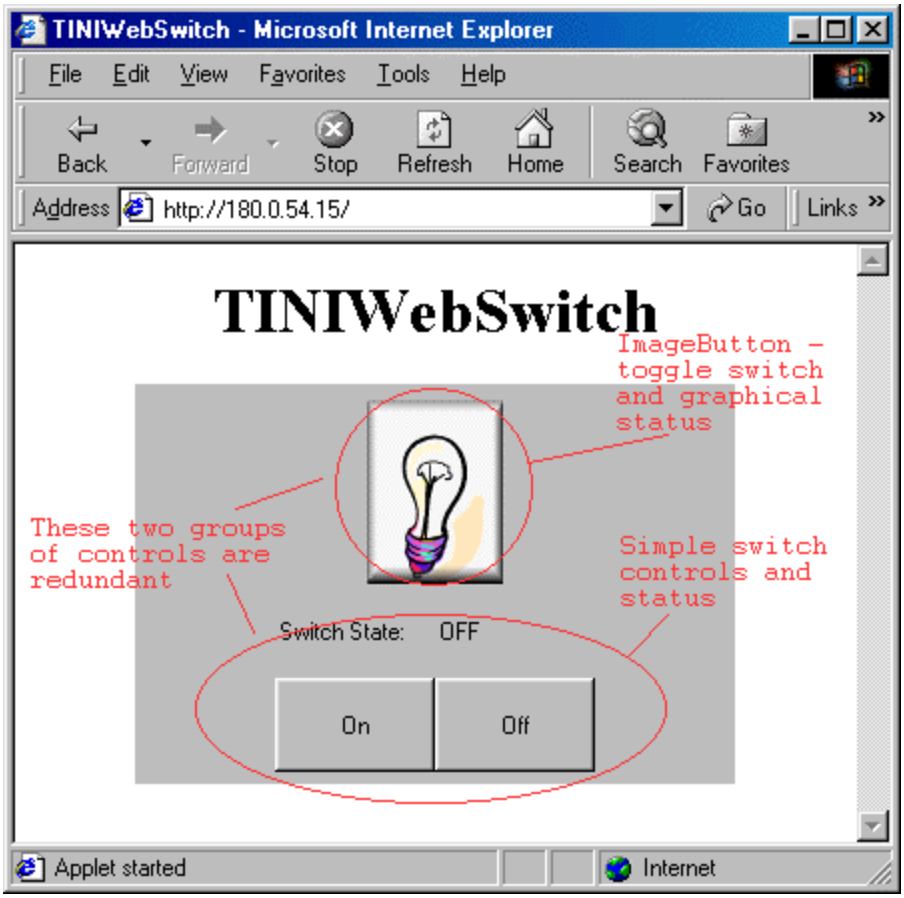
The SwitchControl class creates a status listener thread after creating the graphical elements. This thread sleeps, blocked on read, waiting for status from TINI. When the thread unblocks, the ImageButton bitmap and the status window are updated and the method loops back to the top to wait for the next status byte. The Applet event thread drives the actionPerformed method. Each time one of the graphical buttons are pressed this method is called. If the ImageButton triggered this call the current state is toggled and the On or Off command is sent to TINI. If the On/Off button triggered the call an On/Off command is sent.

Figure 6. SWITCHCONTROL



The ImageButton class is simply an exercise in AWT component programming.

Figure 7. APPLET—GRAPHICAL INTERFACE



ASSOCIATED FILES

Files associated with AN199 are located at ftp://ftp.dalsemi.com/pub/tini/appnotes/AN199/AN199.tgz.

CONCLUSION

With the TINI Runtime Environment, Java software, and a simple relay circuit, it is easy to implement a network-controlled, simple on/off switch. With a very large selection of circuit components available, the application of a network on/off switch can be left of to fertile imaginations. Anything from light bulbs to industrial equipment can be controlled from any location with network access.

DALLAS SEMICONDUCTOR/MAXIM CONTACT INFORMATION

Company Addresses:
Maxim Integrated Products, Inc.
120 San Gabriel Drive
Sunnyvale, CA 94086
Tel: 408-737-7600
Fax: 408-737-7194

Dallas Semiconductor
4401 S. Beltwood Parkway
Dallas, TX 75244
Tel: 972-371-4448
Fax: 972-371-4799

Product Literature/Samples Requests:
800-998-8800
408-737-7600

World Wide Website:
www.maxim-ic.com

Product Information:
http://www.maxim-ic.com/MaximProducts/products.htm

Ordering Information:
http://www.maxim-ic.com/BuyMaxim/Sales.htm

FTP Site:
ftp://ftp.dalsemi.com

APPENDIX A

The following is a listing of the `SwitchWorker` class. This TINI class is responsible for the flow of command and status information between TINI and the host applet.

```
/*---------------------------------------------------------------------------
 * Copyright (C) 2001, 2002 Dallas Semiconductor Corporation,
 * All Rights Reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL DALLAS SEMICONDUCTOR BE LIABLE FOR ANY CLAIM, DAMAGES
 * OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
 * ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * Except as contained in this notice, the name of Dallas Semiconductor
 * shall not be used except as stated in the Dallas Semiconductor
 * Branding Policy.
 *---------------------------------------------------------------------------
 */
  import java.io.*;
  import java.net.*;
  import java.util.*;

/** This class performs the TINI side communications for the On/Off switch.
 */
  public class SwitchWorker
    extends Thread
  {
    // socket for communications between applet and TINI
    Socket s;
    // streams for use with socket
    InputStream in = null;
    OutputStream out = null;
    // lock for switch synchronization
    Object switchLock = new Object();
    // all applets listening for status
    Vector listeners;
    /**
      */
    SwitchWorker(Socket s)
    {
      this.s = s;
      listeners = TINIWebServer.listeners;
      // start worker thread
      start();
    }
    /**
```

```
 */
public void notification(int status)
  throws Exception
{
  out.write(status);
}
/**
  */
public void run()
{
  // add current thread to list of listeners
  listeners.addElement(this);
  Socket switchControlSocket = s;
  // create actual switch control object
  PowerSwitch powerSwitch = new PowerSwitch();

  try
  {
    in = switchControlSocket.getInputStream();
    out = switchControlSocket.getOutputStream();

    while(true)
    {
      // block until command arrives
      int newState = in.read();

      synchronized(switchLock)
      {
        System.out.println(this.toString()+", Request: "+newState);

        // actual switch control
        switch(newState)
        {
          case 0:
            powerSwitch.off();
          break;
          case 1:
            powerSwitch.on();
          break;
        }

        // verify current state
        boolean currentState = powerSwitch.getState();
        int     size = listeners.size();

        // notify all listeners of the current state
        for(int i = 0; i < size; i++)
        {
          System.out.println("notifying element "+i);
          ((SwitchWorker)listeners.elementAt(i)).notification(currentState?
                                                              1: 0);
          if(listeners.elementAt(i) == listeners.lastElement())
            break;
        }
      }
    }
  }
  catch(Throwable t)
  {
    System.out.println(t);
  }
```

```
        finally
        {
          try
          {
            in.close();
          }
          catch(Throwable _){}
          try
          {
            out.close();
          }
          catch(Throwable _){}
          try
          {
            switchControlSocket.close();
          }
          catch(Throwable _){}

          listeners.removeElement((Object)this);
        }
      }
    }
```

APPENDIX B

The following is a listing of the `SwitchControl` class. This host class is responsible for the flow of command and status information between TINI and the host applet.

```
/*---------------------------------------------------------------------------
 * Copyright (C) 2001, 2002 Dallas Semiconductor Corporation,
 * All Rights Reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL DALLAS SEMICONDUCTOR BE LIABLE FOR ANY CLAIM, DAMAGES
 * OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
 * ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 * OTHER DEALINGS IN THE SOFTWARE.
 *
 * Except as contained in this notice, the name of Dallas Semiconductor
 * shall not be used except as stated in the Dallas Semiconductor
 * Branding Policy.
 *---------------------------------------------------------------------------
 */

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

/** This class performs the host side applet communications for the On/Off switch.
 */
public class SwitchControl
  extends Applet
  implements ActionListener
{
  // command bytes
  public static final int CMD_READ = 0x0ff;
  public static final int CMD_WRITE_OFF = 0;
  public static final int CMD_WRITE_ON = 1;

  // socket for communications between applet and TINI
  Socket       s;
  // streams for use with socket
  OutputStream out;
  InputStream  in;
  // reference to state text
  Label        stateLabel;
  // listener thread
  Thread       listenThread;
```

```
ImageButton  imageButton;
boolean      switchOn;

/**
 */
public void init()
{
  try
  {
    // Make the connection to TINI
    s = new Socket(getCodeBase().getHost(), 3456);
    in = s.getInputStream();
    out = s.getOutputStream();
  }
  catch(Throwable t)
  {
    System.out.println("init: "+t);
  }
}
/**
*/
public void paint(Graphics g)
{
  if(imageButton != null)
    imageButton.repaint();
}
/**
 */
public void start()
{
  try
  {
    // set up the AWT controls
    Panel p = null;
    Panel q = null;
    p = new Panel(new GridLayout(2, 1));

    imageButton = new ImageButton(getImage(getCodeBase(),"on.gif"),
                                  getImage(getCodeBase(),"off.gif"));
    imageButton.addActionListener(this);
    p.add(imageButton);

    q = new Panel(new GridLayout(2, 2));

    q.add(new Label("Switch State: "));
    q.add(stateLabel = new Label("Unknown"));

    Button  b = new Button("On");
    b.addActionListener(this);
    q.add(b);
    b = new Button("Off");
    b.addActionListener(this);
    q.add(b);
    p.add(q);

    add(p);
    imageButton.repaint();

    // create status listener
    listenThread = new Thread() // Thread definition
    {
```

```
        public void run()
        {
          try
          {
            while(true)
            {
              // get the current status
              switch(in.read())
              {
                // update AWT controls
                case 0:
                  stateLabel.setText("OFF");
                  switchOn = false;
                  System.out.println("received Off");
                break;

                case 1:
                  stateLabel.setText("ON");
                  switchOn = true;
                  System.out.println("received On");
                break;
              }

              // update ImageButton
              imageButton.setState(switchOn);
            }
          }
          catch(Throwable t)
          {
            System.out.println("listenThread: "+t);
          }
        }
      }; // end of Thread definition

      // start listener thread
      listenThread.start();
      // get current state
      out.write((byte)CMD_READ);
    }
    catch (Throwable e)
    {
      System.out.println("start: "+e);
    }
  }

  /**
   */
  public void actionPerformed(ActionEvent event)
  {
     try
     {
       // determine the source of the event
       if(event.getSource() instanceof ImageButton)
       {
         stateLabel.setText("Checking");

         // toggle the current state
         // and send it to TINI
         if(switchOn)
         {
           System.out.println("Sending Off");
```

```
            out.write((byte)CMD_WRITE_OFF);
          }
          else
          {
            System.out.println("Sending On");
            out.write((byte)CMD_WRITE_ON);
          }
        }
        else if (event.getSource() instanceof Button)
        {
          stateLabel.setText("Checking");

          Button item = (Button)event.getSource();

          // send the desired state to TINI
          if (item.getLabel().equals("On"))
          {
            System.out.println("Sending On");
            out.write((byte)CMD_WRITE_ON);
          }
          else
          {
            System.out.println("Sending Off");
            out.write((byte)CMD_WRITE_OFF);
          }

        }
      }
      catch (Exception e)
      {
        System.out.println("actionPerformed: "+e);
      }
    }
    /**
     */
    public void cleanup()
    {
      if(listenThread != null)
        listenThread.stop();

      try
      {
        if (out != null)
        {
          out.close();
        }

        if (in != null)
        {
          in.close();
        }

        if (s != null)
        {
          s.close();
        }
      }
      catch (IOException e)
      {
        System.out.println("cleanup: "+e);
      }
```

```
    }
    /**
     */
    public void stop()
    {
      cleanup();
    }

    /**
     */
    public void destroy()
    {
      cleanup();
    }
  }
```