Application Note

AJAX Web Page Control and Monitoring via the Internet Using the Zilog® eZ80AcclaimPlus!™ Embedded Web Server

AN030501-0810

Abstract

Today's web users expect interactive and rich content with quick response time in their web experiences. As web technologies have evolved to deliver rich, interactive experiences, the client has increased its processing power and capabilities, allowing web servers to become more dynamic and responsive.

This application note describes how easy it is to implement the eZ80Acclaim*Plus!*™ embedded web server to deliver web pages using asynchronous JavaScript and XML technologies (AJAX) for server communications.

The Zilog ZTP and RTZ stacks implement most of the low-level requirements for the web server so that the focus remains on the response to AJAX requests and monitoring and controlling the hardware. The web pages use JavaScript and cascading style sheets (CSSs) to provide dynamic web pages at the client that allow monitoring and control of the eZ80F91 development board in the background.

Note: *The source code (AN30501-SC01) associated with this application note has been tested with ZDS II—eZ80Acclaim! 5.1.1.*

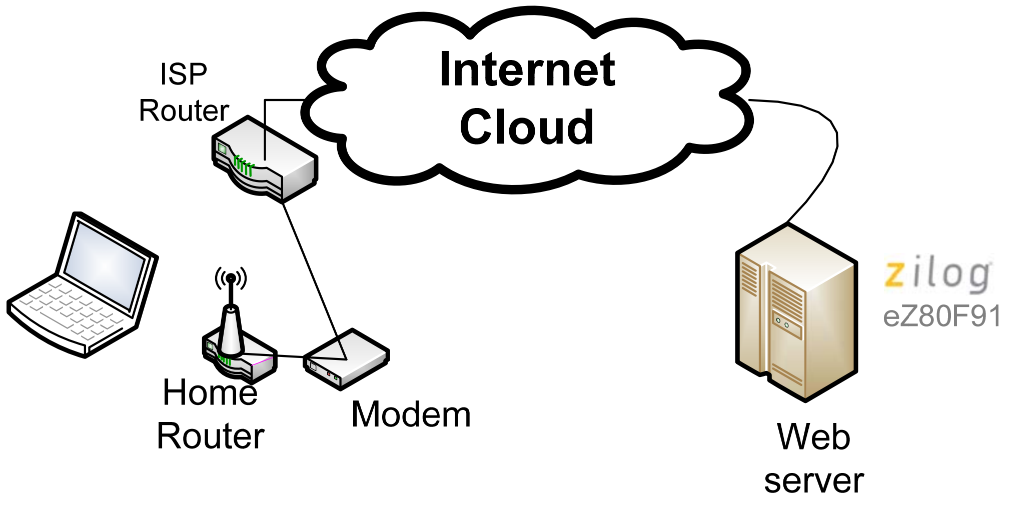
Features

The following are the main features of the eZ80F91:

- Acts as a dynamic web server using ZTP and RZK on the eZ80Acclaim*Plus!*™ microprocessor
- Implements AJAX technologies within the web server
- Updates individual locations without reloading entire page
- Monitors and controls development board hardware via AJAX requests

Discussion

The purpose of a web server is to deliver a response to a client's request (client-server model). Typically, with browsers, HTTP (Hypertext Transfer Protocol) is the backbone communication protocol used. The advantage of using HTTP is the ability to deliver many different technologies using one communication protocol. The traditional client-server model is shown in the following figure. In this application note, the term *server* refers to the eZ80F91 with the TCP/IP stack.



Web browsers have been providing increasingly richer user experiences. To accomplish this, both the browser and web server technologies have improved. The old HTML static pages have been replaced with graphical elements, extensive client-side scripting, dynamic client-side formatting, and so on. The web servers have improved their abilities with server-side scripting, dynamic processing, and delivering audio and video content in ways to keep users interested. The more demand on content from the user, the more dynamic the web server needs to be. The clients have become extremely powerful as well. This allows extensive client-side processing, offloading significant power from the server. One “technology” that has helped propel these capabilities is AJAX. Although AJAX is a combination of technologies, this mixture can be a very powerful tool.

AJAX (Asynchronous JavaScript and XML) is a group of technologies that improves the exchange of data between the web client and the web server. In a typical web application, every time information needs to be sent to the web server (such as when a date changes or a user submits a filled-out form), the information is sent, and the entire page is redrawn. This duplicate processing slows down the web pages. If the content of the pages expires quickly (as is typical in a interactive web application), all the supporting files—such as the images, libraries, style sheets, and so on—also have to be reloaded. AJAX gives the web application designer the ability to only update sections or parts of the page without having to reload everything. This is accomplished by a background data transfer when a web page changes. The result is fewer data transfers (because only the data that has changed is transferred), and the data transfers are quicker. The user gains a richer web experience. AJAX has been implemented in most Internet browsers through the `XMLHttpRequest` object. The eZ80F91 web server passes the dynamic web requests from the web client to the application for processing and client responses. It is this ability that opens the opportunity to exploit AJAX technologies.

With AJAX, the eZ890F91 web server can uses the dynamic page capabilities of the ZTP website to send XML documents reporting the status of the server hardware for each `GET` request. XML documents require few resources to display, which allows the server to concentrate on the hardware portion of the unit and servicing other web requests. The XML document, however, has all the information necessary for a client to present the data to the user in a rich environment.

The web server relies heavily on the ZTP and RZK libraries that are included with Zilog Developer Studio II (ZDS II—eZ80Acclaim! in this case). ZTP integrates a rich set of networking services with an efficient, real-time operating system RZK (RTOS). The operating system is a compact, preemptive, multitasking, multithreaded kernel with interprocess communications (IPC) support and soft real-time attributes.

The ZTP software suite provides the following features:

- Industry standard, RFC-compliant protocols
- Core protocols: IPv4, TCP, UDP, DHCP/BOOTP, ICMP, IGMP, ARP, and RARP
- Additional protocols: HTTP, TFTP, SNMP, TELNET, SMTP, DNS, TIMEP, SNTP, PPP, and HDLC
- Optional protocols: SSL server, SNMP V3, and HTTPS
- Interconnects: UART(x2), I^2C, and SPI
- FTP server and client services using an embedded Flash file system supporting multiple disk volumes
- Local or remote runtime debugging OS command shell
- Dynamic memory allocation support

Hardware Architecture

The hardware used for this application note is the eZ80F91 Series Development Kit (eZ80F910300ZCOG). The hardware includes a 5x7 LED matrix, which is monitored and controlled.

The eZ80Acclaim*Plus!*™ integrates a high-performance Flash core with a fast 10/100 BaseT EMAC. The power-efficient, optimized pipeline architecture features a high-performance core that operates up to a speed of 50 MHz and offers on-chip Flash memory, SRAM, an Ethernet MAC (EMAC), and rich peripherals. The 24-bit linear addressing capability helps simplify code development and enables the code to execute efficiently with the TCP/IP/RZK software stack.

eZ80Acclaim*Plus!*™ Key Feature Summary

- 50-MHz high-performance eZ80® CPU core
- On-chip 10/100BaseT Ethernet MAC
- 256-KB Flash program memory with an extra 512 bytes of device configuration Flash
- 16-KB total on-chip high-speed SRAM
- 24-bit linear addressing
- Low-power PLL and 32-KHz on-chip oscillator
- Interfaces supported: 32-bit GPIO, UARTs (x2), I^2C, SPI, and IrDA-compatible Infrared Endec
- Power management: HALT/SLEEP modes with selective peripheral power-down controls

Software Implementation

The software implementation is described in two sections:

- The server side includes everything used to produce the web server that will respond to the client's requests.
- The client's script implementation provides the ability to dynamically monitor and control the remote hardware.

Server Side

With the ZTP website stack, the server-side implementation is quite simplistic. Most of the hard details of providing a web server are already built, so the developer can concentrate on your specific needs.

To set up the website using the ZTP and RZK stacks requires only a few steps:

1. Adjust the configuration file.

   The ZTP and RZK libraries have a `Conf` folder that contains different configuration files that can be modified for the particular implementation. For this application, copies of the files that will be modified are saved into the local directory, so the originals remain unchanged. The other files have been left in their original directory and are just included in the project.

   Unmodified Configuration Files

`Data_Per_Conf.c`	Controls configuration for data persistence
`Emac_conf.c`	Defines the EMAC driver and settings
`eZ80eval.c`	Provides generic routines for `putch`/`getch` on console
`ez80Hw_Conf.c`	This has the initialization of the hardware for RZK.
`F91PhyInit.c`	Physical implementation for the F91 Ethernet port (hardware specific)
`Get_heap.asm`	Default heap handling procedures
`Rtc_conf.c`	Configuration for the Real Time Clock
`Shell_conf.c`	Configuration for the Shell application. You can remove, add, and change the shell functionality through this file.
`Tty_conf.c`	Configuration for the TTY devices
`Uart_conf.c`	Configuration for the UART devices
`ZTPuserDetails.c`	User details for login details

   Modified Configuration Files

`ZTP_Config.c`	This contains the DHCP, IP, and other interface settings and must be modified for local networks.
`RZK_conf.c`	Configuration file for RZK and subcomponents

2. Add the initialization routines.

   `Main.c` contains only the `main()` function that C programmers will recognize. The function is used to set up the RZK functionality, initialize any hardware and devices, create the application entry thread, and then start the RZK scheduler to run the threads.

   The meat of the application setup is in the `appentry.c` file. The `ZTPAppEntry()` function is a predefined name for a user function that is called from RTZ's `CreateZTPAppThread()` call. This function is designed to initialize all services that are threaded, such as timers, application threads, and network stacks. The `ZTPAppEntry()` function initializes the network stack by calling the `networkInit()` function. The `networkInit()` function handles the initialization of network interface tables and DHCP and then starts the web server by calling `http_init()`. Any other initializations—such as console devices, shell applications, and so on—are completed here as well.

   After the initializations are all completed, the application exits. When the application exits, the control returns to `main()`, which calls the `RZK_KernelStart()` function. This function starts the scheduling and handles all the threads. There is no return from this function because the application itself is strictly event driven, so no loop is needed. The kernel takes care of handling the events and running the threads as necessary.

3. Define the website.

   The website definition is nothing more than a list of web page names that the server will send to the client in response to the `GET` request. The information is in the `website.c` file.

   There are two sections of the website information:
   - The mime type table is used when the web server is serving pages from a file system.
   - The web page table contains all the web pages that a client could request. The format is type, request address, mime, reference.

   The web page type can either be static or dynamic:
   - If it is static, the mime type is put in the header, and the reference page is sent in response to the `GET` request with the indicated address.
   - The dynamic page type is where the extensibility exists. On response to a `GET` request for that address, the mime type is filled in the header, and the request structure is passed to the function reference. Nothing is sent to the client; the function that is being called is responsible for that. This allows one to craft the response in any way that makes sense (as long as it follows the HTTP protocol rules).

That is all there is to setting up a web server with the ZTP/RZK stack.

Handling the Dynamic Page

AJAX offers the ability of offloading the real work for display to the user with two dynamic page requests:

- `update.ajax`
  The `update.ajax` request calls the `updateRequest()` function. This function prepares an XML document defining the server, server type, and the status of all of the hardware. The function then sends the XML document out to the client.

- `command.ajax`
  The `command.ajax` request calls the `commandRequest()` function. This function expects parameters to be included. Clients send parameters by appending `?` to the name and using `key=value` strings, separated by the `&` sign. A typical request would look like the following:

  ```
  command.ajax?command=2&val=0
  ```

  The `commandRequest()` function parses the command and value and then sets the hardware accordingly. It builds an XML page with the server information and the new status of the item request and returns that data to the client.

  Because XML pages are just text files that are made up of tags – values – end tags, very little processing is necessary. The `sprintf` functions (located in the `ajaxrequest.c` file) do most of the work.

The extensions used here are just an example; it doesn't matter which extensions are used as long as the extensions match.

Specific Implementation Details

Main.c

The `Main()` function calls the `RZK_KernelInit()` function to initialize the kernel. Then, the function calls other functions to initialize all the hardware being used:

1. The Serial 0 device (RTZ library call) is initialized.
2. The TTY device (RTZ library call) is initialized.
3. The hardware (discussed in the `hdwareif.c` section on page 6) is initialized.
4. The EMAC device (RTZ library call).

Then, the function calls the `CreateZTPAppThread()` function to initialize the application stacks (see step 2 on page 4). Finally, the function calls the `RZK_KernelStart()` function to start the scheduling and thread processing.

AppEntry.c

This file handles the application entry initialization functions.

The `ZTPAppEntry()` function handles the initialization of stacks, threads, and timers. First, the function calls the `networkInit()` function to initialize the network stacks. Then, the function calls the `InitMatrixTimer()` function (discussed in the `hdwareif.c` section on page 6) to start the timer for the LED Matrix output. Then, the function opens the serial port and assigns the TTY device. The `shell_init()` function (RTZ library call) is called to start the shell application on the TTY device. At this point, everything has been initialized, so the function exits with an “OK” return code.

The `networkInit()` function initializes the network stacks. First, the function initializes the network interface tables through the `nifDriverInit()` function (RTZ library call). Then, if DHCP is enabled, the function calls the `DHCP_Init()` function (RTZ library call) to initialize DHCP and get an address. The `SpiderZInit()` function is called to keep track of IP addresses; then the `nifDisplay()` function (RTZ library call) is called to display the IP address on the console. Now that the network stack is initialized, the function can call the `http_init()` function (RTZ library call) function. This function initializes the web server stack and starts the web server with the following information:

- `http_defmethods`
  These are default methods to be called on the receipt of requests (referenced from `http.h`).
- `httpdefheaders`
  These are the default headers (referenced from `http.h`).
- website
  This is the definition of the website in a variable structure defined in `website.c`.
- port number to listen on

The `OpenSerialPort()` function opens the serial port and assigns it to the TTY device. The device capabilities structure is allocated, and the handle for the device is the CONSOLE (defined in the `RZK_Conf.c` file). The device is then opened through the call to `RZKDevOpen()` with the resulting device structure stored in the `TTYDevID` variable.

Hdwareif.c

This file has the development-board-specific hardware functions. By keeping the hardware separate, it will be easier to reuse the rest of the files with other hardware.

The `InitHardware()` function initializes all the hardware. Because the relay and LCD text are emulated, variables are set up here to hold them. The LED matrix is initialized with all LEDs off. Any hardware initialization—such as setting pins and functions—are added here.

The `Set_LED()` function sets the LED array value.

The `Set_Relay()` function sets the relay variable. Because the relay is emulated, there is no hardware to set. Instead a message is displayed on the console to reflect that the state of the relay has changed.

The `LCDPrintString()` function "prints" a string on the LCD. Because the printing is emulated, the function stores the string and displays the string on the console.

LED Matrix Functionality

On the development board, there is a matrix of 35 LEDs arranged in a 5x7 format. The rows (anodes) and columns (cathodes) are controlled by two D-type flip-flops. Because all of the columns are wired together, only one "column" can be active at time. By turning on each row quickly, it looks like all of the rows are on.

The row flip-flop address is at `0x800000`, and the column flip-flop is at `0x800001`. By setting these to the masked values, the correct LEDs are lit in the specific column. There are seven rows of LEDs in each column. The bit representation of what is to be on and off is stored in the `Led[column]` variable. A timer is used to set both the column and the row values on the flip-flops as fast as possible. The tick value in the `RZK_conf.c` file is adjusted to 5 ms. This prevents the flicker you get from the default 10-ms tick. If the tick is too fast, there can be stability issues because of more thread switching. If the tick is too slow, the LEDs on the matrix flicker.

The `MatrixTimer()` function sets the flip-flops. Each time the function is called, the `CurCol` variable is incremented so that the function will rotate through the five columns. The function sets the bit to the correct position for the current column and then inverts it. The function sets the `MATRIX_ROW` value to the LED array variable specified by `CurCol` (the order is in reverse order, so `CurCol` needs to be subtracted from 4 to keep it aligned). Then, the function sets the MATRIX_COL port. `CurCol` is incremented and checked to see if the value has exceeded the maximum count. If it has, `CurCol` is reset to 0.

The `InitMatrixTimer()` function sets up the timer through the `RZKCreateTimer()` function. This function requires the type of timer ("RZKappTimer"), the function to call when the timer is reached, the initial delay in ticks, and the timeout period in ticks (how long the timer to be set for). Then, the function initializes `CurCol` to 0 and enables the timer.

Website.c

This contains all the variables for the website. The pages located in the website folder of the project will be compiled to a variable with the format of `filename_ext`. This format allows references to be created to external variable that will be used in the website. The website is an array of pages with the type, request name, content type, and data to send. If the page is the HTTP_PAGE_STATIC type, the referenced variables are sent when the web server receives a request for that page. If the page type is HTTP_PAGE_DYNAMIC, the page calls either the `updateRequest()` or `commandRequest()` function to send the data. These functions are located in the `ajaxrequest.c` file.

<u>Ajaxrequest.c</u>

The `updateRequest()` function is called when a browser has requested the `/update.ajax` page from the server. The function allocates space for the response. This is a multithreaded application, allowing multiple requests at any time, so, to save the stack space, space is allocated from the heap. The XML document is then built for the response:

1. Header
2. Address
3. Server type
4. Variables and tags
5. Footer

The content header length is set to the size of the XML document that was built (in bufcount); the reply is sent with the HTTP200 OK response and the data. The `WriteFinalOut` function is used because the `__http_write()` is a “defined” function with a return if there is an error. After the message is sent, the function deletes the temporary files reports the results.

The `commandRequest()` function is called when a browser has requested the `/command.ajax` page from the server. The syntax of the command is `command=x&val=x`, and the browser needs to parse it. The `http_find_argument()` function identifies what is on the right side of the equals sign. The helper function `atoi()` returns the integer for the command. The same procedure is used to get the `val`. From here is it just a matter of using a `switch` statement to figure out which command and set the appropriate hardware and create an XML fragment. At this point, XML response can be built. Space is allocated for the response and then the XML document is built for the response:

1. Header
2. Address
3. Server type
4. XML fragment that was created when the hardware was set
5. Footer

The content header length is set to the size of the XML document built (in bufcount). The reply is sent with the HTTP200 OK response and the data. The `WriteFinalOut` function is used because the `__http_write()` is a “defined” function with a return if there is an error. After the return occurs, the function deletes the temporary files reports the results.

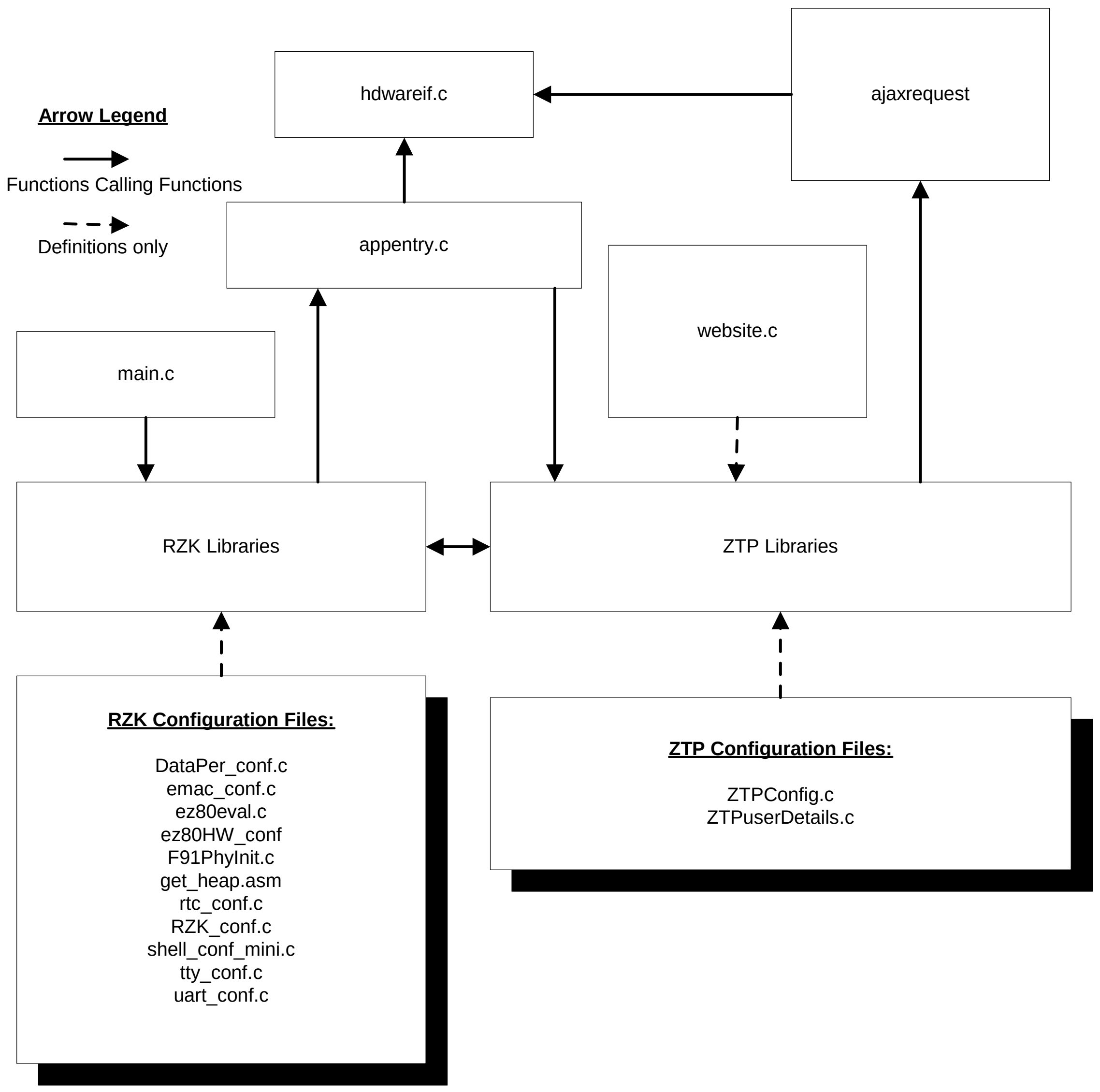


Figure 1. Server-Side Block Diagram

Client Side

As you can see, there is not much work for the web server to provide the information. With AJAX, that boring XML document is transformed from a bunch of strings to a graphical presentation and dynamic interactions. The majority of the work is on the client-side pages. These consist of the `main.htm` page, the `ajaxappnote.css` file, and the `ajaxappnote.js` file.

The dynamic requests are handled by a functionality built into most web browser called the

`XMLHttpRequest` object. To send a request, without reloading the entire page, you create a new `XMLHttpResponse()` object.
If you do not want to wait for a reply, set a function in the `onrreadystatechange` property of the object. Call the open method with the command (typically `GET` or `POST`), with the page to access, and a Boolean value that specifies if you are in a hurry or not. If you specify `true`, it will return immediately and call the function in the `onreadystatechange` property when it receives a response. If you specify `false`, it will block until a response is received then continue processing. The `readyState` property defines the state of the request. If the `readyState == 4`, it has received a response and the properties: status; responseXML; responseText are valid.

Note: *If the status does not equal 200, the text can contain the HTML error page from the server.*

The first section of `ajaxappnote.js` has the implementation for the request process. Because there can be multiple servers and multiple updates, pending requests are held in an array. When the requests are finished, they are removed from the array.

The `main.htm` file is a standard HTML document:
- The <HEAD> has some basic JavaScript functions.
- The <BODY> is what is actually displayed to the client.

Within the body, there are numerous `<div>` sections. These define the layout of how the page is actually presented to the user. Assigning IDs to the different divisions, tables, and items allows dynamic control of how the page looks at any time. Each of these elements can also be assigned event handlers to be executed on an event. For example, when a user clicks on an image, the event is `onclick`, and the handler is defined as a function. These, too, can be dynamically assigned and removed.

An HTML document follows what is called the Document Object Model (DOM). The DOM is what gives the ability to modify elements. The document is defined by nodes and childNodes, forming a tree type document. You can “walk the tree” by getting the nodes, nodelists, childNodes, and so on, or you can get the element by searching for its ID. You can explore the capabilities of the DOM by reading are references listed in the “References” section on page 14. Most current browsers support the DOM handling. You will see many DOM requests in the JavaScript source. Anything that starts with “document” is referencing the DOM.

The `ajaxappnote.js` file contains the dynamic page handling functions in JavaScript. JavaScript is an object-oriented procedure language that allows programmers to create new objects with specific functions and properties. Because these are objects, there can be multiple instances at any time. This application uses two types of objects:
- Navigation Object
  In this implementation, there is only one Navigation Object; its role is to control the “tabs” at the top of the page that specifies the servers that the page is monitoring.
- Server Objects
  The Server Objects are the servers that are being monitored.

Server Object

There are two different types of server objects:
- Generic (undefined type)
- F91DEVPlatform

Each of these has exactly the same prototype definitions, so that they can be used interchangeably. To add more server types, you just copy the prototype definitions, attach a new name, and modify the specifics. This allows you to monitor and control different types of servers. When a page is first loaded, it only has the IP address of the server delivering the page, but it has no idea of the type to display (this is information in the XML document from both the command and update requests), so it creates a generic server. The display functionality requests the server object to run the display. This allows the generic server to display its page (which is nothing), or the F91DEVPlatform to display its specific page. This means that it is the same display call for any type of page. All processing of the XML document is also done by the server. When the generic server gets the XML document from the request, it can get the server type and turn itself into the correct type of server object and display the information appropriately.

The Navigation Object (NavObject) controls the data area and tabs. When a user adds a server, a new page is created through the NavObject. When a user clicks on a tab to activate it, the NavObj is responsible for showing the correct data in the data window, making the selected tab the foreground tab, and setting the other tab as a background tab. It is also responsible to update the server status on the left status area.

The JavaScript section of `Main.htm` contains basic functionality needed for the page. This includes an `Init_Page()` function that centers the page, creates a NavObj, creates a generic server for the current page, and sends an UpdateRequest for the server. At this point, the page just handles events.

Cascading Style Sheets (CSSs)

The `ajaxappnote.css` file contains the specific "class" styles that allow you to dynamically change the look by modifying or changing the style "class". Example: The style class `.ledon` specifies a background image of `ledon.png`. The `.ledoff` class specifies a background image of `ledoff.png`. To reflect the change of turning on a LED, there is an `onclick` event handler assigned to the table cell that has a style class name associated to it of `ledon`. When the `onclick` event is triggered, the event handler requests the command (through the `XMLHttpRequest` object) and changes the class associated with that cell from `ledoff` to `ledon`. The item is updated and the background image `ledon.png` is now displayed. You can also assign style items dynamically by accessing the DOM element's style property and setting its value. This capability allows a boring XML file to be displayed in a graphical way.

Putting it All Together

Now that you have an understanding of the basic parts, here is the process of how the page works:

1. The browser sends a request to the web server for "\".
2. The web server sends `main.htm`.
3. The browser parses the file and requests the `ajaxappnote.js` file and the images.
4. The browser finishes loading and calls the `Init_Page()` function. The function initializes everything and issues an update request
5. The web server passes the update request to the `updateRequest()` function that returns an XML document that describes the current state of the hardware.
6. The browser passes this XML document to the server object, which parses and draws the data area by using DOM commands and style classes to present a graphic view of the status.

Setup

Use the following procedure to install the source code:

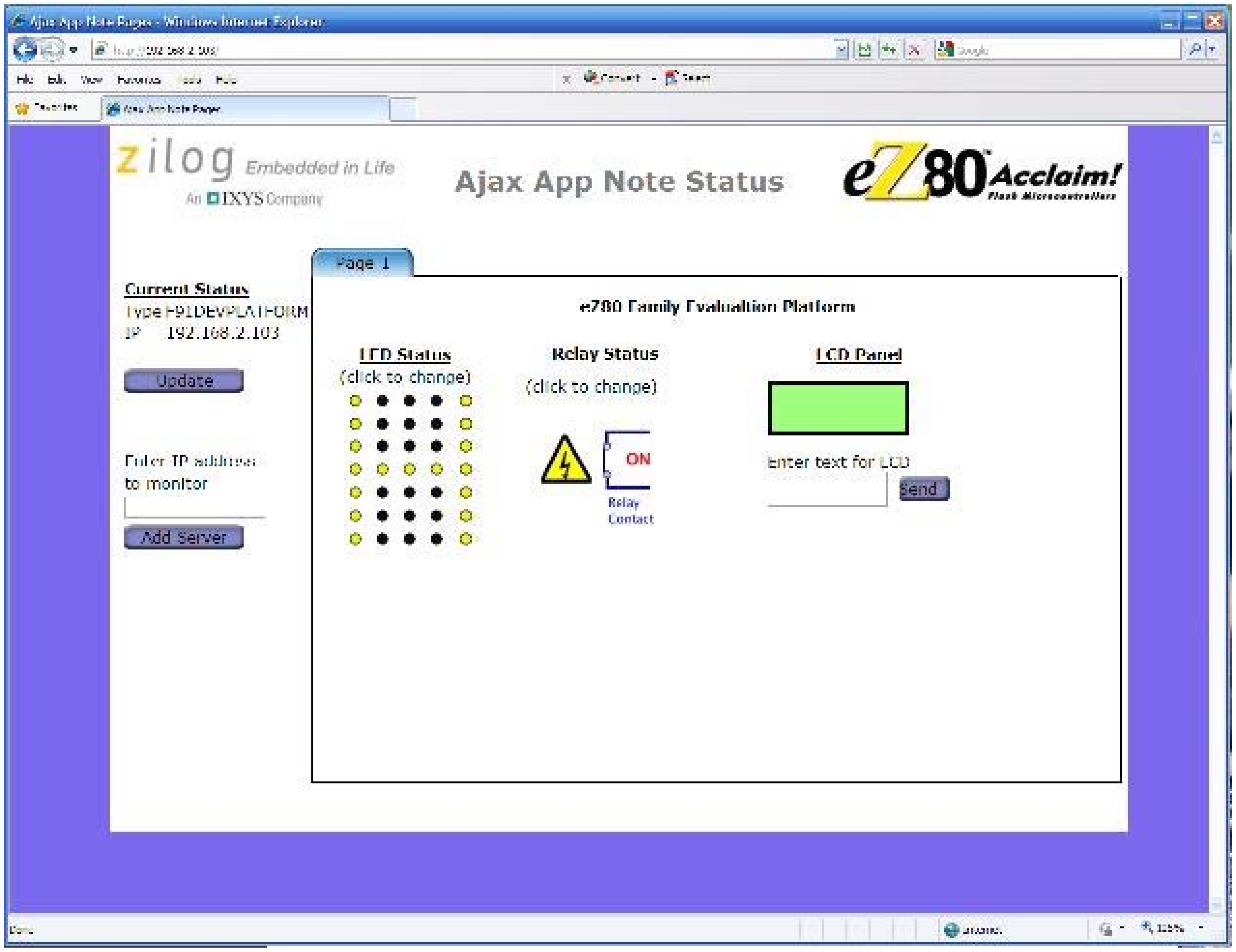
1. Launch the ZDS II program.
2. Select Tools>Options.
3. Click on the File Types.
4. Verify that the `.js` and `.css` files are associated with the web files project folder. If they are not included, click on the Associated File Types text box and add the following at the end:
   `.js;*.css`
5. Extract the zipped source code file into the following directory:

`Program Files\Zilog\ZDSII_eZ80Acclaim!_x.x.x\ZTP\ZTPx.x.x_lib\ZTP\SamplePrograms`

6. Open the project `F91_DevPlatform.zdsproj` file.
7. If you are not using DHCP, adjust the `ZTPConfig.c` file to reflect the Default IP address and Default Gateway; set `b_use_dhcp` to `FALSE`.
8. Rebuild all. There should be no warnings or errors. If there are, verify the following:
   - Verify that the files were extracted to the `ZTP\ZTPxxx_lib\ZTP\SamplePrograms` directory.
   - Verify that the file types for web files include `.css` and `.js` files.

Procedure

1. Connect a network cable
2. Connect a serial cable between the development platform and the com port on your PC.
3. Launch HyperTerminal and connect using 57600 Baud, 8 databits, no parity, 1 stop bits, and no flow control.
4. With the USB Smart Cable connected to the development platform, select Debug>Go. This will download the code and start the execution. The HyperTerminal shows the initialization of the network and displays the network address.
5. In a web browser connected to the same network that the server is attached to, type the following in the address box:
   http://xxx.xxx.xxx.xxx/
   (where the `xxx.xxx.xxx.xxx` is the IP address specified on the HyperTerminal output). The application is displayed on the browser as shown in the following figure.



Current Status
Type F91DEVPLATFORM
IP 192.168.2.103

Enter IP address to monitor

eZ80 Family Evaluation Platform

LED Status
(click to change)

Relay Status
(click to change)

LCD Panel

Enter text for LCD

6. Click on any of the 35 LEDs. The corresponding LEDs light up on the development board.
7. Make an H with the LEDs. Click on all the LEDS down one side, then the other, and a single line between them. An "H" is displayed on the LCD Matrix.
8. Click on the relay of the output. The HyperTerminal window displays the relay status.
9. Close the browser.
10. Launch a new instance of the browser.
11. Direct the address bar to the same address as before.
12. When the page loads, an "H" displays on the browser page, and the relay is set to its previous setting.
13. If you have another browser on a different computer, try it. You can access it with multiple machines. As you change something on one machine by clicking on the Update button, it will update that status of the current machine. If you have multiple units running the server software, you can type in the address to add a server. This will create a security violation, which you can allow in your browser preferences.

Summary

The eZ80Acclaim*Plus!*™ microcontroller can be used for building rich Internet-enabled products that are controlled over an Internet or intranet using web pages. This application note demonstrates how AJAX and

JavaScript can be used to develop a modern web page to control or monitor devices over the Internet. To exploit XML technologies and rich user experiences, explore the power of XML with transforms, schemas, and so on. Although this application is a simplistic implementation, the building blocks with AJAX and the eZ80Acclaim*Plus!*™ microcontroller allow you to develop rich user experiences that are limited only by your imagination!

Zilog has a technical forum on the Zilog.com website. This is an excellent place to suggest features or capabilities you would like to see in future application notes.

References

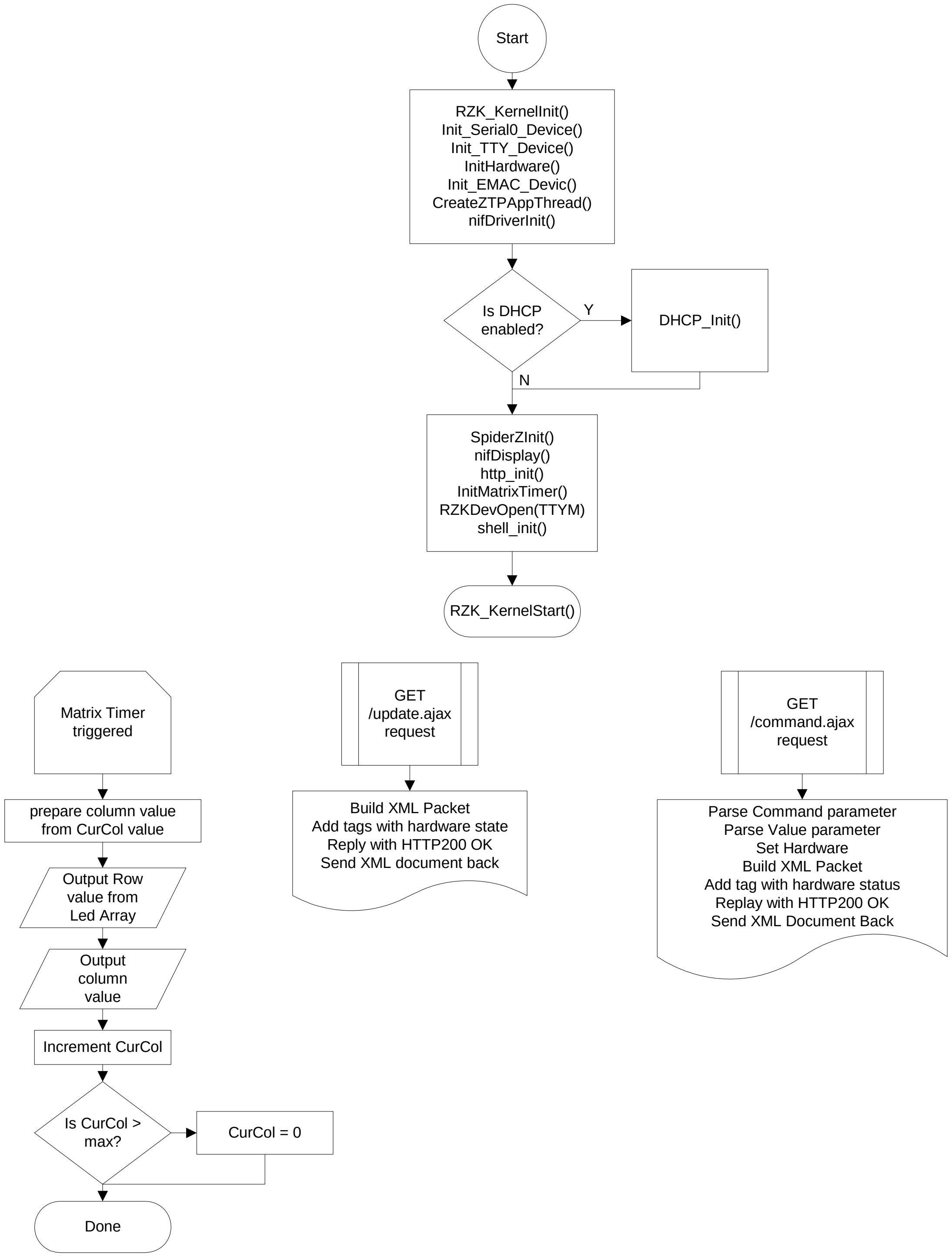
• RM0041	Zilog TCPIP Software Suite Programmer's Guide Reference Manual
• RM0040	Zilog TCPIP API Reference Manual
• RM0006	Zilog Real-Time Kernel Reference Manual
• UM0075	Zilog Real-Time Kernel Users Manual
• UM0144	Zilog Developer Studio II
• PS0272	eZ80F91 Ethernet Module Product Specification
• PS0192	eZ80F91 MCU Product Specification

Web Development References

- http://www.w3schools.com/
- http://www.tizag.com/
- http://www.microsoft.com/express/Web/
- http://www.w3.org/DOM/
- *HTML 4 for the World Wide Web*, (Forth or Fifth) edition by Elizabeth Castro
- *JavaScript: The Missing Manual* by David Sawyer McFarland
- *Ajax in 10 Minutes* by Phil Ballard
- *JavaScript & AJAX for Dummies* by Andy Harris
- *XML Developer's Guide* by Fabio Arciniegas

Appendix A—Flowcharts

The following is a flowchart of the server functions in the AJAX web server application.



Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

©2010 by Zilog, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

eZ80Acclaim*Plus!* is a trademark and eZ80 and eZ80Acclaim! are registered trademarks of Zilog, Inc. All other product or service names are the property of their respective owners.