



# **TCP/IP Networking: Web-Based Status Monitoring**

## **Microchip TCP/IP Stack HTTP2 Module**

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Web-Based Status Monitoring

Slide 1

Welcome to the first in a series of Microchip's web seminars on building embedded web applications. We'll be discussing the HTTP2 web server module included with Microchip's free TCP/IP Stack, and more specifically, how you can use Dynamic Variables to present data from your system to a web browser.

My name is Elliott Wood, and I'm an Applications Engineer at Microchip's headquarters in Chandler, AZ. I am one of the main developers of Microchip's TCP/IP Stack, and I'll be explaining how to build a simple web-based status monitoring application for an embedded product.

## Project Overview

- **Web Enabled Vending Machine**
- **Online Features:**
  - Stock check
- **Scope:**
  - Basic familiarity with TCP/IP Stack
  - Vending machine and web pages are written
  - Just need to add the HTTP2 callbacks

[www.microchip.com/tcpip](http://www.microchip.com/tcpip)

Suppose you are tasked with maintaining several vending machines. Your job is to make sure that the machines stayed stocked with product. Visiting each machine every day to check on its stock could be time consuming. Wouldn't it be easier if you could check each machine's status from the PC at your desk?

This seminar will take a simple vending machine application and add web-based monitoring capabilities to the device. Over the next 25 minutes or so, we will build an online interface to our vending machine, allowing us to check its stock from the Internet.

If you're just viewing out of curiosity or to see what's possible, all you need is a general understanding of embedded design. If you want to follow along with the examples, you will need to have downloaded and installed the full TCP/IP Stack distribution from the web page shown on the slide. We will be using the TCPIP WebVend App project. You should have also at least completed the Getting Started section in the TCPIP User's Guide, and know how to use the MPFS2 utility.

For this seminar, we will assume that the actual vending machine application has already been written. We will also assume that a web developer has designed a simple web page for us to use. Our job will be to add the required hooks between the web pages and Microchip's HTTP2 web server module. In essence, we will provide the "glue" between the embedded application and the HTML web pages. The source code for this basic vending machine is all included in the TCPIP WebVend App project installed with the TCPIP Stack.

## Agenda

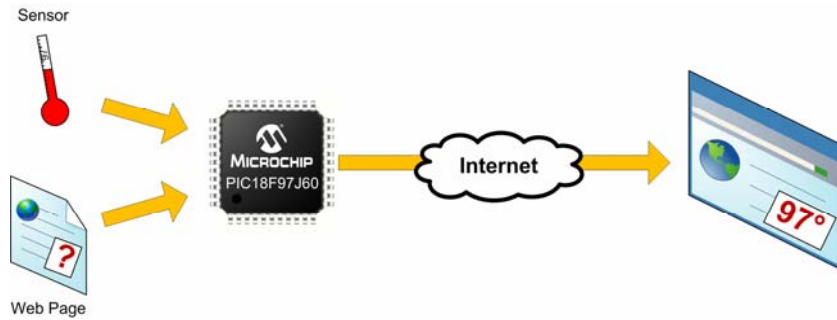
- **Simple Dynamic Variables**
- **Dynamic Variable Parameters**
- **Outputting HTML**
- **Wrap-up: Working Demo**

In this seminar, we will describe three main uses of dynamic variables. First, we'll cover the basics and explain how to dynamically insert text strings into a web page. Next, we will expand upon this use and show how to pass parameters, allowing you to write less code. Finally, we will demonstrate the use of dynamic variables to output HTML, which provides the power to control design, layout, and graphical elements on a page. A working demo will be shown when we've completed all three topics.

# Basic Dynamic Variables

Let's get started. The first task is to learn the basics of dynamic variables.

## Dynamic Variables: Overview



- **Combine system data into web pages**
- **Present completed page to user's browser**


Dynamic variables allow the web server module to take data from your system, such as the value from a sensor or data in memory, and combine it into a template web page. The completed web page is then transmitted through a network or the Internet, and the system data is displayed on the user's screen. In the simplified example on the slide, the microcontroller inserts the value from the thermometer into a web page and transmits that page to the user's browser.

## Dynamic Variables: Overview

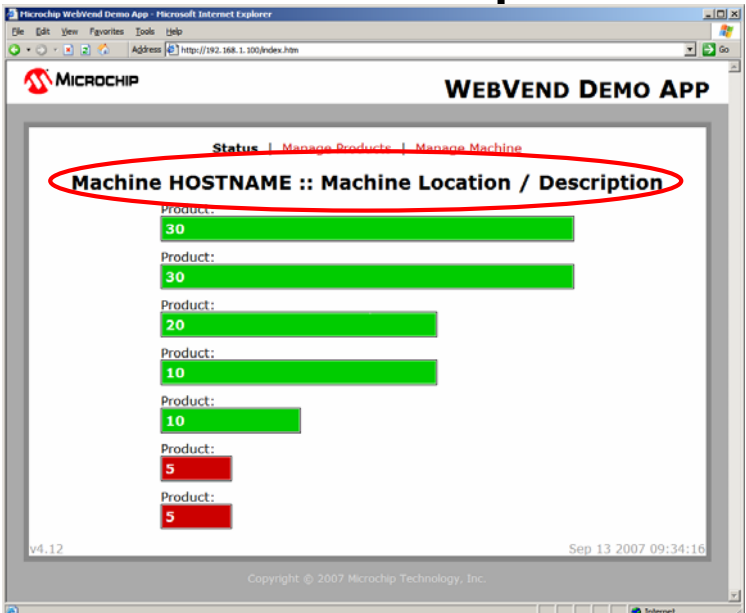
- Display system data on web pages
- Indicated in HTML by names in ~ ~ pairs
- Invoke a callback function
- To create a variable called `foo`:
  - Insert `~foo~` in your web pages
  - Implement `HTTPPrint_foo()` in *CustomHTTPApp.c*

Now that you know what a dynamic variable does, let's look at how to make one. Dynamic variables are created by enclosing any name inside a pair of tilde (~) characters and placing that variable in your web pages' HTML code. When the HTTP2 web server module encounters this variable, it will execute a callback function. This callback is implemented in your application's C code, and controls what is transmitted to the browser and ultimately displayed on screen.

For example: to create a variable called `foo`, two pieces are necessary. You must first insert the variable `~foo~`, enclosed in tilde characters, somewhere in our web pages. Second, you must implement the callback function `HTTPPrint_foo()` in your project's `CustomHTTPApp.c` source code. The `HTTPPrint_` portion indicates a dynamic variable callback, but the name "foo" will change depending on the name of your variable.



# Basic Example



Here's a simple example. If we take a brief look at our web page, we will see that there is a space at the top reserved for the device's name and a description line. Right now they are hard-coded in, but we'd like those elements to display our host name and description string.

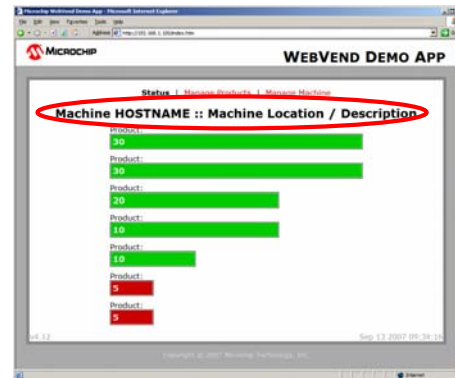
## Basic Example

- **GOALS:**

- Display host name on main web page
- Display location and description on screen

- **SOLUTION:**

- Replace host name text with `~hostname~`
- Replace description with `~machineDesc~`
- Implement appropriate callbacks



To accomplish this, we will add two dynamic variables. The first will output the host name, and the second will write the machine description or location string. We must then implement the appropriate callback functions to display these values.



## Basic Example: index.htm

### Before:

```
<div id="location">  
    Machine HOSTNAME ::  
    Machine Location / Description  
</div>
```

### After:

```
<div id="location">  
    Machine ~hostname~ :: ~machineDesc~  
</div>
```

Let's start with the HTML file, located in the WebPages2 folder within the project. If we open up index.htm, we'll see some code towards the top where the host name is hard-coded. We will replace this name with a dynamic variable. Let's call it "hostname", all one word, and place it within a pair of tilde (~) characters. We will do the same for the machine description string, as shown. When we save this HTML file, our variables will have been added to the web page.

## Dynamic Variables: Callbacks

- **Write directly to the socket**
- **Guaranteed 16 bytes free**
  - **Must manage output for longer strings**
  - **Use `curHTTP.callbackPos`**
- **Add callbacks in *CustomHTTPApp.c***
  - `HTTPPrint_hostname()`
  - `HTTPPrint_machineDesc()`

The second part of a dynamic variable is the callback function. When a dynamic variable is encountered, a callback function is executed. This function will generally write some data to the socket.

For technical reasons outside the scope of this presentation, the HTTP server operates in a buffer of a fixed size. When a callback is invoked, the server guarantees at least 16 bytes are free in the buffer. If a dynamic variable is only to write small amounts of data, it can write immediately and return. If more than 16 bytes need to be written, the callback must first check how much space is available, and use the variable `curHTTP.callbackPos(ition)` to indicate whether or not it is finished. We will demonstrate this in the next few slides.

To hook in the two dynamic variables we just created, we will need to add two callback functions to the file `CustomHTTPApp.c`. They must be named according to the variable names chosen when the dynamic variables were inserted into the HTML.

## Example: CustomHTTPApp.c

```
void HTTPPrint_hostname(void)
{
    TCPPutString(sktHTTP,
        AppConfig.NetBIOSName);
}
```

The first callback function to be added is HTTPPrint\_hostname(). This function outputs the host name of the device. This host name is the NetBIOS name currently being used, and is stored in the AppConfig structure. Since it is defined to be 15 characters, we can write it directly without worrying about buffer space. Simply call an appropriate TCPPut() function and write the data to the HTTP socket. In this case, I've chosen to use TCPPutString. Refer to the TCP API to learn what functions are available.

## Example: CustomHTTPApp.c

```
void HTTPPrint_machineDesc(void)
{
    if(strlen((char*)machineDesc) >
        TCPIsPutReady(sktHTTP))
    { // Not enough space
        curHTTP.callbackPos = 0x01;
        return;
    }

    TCPPutString(sktHTTP, machineDesc);
    curHTTP.callbackPos = 0x00;
}
```

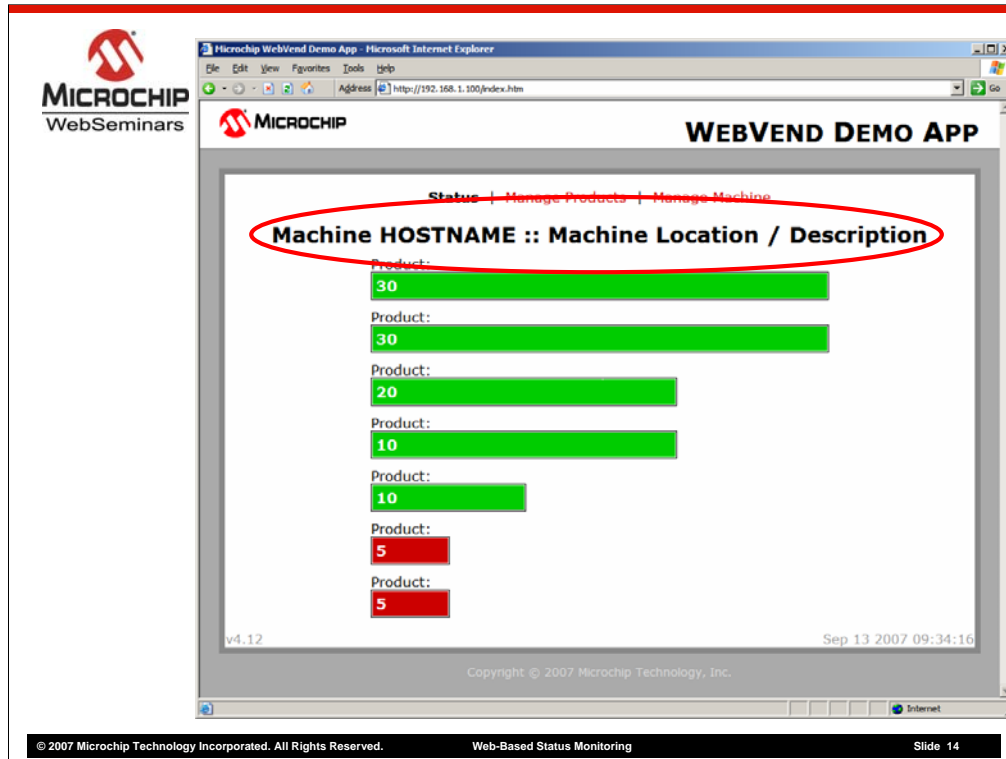
The machineDesc(ription) callback is slightly more complex. A string, “machineDesc” is defined as a global variable for our application. It is 32 bytes in length, so it is longer than our 16 byte limit and therefore we must make sure enough space is available before printing. The first if-statement performs this check. If the length of this description string is longer than the amount of space free in the buffer, we must wait and try again later. To accomplish this, we set the callback position to a non-zero value.

If enough space is available, we will write the string to the buffer and return. Before this happens, we must set the callback position to zero. This clears any flag that may have previously been set if we had to wait for space. The value of zero indicates to the HTTP server that our callback function is finished with all its output.

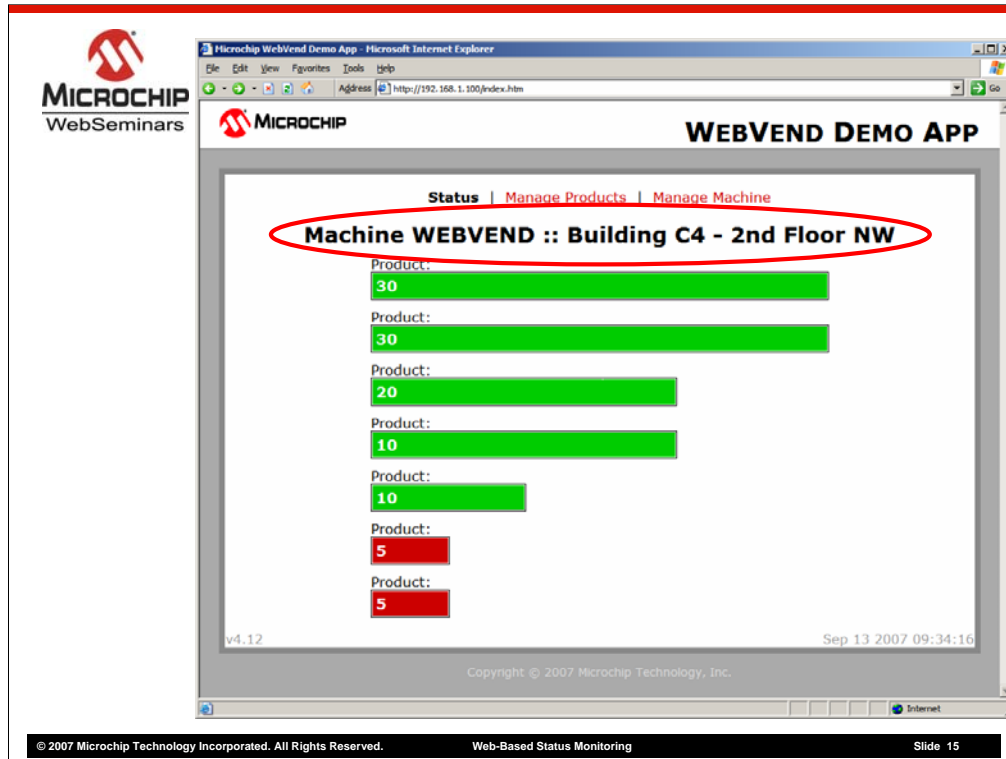
For this example we know that at some point there will be at least 32 free bytes, so we can just return and wait until enough space is available. To write arbitrarily large amounts of data, you should use determine how much space is available, write that amount, and track your output state in the callback position variable. As a practical maximum, you should support partial writes like this for any output that may be longer than about 40 bytes.

# Demonstration

Let's take a look at what we've just changed.



Our original webpage showed just some static text at the top of the page. If I reload the page with the changes we just made, we'll see that information become populated by the board.



The default host name for this application is WEBVEND, and the description string defaults to the location of my office here at the Chandler facility.

## Basic Example Review

- **Allow arbitrary output**
- **Two components:**
  - ~varName~ in web page HTML code
  - HTTPPrint\_myVar( ) in *CustomHTTPApp.c*
- **Must manage output if > 16 bytes**
  - Use `curHTTP.callbackPos`

So far, we've seen how dynamic variables provide the capability to output data. Each dynamic variable has two components. The first is the variable declaration in the web page. The second is an associated callback function in *CustomHTTPApp.c*. And remember that outputs longer than 16 bytes must be managed by your application.



# Dynamic Variable Parameters

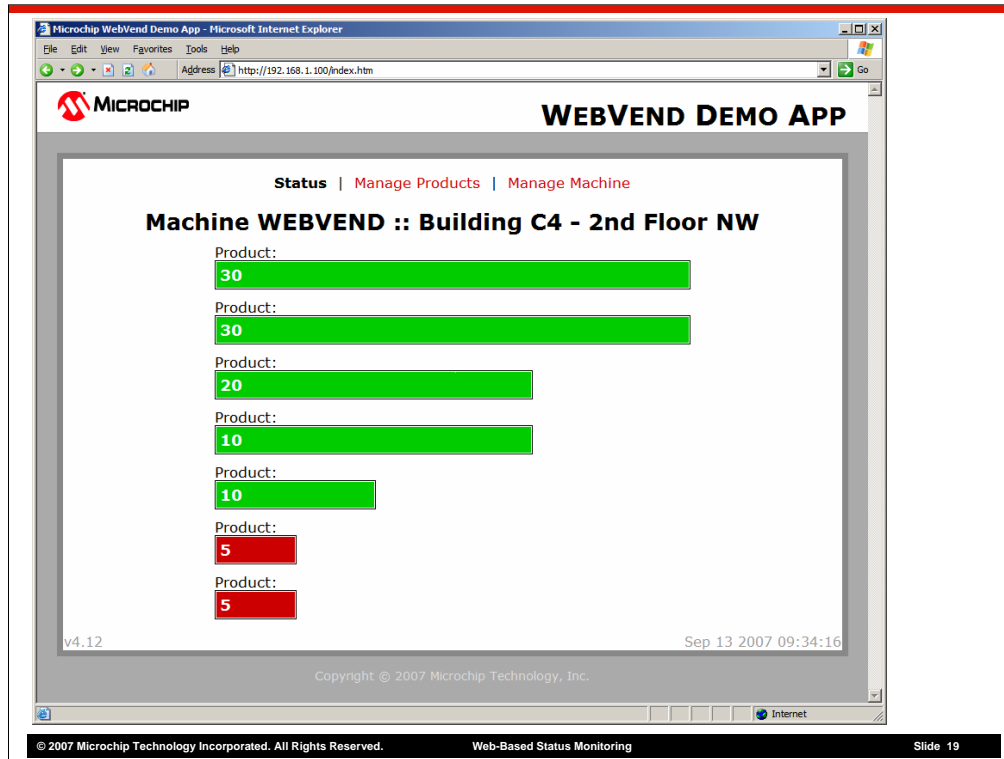
Let's move on to some more advanced uses.

## Parameters: Overview

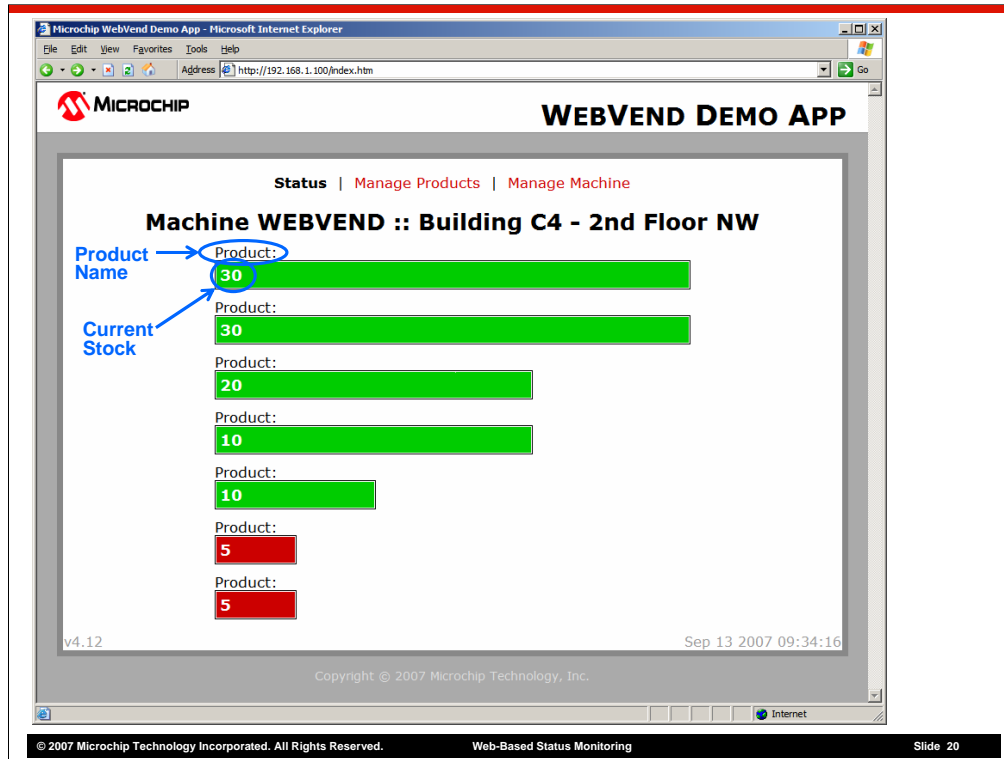
- Can pass parameters to callbacks
- Extremely useful for arrays
- Add parameters in ( ) after varname
  - `~myVector(3)~` , `~myArray(7,25)~`
- Callback functions get WORD parameters
  - `HTTPPrint_myVector(WORD)`
  - `HTTPPrint_myArray(WORD, WORD)`

Parameters can also be passed through dynamic variables. These parameters are entered in the web page and are ultimately passed to the callback function. This method can be extremely useful for multiple outputs of similar data, such as arrays.

To add a parameter to a dynamic variable, enclose that value in parenthesis after the name. Multiple parameters are supported in this fashion, but every call to a variable of that name must have the same number of parameters. Each parameter will be passed to the associated callback function as a 16-bit word value.



Let's try this out with an example. Look again at our vending machine demonstration. There are bar graphs for seven different products.



We want to display the product name above each bar, and display the current stock remaining inside of the bar.

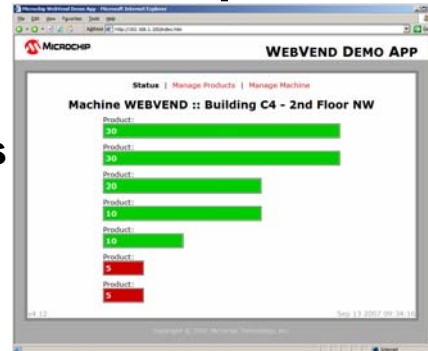
## Parameters: Example

- **GOALS:**

- Display product names
- Update numeric stock display

- **SOLUTION:**

- Replace product names with `~name(#)~`
- Replace stock display with `~stock(#)~`
- Implement `HTTPPrint_name(WORD)`
- Implement `HTTPPrint_stock(WORD)`



Rather than writing seven different callback functions for each product name, plus another seven for the stock levels, we can write just two callback functions. One will handle all the product names, and the other will handle the stock. The product we're currently interested in will be passed to the callback function as a parameter.

## Parameters: index.htm

### Before:

```
<div class="productname">Product:</div>
<div class="bar-out" style="width: 17em">
  <div class="bar-in-ok">stk</div>
</div>
```

### After:

```
<div class="productname">~name(0)~:</div>
<div class="bar-out" style="width: 17em">
  <div class="bar-in-ok">~stock(0)~</div>
</div>
```

Looking again at index.htm, we see seven blocks of code. Each has a label for the product, and a place where the stock should be displayed. We will replace each of the “Product:” labels with a dynamic variable indicating the product name to be displayed. Add a number inside of parenthesis to the end of each variable. This parameter will indicate which product name to print at that location. Repeat this procedure for each of the stock displays.

Our web pages are now ready to display product names and stock levels.

## Parameters: Callbacks

- Work exactly as before
- Have an additional parameter list
- Add callbacks in *CustomHTTPApp.c*
  - HTTPPrint\_name(WORD)
  - HTTPPrint\_stock(WORD)

As in the prior example, we need to implement two callbacks in CustomHTTPApp.c. These callback functions work exactly as before, except that they now receive a single 16-bit word parameter. The callbacks take on the name of the dynamic variables we just added into the HTML code, “name” and “stock”.

## Parameters: CustomHTTPApp.c

```
void HTTPPrint_name(WORD item)
{
    TCPPutString(sktHTTP, items[item].name);
}

void HTTPPrint_stock(WORD item)
{
    BYTE digits[4];
    uitoa(items[item].stock, digits);
    TCPPutString(sktHTTP, digits);
}
```

The first callback handles printing of the names. The parameter “item” indicates which name to print. Since the name strings are 10 bytes or less, we just write them to the socket and return.

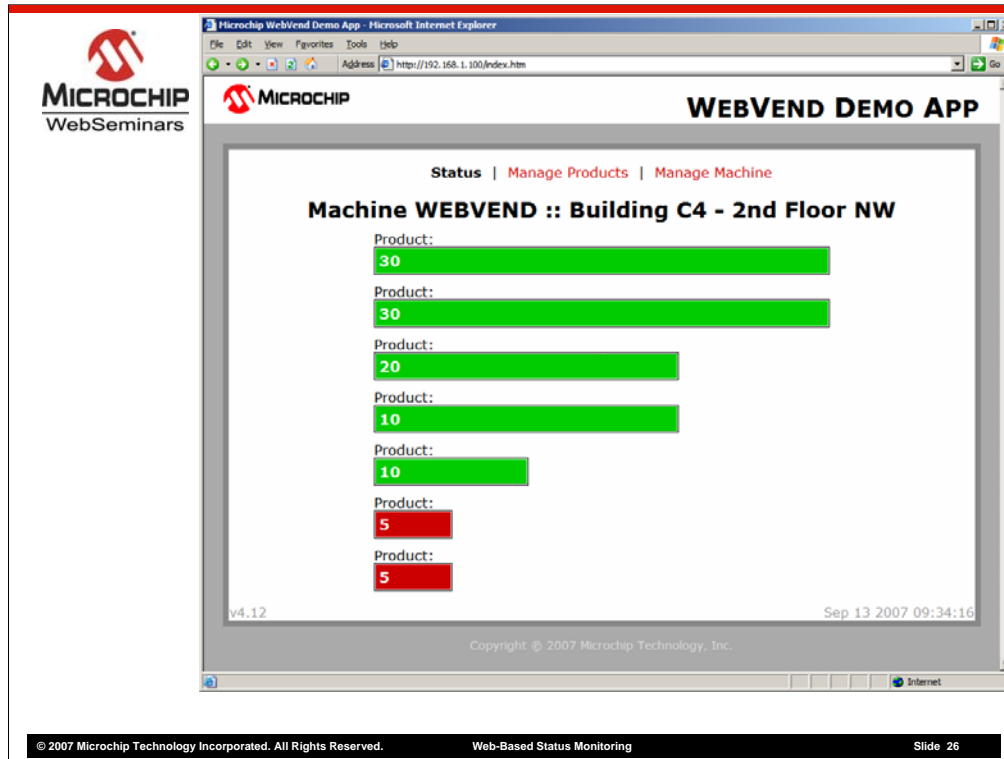
The second callback is only slightly more complicated. Since we would like to display a number, we must convert that number using the unsigned integer to alpha function, then print that string. The “item” parameter is again passed to this function, and we will use that parameter to determine which product stock should be printed.

These two callback functions will take care of all cases for printing names and stock levels.

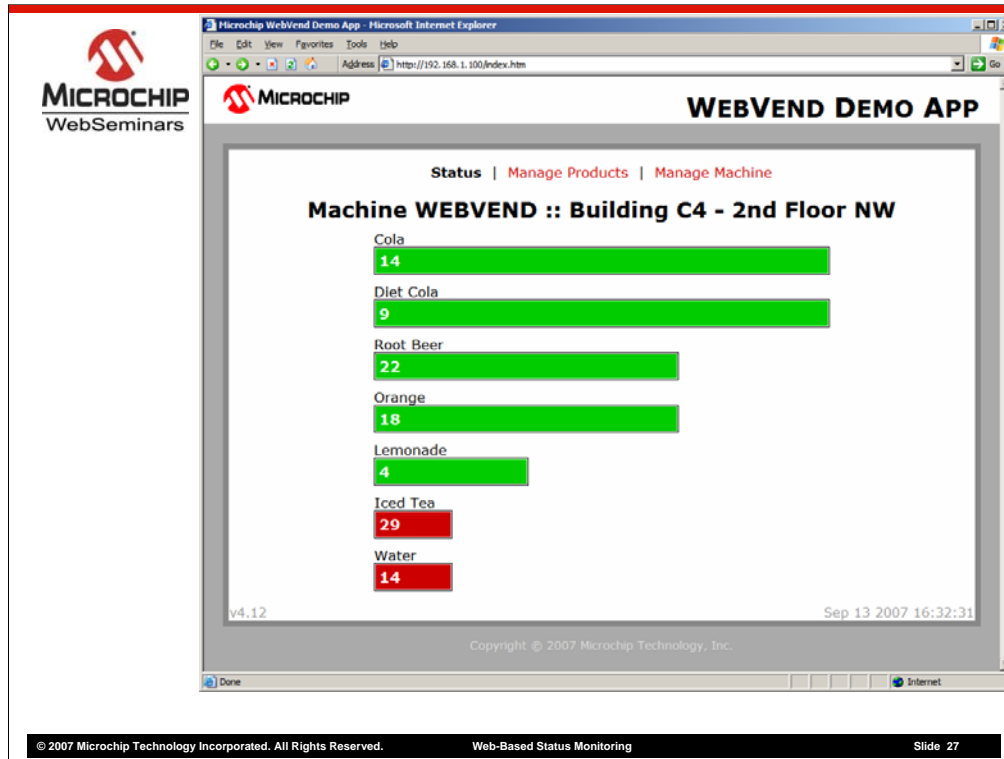


# Demonstration

Let's check on those changes we just made.



Our original web page had statically listed products and stock levels.



After making this change, we see that the names above each of the bars have been populated with data from our machine, and the stock number in each bar has also changed. We'd still like to modify a few more elements, but things are looking good. Let's continue.

## Parameter Example Review

- **Callbacks can receive parameters**
- **Enables less code in CustomHTTPApp.c**
- **All parameters are WORD values**
- **Multiple parameters are supported**
- **Two components:**
  - `~varName(0)~`
  - `HTTPPrint_myVar(WORD)`

In the past several slides, we saw how parameters can be passed to dynamic variables. The callback functions receive these parameters, which enables us to write less code in CustomHTTPApp.c. The same two components are required for these variables. The only difference is the parameters, indicated in parenthesis after the variable name, and passed to the callback function as 16-bit WORD values.

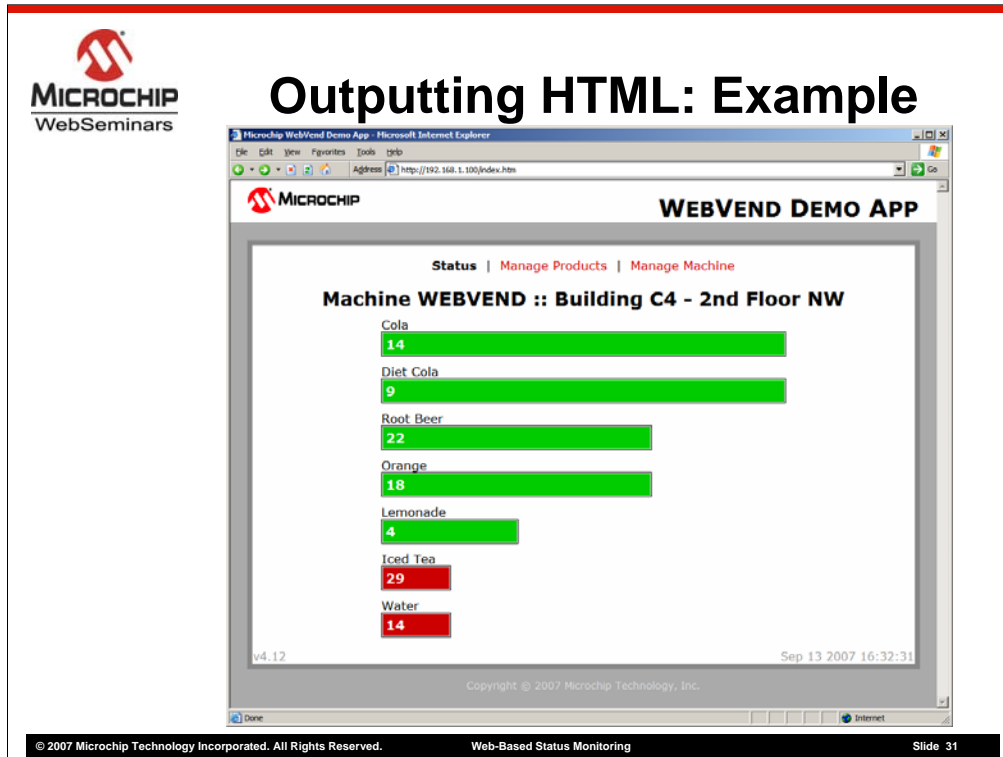
# Outputting HTML

Our last topic covers outputting HTML.

## Outputting HTML: Overview

- **Dynamic variables can send HTML**
- **Can be used to control display elements**
- **Works for CSS, JavaScript, etc.**
- **No special changes required**

So far, we've used dynamic variables to display text on our web pages. Yet dynamic variables are much more powerful than that. They can be used to output text anywhere in the HTML code. This allows them to control display elements, images, CSS, JavaScript, and anything else that can be modified within HTML. No special changes are required for this functionality – the variables are just inserted in a place where they will modify the HTML code rather than the text display.



Our vending machine monitor has two logical places for outputting HTML rather than displayable text. The first, and most obvious, is the length of the bar. We want that to grow and shrink with the stock, and we can control that with HTML. The second is more subtle, but a nice feature nonetheless. The shorter bars in the example our web developer provided are red rather than green. We'd like the colors to change automatically when stock gets low as well.

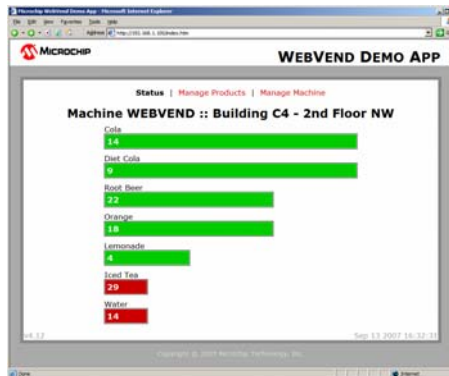
## Outputting HTML: Example

- **GOAL:**

- Vary length of bar with stock
- Change bar color when stock is low

- **SOLUTION:**

- Output stock as bar length
- Have a dynamic variable control color
- *vending.css* defines “bar-in-ok / low”



The solution, again, uses dynamic variables. It will make more sense when we look at the code, so we'll describe it in detail on the next slide.



## Outputting HTML: index.htm

### Before:

```
<div class="productname">~name(0)~</div>
<div class="bar-out" style="width: 17em">
  <div class="bar-in-ok">~stock(0)~</div>
</div>
```

### After:

```
<div class="productname">~name(0)~</div>
<div class="bar-out" style="width: ~stock(0)~em">
  <div class="bar-in-~status(0)~">~stock(0)~</div>
</div>
```

We see in the second line of each block there is a definition for the width of the bar element. It's measured in EMs, which is a standard print measurement. If we just output the stock in place of that fixed number, our application will be able to control the width property of each bar. We've already got a dynamic variable written for the stock of each product, so we'll just use that again.

The bar color requires closer inspection. By comparing two different bars, it looks like the class of the inner bar is being modified to be either "bar-in-ok" or "bar-in-low", depending on the stock level. That class is defined in the site's CSS style sheet, but its exact implementation doesn't really affect our application. For now, we can just replace this "ok" or "low" suffix with a dynamic variable called "status". We will again pass the product number as a parameter, and will determine whether to write "ok" or "low" based on the current stock of that product.

That's all for the modifications in the HTML. Now we must tie in the callbacks.

## HTML: CustomHTTPApp.c

```
void HTTPPrint_status(WORD item)
{
    if(items[item].stock < 10)
        TCPPutROMString(sktHTTP,
                        (ROM BYTE*)"low");
    else
        TCPPutROMString(sktHTTP,
                        (ROM BYTE*)"ok");
}
```

The callback for “stock” was written in a prior example, so we are just using it as-is. No problem there. We do need to implement the “status” callback however.

The status we removed was either “ok” or “low”. We can output the correct class suffix again by checking the stock level for the specified item. If the stock is less than 10, we will output “low”. Otherwise, “ok” will be written. That’s all we need to do. The CSS takes care of all the necessary color and style changes, and our web developer already wrote that portion for us.

## Outputting HTML Review

- **Dynamic variables can output HTML**
- **Allows control of layout, display, etc**
- **Works for CSS, JavaScript, etc**
- **No special changes required**

We've just shown how to use dynamic variables to control HTML output. This provides the capability to control graphical elements, and even place values within CSS or JavaScript as necessary. No special changes were required to do any of this; we simply placed the dynamic variable within the HTML or CSS we wanted to modify.

## Working Demonstration

Now that we've added all the dynamic variables, let's take a look at this page. If we program our board and load the web page, we will see default products and stocks shown on our screen.

If I insert several quarters (by pressing BUTTON3, the right most one), then vend a few drinks by pressing BUTTON2, the machine will track this change in its memory. I'll buy a few Colas and a couple of Diets right now.

When I click refresh in the browser, we see the stock numbers change and the bars shrink. If I buy a few more Colas right now, there will be fewer than ten remaining. When I refresh the page again, the bar will turn red, indicating that it's about time to refill.

It looks like our vending machine project is a success!

## Dynamic Variables: Summary

- Allow output of arbitrary data
- Can output text, data, or HTML
- User-defined output functions
- Can pass parameters
- Two components:
  - ~varName~ in web page HTML code
  - HTTPPrint\_myVar( ) in *CustomHTTPApp.c*

Today's seminar explained how add web-based monitoring to a simple embedded application. We used dynamic variables to output system data to a web browser. These variables can write either text or HTML code, and will invoke custom callback functions. We can also pass parameters to these callback functions, which simplifies output for arrays.

## Conclusion

- **Use dynamic variables to provide web-based output from your application**
- **Web status monitoring application**
- **First of several TCP/IP seminars**
  - **Will cover Form Processing, Authentication, E-mail, and more**
- **Please provide feedback!**

The tools presented should provide everything needed to build a simple web status monitoring application.

As mentioned at the beginning, this is the first of several TCP/IP seminars. If you'd like to learn how to update the system through the web, or have it automatically e-mail you when stock runs low, check for future seminars. We will soon cover other topics involving the HTTP2 module such as processing web forms, and basic password authentication. We will also explain how to automatically send yourself an e-mail when stock runs low, and much more.

Before you go, please remember to provide feedback using the link on this seminar's web page. Comments both good and bad will be appreciated so we can better tailor future seminars to your needs.



## For More Information

- **[www.microchip.com/tcpip](http://www.microchip.com/tcpip)**
- **TCPIP Stack User's Guide**
  - Installed with the TCP/IP Stack
  - Accessible via Start menu
- **TCPIP Demo App**
  - *WebPages2*
  - *CustomHTTPApp.c*

© 2007 Microchip Technology Incorporated. All Rights Reserved.

Web-Based Status Monitoring

Slide 39

For more information on the topics presented here, please refer to the TCPIP Stack User's Guide. This file is installed with the TCP/IP Stack, and can be accessed via the Start menu.

More examples and labs are available within the TCPIP Demo App that is installed with the stack. The demo application has its own WebPages2 folder and CustomHTTPApp.c source file that explain how everything works in more detail.

Thanks for watching!

# Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KeeLoq, KeeLoq logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.