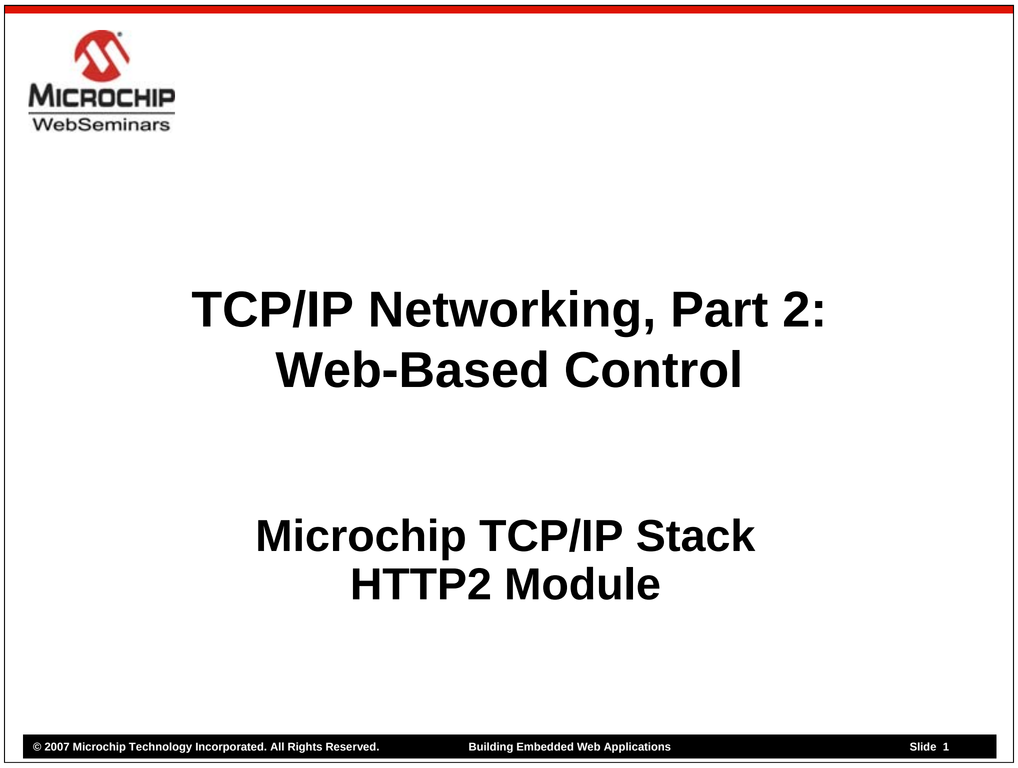
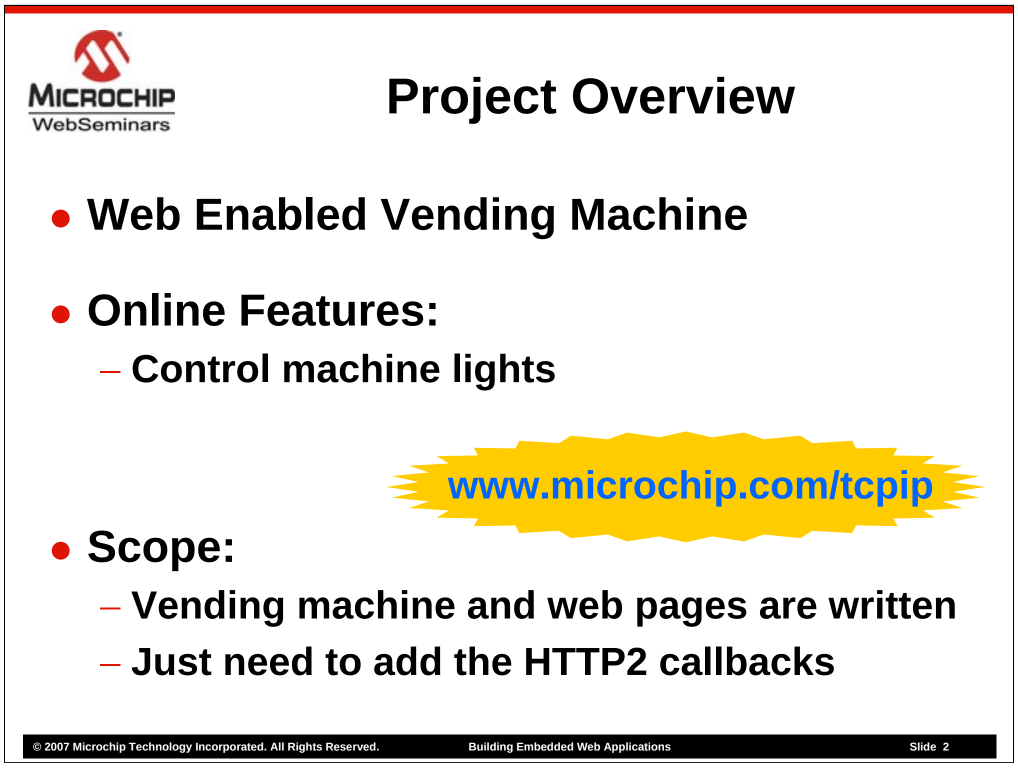


TCP/IP Networking, Part 2: Web-Based Control

Microchip TCP/IP Stack HTTP2 Module

Welcome to the next web seminar in Microchip's series on 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 web-based control to accept input to your embedded application.

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 control system for an embedded product.


Project Overview

- Web Enabled Vending Machine
- Online Features:
  - Control machine lights
- Scope:
  - Vending machine and web pages are written
  - Just need to add the HTTP2 callbacks

www.microchip.com/tcpip

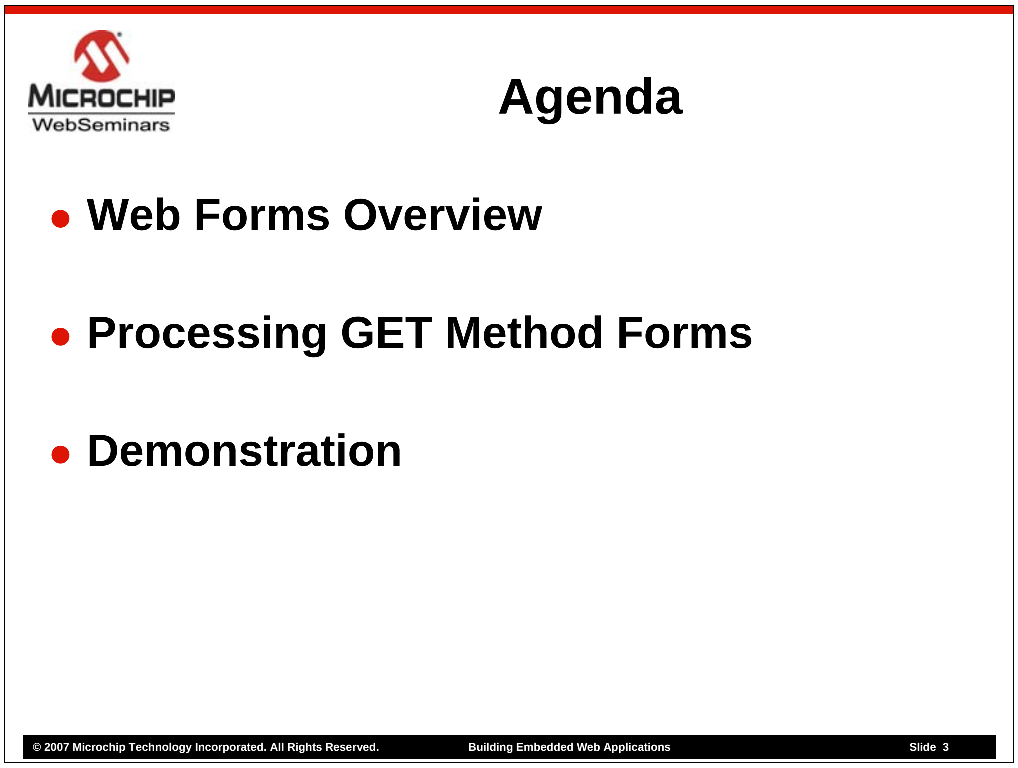
© 2007 Microchip Technology Incorporated. All Rights Reserved. Building Embedded Web Applications Slide 2

In the previous seminar, we added web-based monitoring capabilities to a simple vending machine. If you haven't watched that seminar, you may want to do so before continuing. This presentation will assume a basic understanding of the HTTP2 web server module, and how to use dynamic variables.

Today we will extend that application. When we're done, you'll be able to control our vending machine's lights online from a simple form.

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. 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. We will be using the TCPIP WebVend App project that is installed with the TCP/IP Stack.

Just as in the last seminar, we will assume that a web developer has already designed a simple web page for us. We will start with the working vending machine application from the last session, plus this additional web page. Our task will be to add the required hooks between the form in this new web page, and our embedded application. If you're following along, you should copy the base "WebPages2" folder and "CustomHTTPApp.c" source file from the "02 – Control with GET" directory inside the "Labs" folder.

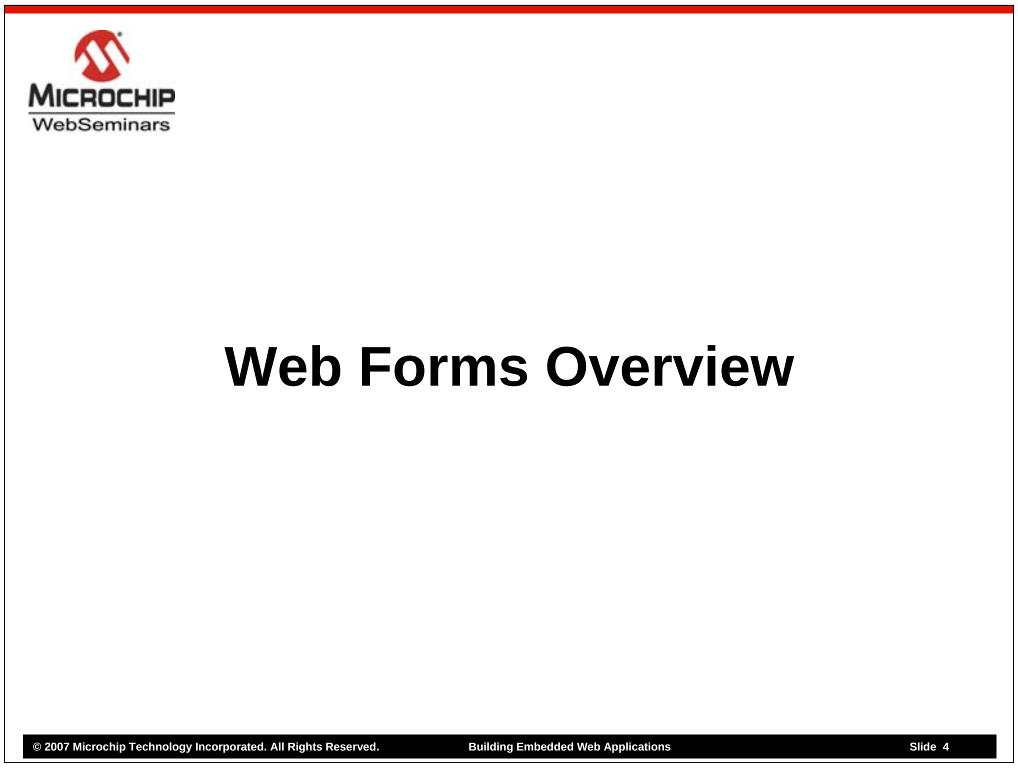


Agenda

- Web Forms Overview
- Processing GET Method Forms
- Demonstration

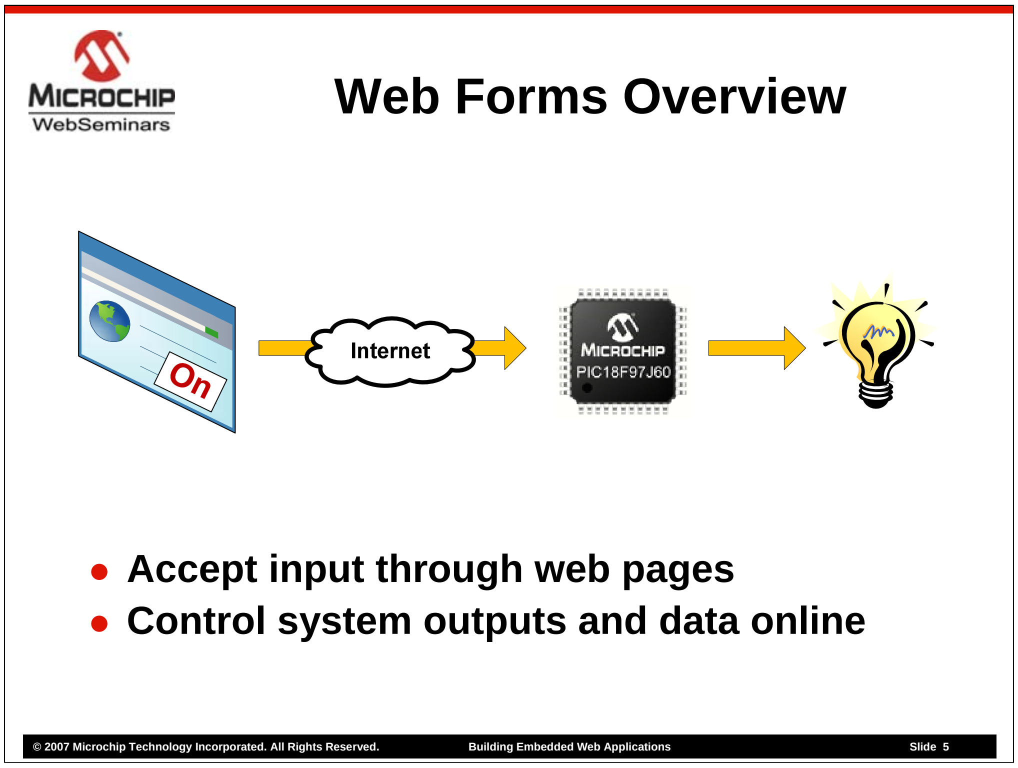
We will start with a very brief overview of web forms, and how to read them in HTML code. Since most people are unfamiliar with the difference between the GET and POST methods, this will also include a brief comparison of the two. We will then describe how to process data submitted from web forms using the GET method. At the end, we'll see a demonstration of this capability.

Let's get started.

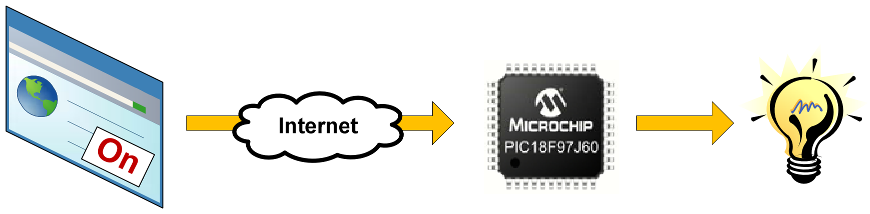


Web Forms Overview

[no narrative]

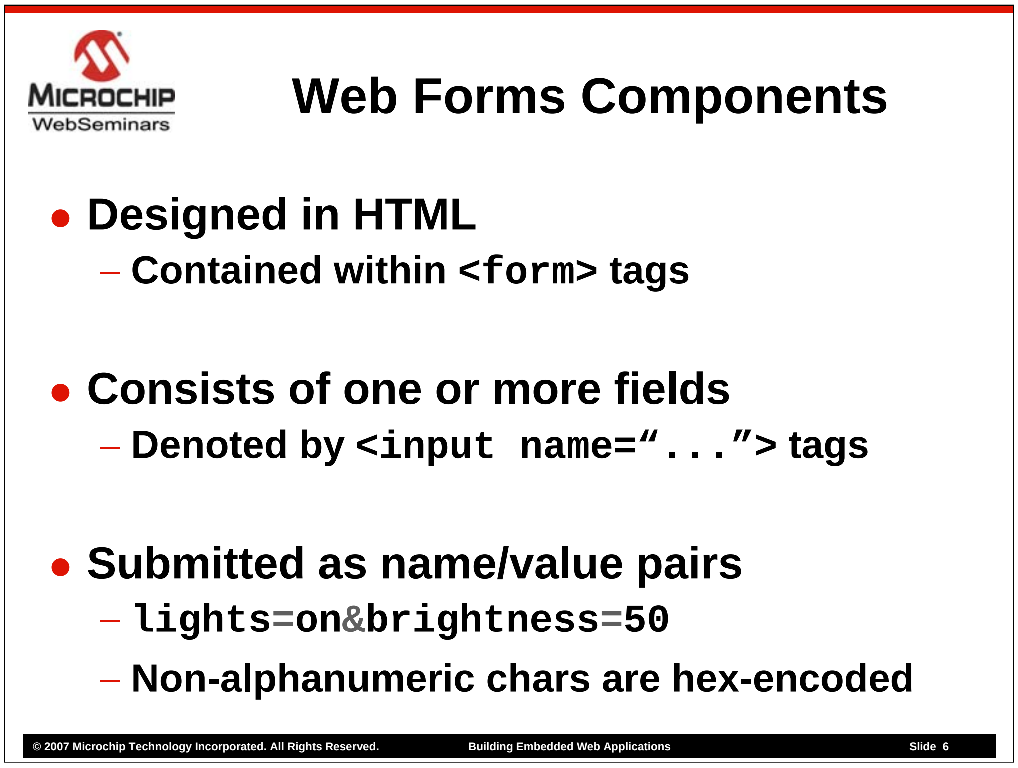


Web Forms Overview



- Accept input through web pages
- Control system outputs and data online

Web forms allow the web server module to accept data from users through a network. This data can then be used to control system memory or outputs. In the simplified example on the slide, the microcontroller accepts the command of "On" from the web page and responds by turning the light bulb on.



Web Forms Components

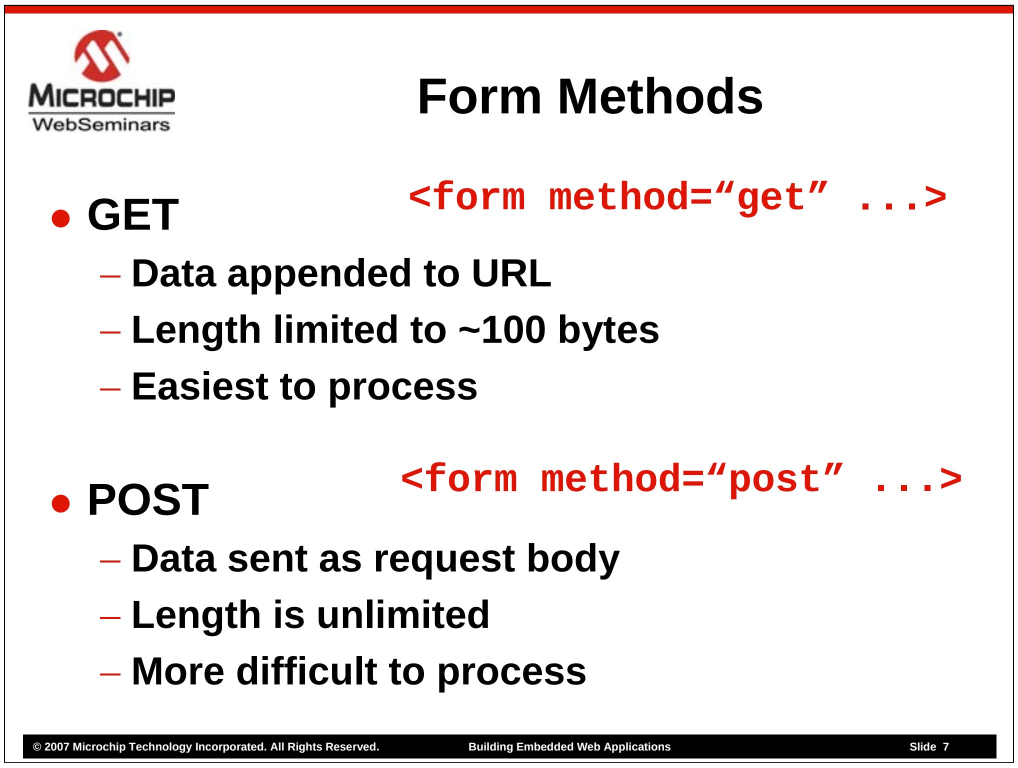
- Designed in HTML
  - Contained within `<form>` tags
- Consists of one or more fields
  - Denoted by `<input name="...">` tags
- Submitted as name/value pairs
  - `lights=on&brightness=50`
  - Non-alphanumeric chars are hex-encoded

© 2007 Microchip Technology Incorporated. All Rights Reserved. Building Embedded Web Applications Slide 6

Web forms, like the rest of any web page, are designed in HTML. They are recognizable as the sections contained within <form> tags. Within the <form> tags, there is a series of <input> tags. Each input tag represent some user-interface element (such as a text field, checkbox, or a button). It is assigned a "name" parameter, which controls the name of the field when it gets returned to the browser.

When variables are submitted to the server, they are encoded as a series of name/value pairs. Each field is separated by an ampersand symbol (&) and names are separated from values with an equal sign (=). Non-alphanumeric characters, if present, are escaped by a percent sign (%) and then followed by two characters representing the ASCII value in hexadecimal.

Numerous websites explain how to create web forms, but that is outside the scope of this presentation. If you're interested in more information, search for "HTML forms" on your favorite search engine.



Form Methods

- GET `<form method="get" ...>`
  - Data appended to URL
  - Length limited to ~100 bytes
  - Easiest to process
- POST `<form method="post" ...>`
  - Data sent as request body
  - Length is unlimited
  - More difficult to process

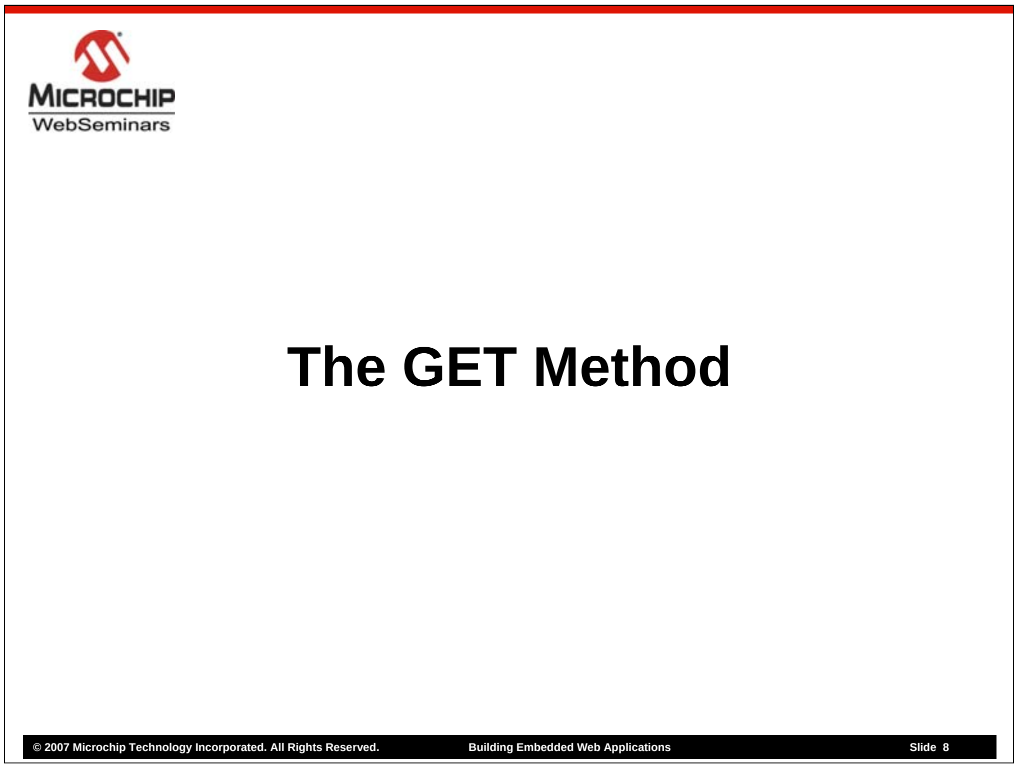
© 2007 Microchip Technology Incorporated. All Rights Reserved. Building Embedded Web Applications Slide 7

Each form tag has a parameter called the "method". This parameter is set to either "get" or "post", which controls how the data is submitted to the server. There's a technical history behind this two, but embedded devices have different needs so this historical differentiation is mostly irrelevant.

The "GET" method appends the data to the URL and arrives before the headers. In the Microchip platform, this data is easiest to process because it is stored in memory all at once. However, the length of all names, values, and delimiting characters is limited to the size of the available buffer – generally 100 bytes.

The "POST" method sends the data as the body of the request, following the headers. On the Microchip platform, this data is more difficult to process because the application assumes responsibility for the TCP buffer. However, this additional complexity allows unlimited amounts of data to be received.

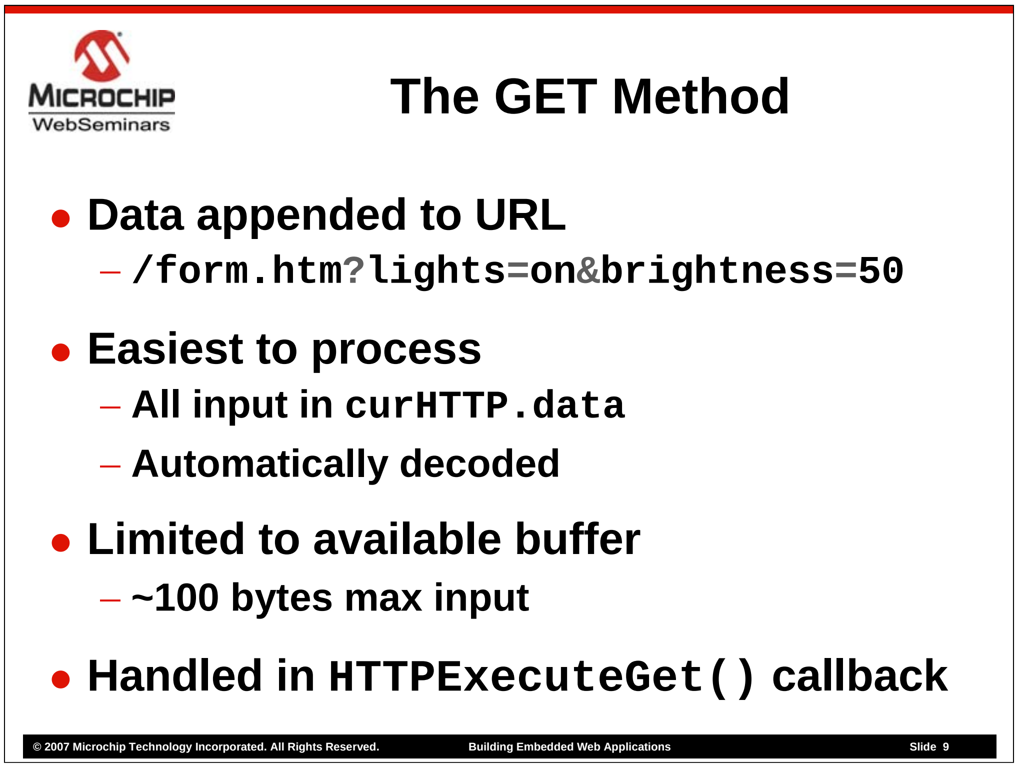
This webinar focuses only on the "GET" method. The "POST" method is described in the next seminar in this series.



The GET Method

Building Embedded Web Applications Slide 8

Let’s take a closer look at the GET method.



The GET Method

- Data appended to URL
  - `/form.htm?lights=on&brightness=50`
- Easiest to process
  - All input in `curHTTP.data`
  - Automatically decoded
- Limited to available buffer
  - ~100 bytes max input
- Handled in `HTTPExecuteGet()` callback

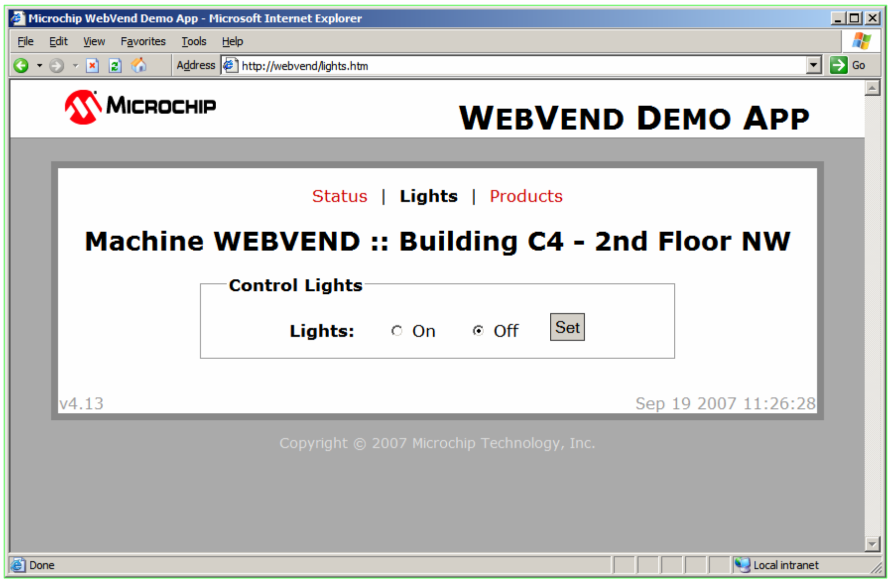
© 2007 Microchip Technology Incorporated. All Rights Reserved. Building Embedded Web Applications Slide 9

As mentioned, the GET method appends all submitted data to the URL. The data string follows a question mark in the URL requested by the client. In the example on the slide, the field “lights” is set to a value of “on”, and the field “brightness” is set to 50.

As mentioned, in Microchip’s HTTP2 web server module, data submitted in this fashion is easiest to process. All data submitted via GET is automatically decoded and placed in the curHTTP.data byte array all at once. Since it is in memory, searching for parameters is simple and efficient. The drawback of course, is that memory is limited, so data submitted in this manner must fit completely in the 100 bytes allocated to each HTTP data buffer.

The HTTPExecuteGet() callback function is called when data is submitted using this method. Like the dynamic variable callbacks, this function is also user-implemented and is defined in CustomHTTPApp.c.

GET Method Example


Microchip WebVend Demo App - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://webvend/lights.htm
Go

WEBVEND DEMO APP

Status | Lights | Products

Machine WEBVEND :: Building C4 - 2nd Floor NW

Control Lights

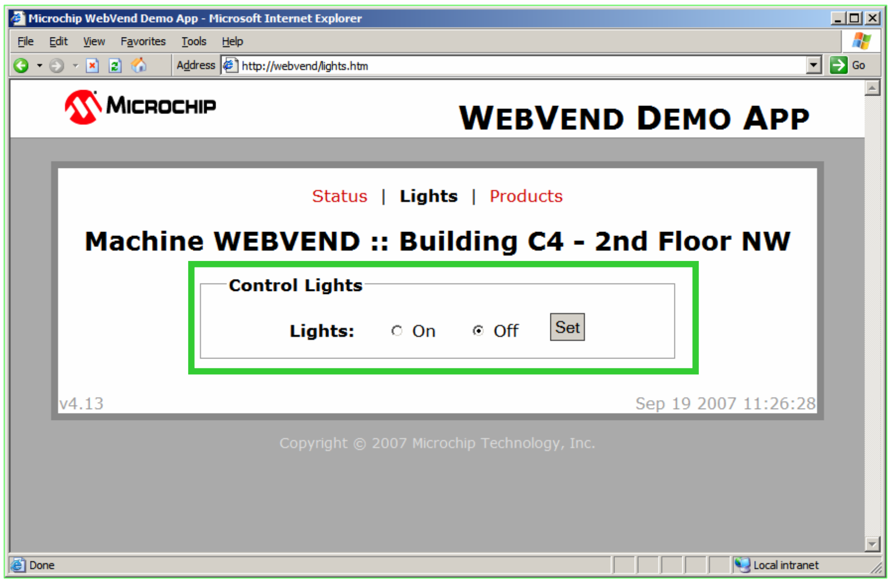
Lights: On [✓ Off] Set

v4.13 Sep 19 2007 11:26:28

Copyright © 2007 Microchip Technology, Inc.

Let’s start with an example.

GET Method Example


Microchip WebVend Demo App - Microsoft Internet Explorer

Address http://webvend/lights.htm

Machine WEBVEND :: Building C4 - 2nd Floor NW

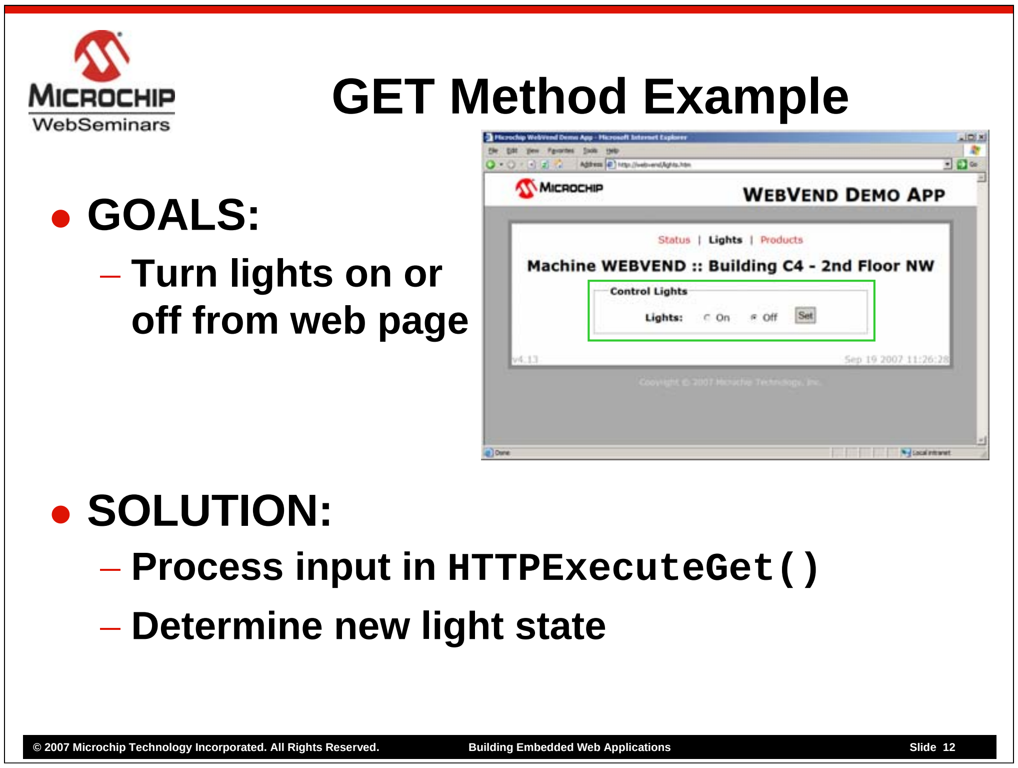
Control Lights

Lights: On [✓ Off] Set

v4.13 Sep 19 2007 11:26:28

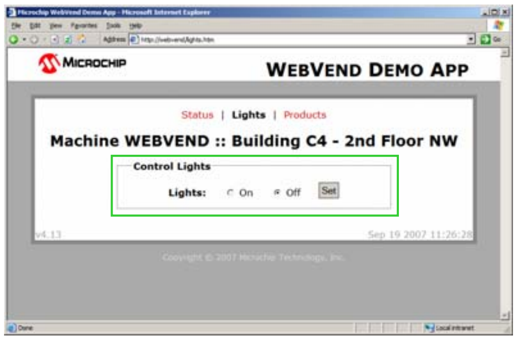
Copyright © 2007 Microchip Technology, Inc.

If we take a look at the “Lights” page, we see a set of radio buttons intended to let us control whether the vending machine’s lights are on or off. On our development board, we’d like this setting to control several of the LEDs. When this form is loaded, we’d also like the current state to be pre-selected.



GET Method Example

- **GOALS:**
  - **Turn lights on or off from web page**



WebVend Demo App

Status | Lights | Products

Machine WEBVEND :: Building C4 - 2nd Floor NW

Control Lights

Lights: On [✓ Off] Set

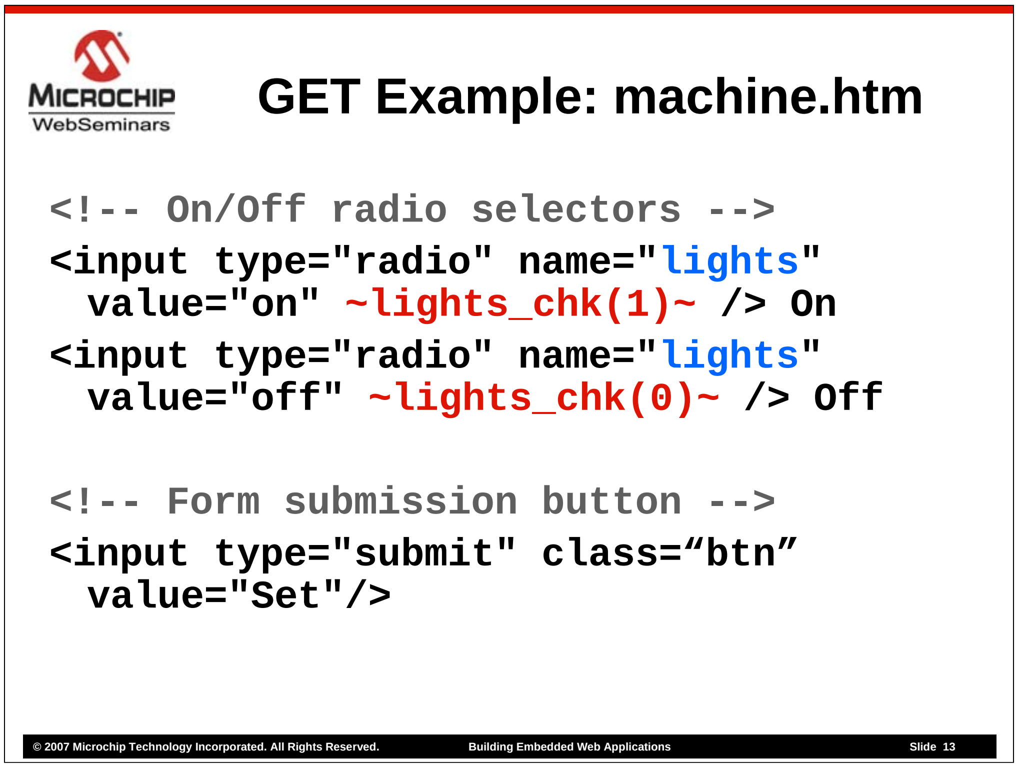
v4.13

Sep 19 2007 11:26:28

Copyright © 2007 Microchip Technology, Inc.

- **SOLUTION:**
  - **Process input in `HTTPExecuteGet()`**
  - **Determine new light state**

To accomplish this, we need to modify our HTTPExecuteGet() function to accept this input parameter. Since we're only sending an "on" or "off" command, it should be safe to assume that our input will fit in the allocated 100 bytes. The GET method will be appropriate for this situation.



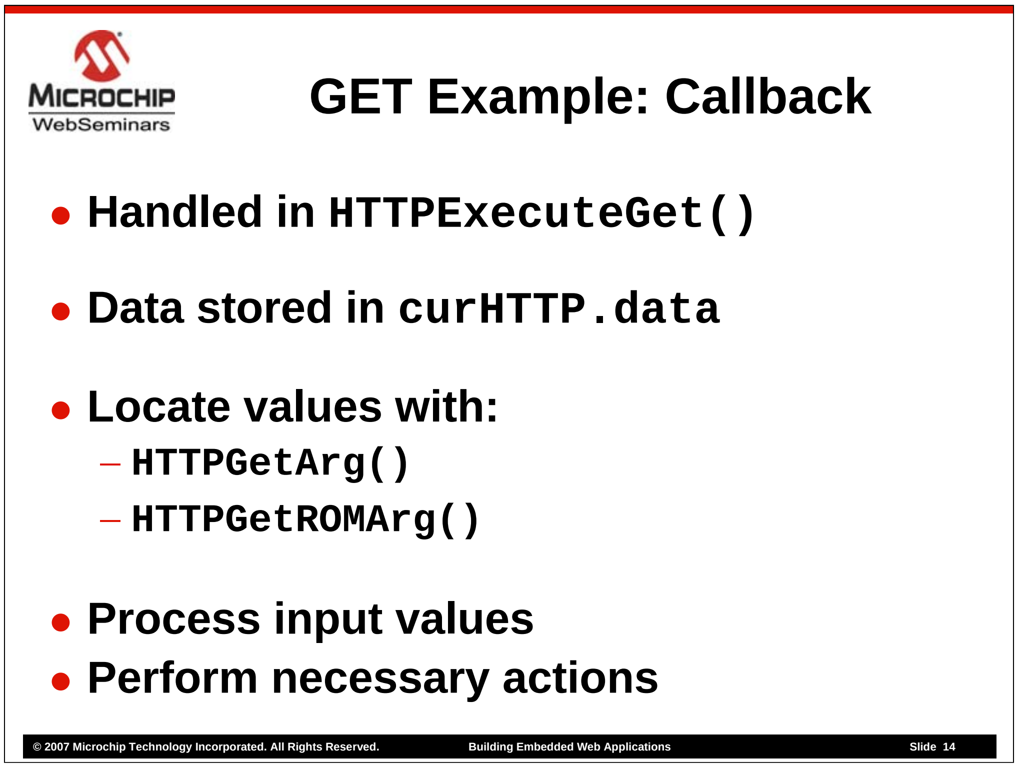
GET Example: machine.htm

```
<!-- On/Off radio selectors -->
<input type="radio" name="lights"
  value="on" ~lights_chk(1)~ /> On
<input type="radio" name="lights"
  value="off" ~lights_chk(0)~ /> Off

<!-- Form submission button -->
<input type="submit" class="btn"
  value="Set"/>
```

We will start by taking a look at the source code to "lights.htm". Scan for the <form> tags, then locate the <input> tags that describe these fields. They should look something like the tags on the slide. Remember that the "name" parameter controls the name of the field when it gets submitted to the application. This is highlighted in blue, and will be important when we write our callback function. You'll also notice a dynamic variable, highlighted in red, placed inside each tag. These dynamic variables control which of the two settings are selected by default when the form loads. Since the last seminar explained dynamic variables in detail, they have already been written and will not be described here.

The final <input> tag creates the button to submit the form. Since it has no "name" parameter, it will not be sent with the form data.



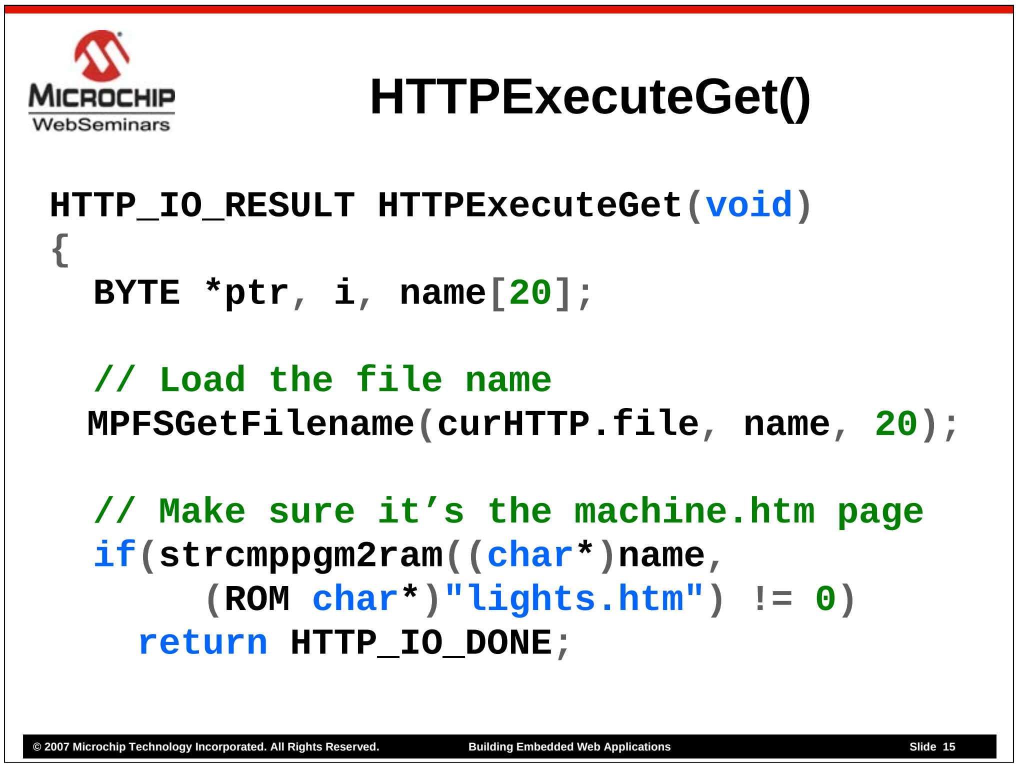
GET Example: Callback

- Handled in `HTTPExecuteGet()`
- Data stored in `curHTTP.data`
- Locate values with:
  - `HTTPGetArg()`
  - `HTTPGetROMArg()`
- Process input values
- Perform necessary actions

Building Embedded Web Applications Slide 14

Our web page is ready to go, so let's write our callback function. As mentioned before, data submitted via GET is handled in the HTTPExecuteGet() callback function. Data received is stored in the curHTTP.data byte array. The server decodes it automatically into a series of null-terminated string pairs. This format allows the provided HTTPGetArg() functions to quickly locate values associated with names.

The basic form of an HTTPExecuteGet() function is to search for input values and perform the necessary actions once these values are found. Let's look at an example.



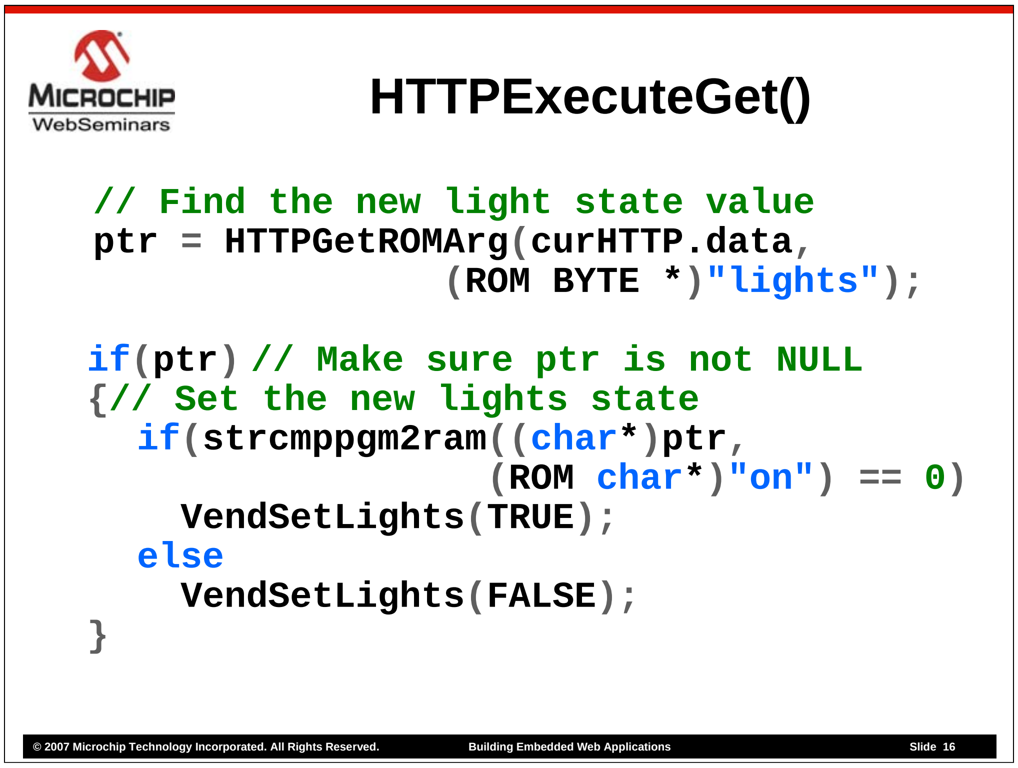
HTTPExecuteGet()

```
HTTP_IO_RESULT HTTPExecuteGet(void)
{
  BYTE *ptr, i, name[20];

  // Load the file name
  MPFSGetFilename(curHTTP.file, name, 20);

  // Make sure it’s the machine.htm page
  if(strcmppgm2ram((char*)name,
       (ROM char*)"lights.htm") != 0)
    return HTTP_IO_DONE;
```

The first step in our callback function is to check the file name. We can use the MPFSGetFilename() function to look up the name of the file requested. For applications with multiple forms, simple logic can be added here to handle data from different forms. For our case, we only have a single form on the “lights.htm” page, so if that’s not the page we’re working with then we should ignore the input and indicate that we’ve finished.



HTTPExecuteGet()

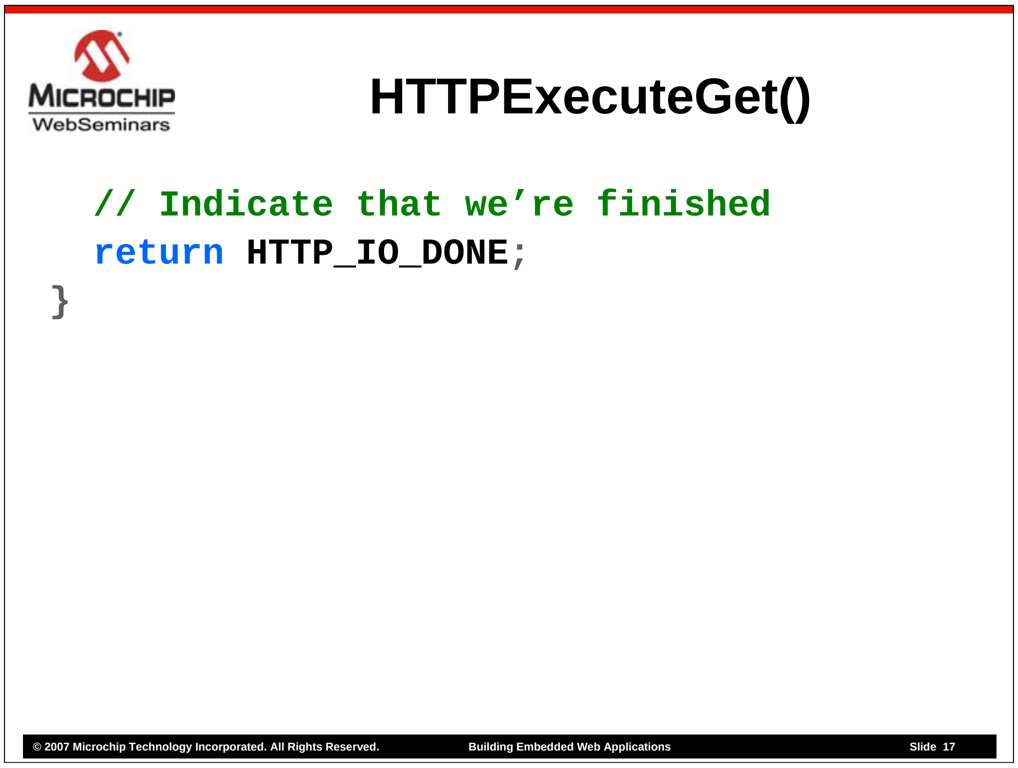
```
// Find the new light state value
ptr = HTTPGetROMArg(curHTTP.data,
                    (ROM BYTE *)"lights");

if(ptr) // Make sure ptr is not NULL
{// Set the new lights state
  if(strcmppgm2ram((char*)ptr,
                    (ROM char*)"on") == 0)
    VendSetLights(TRUE);
  else
    VendSetLights(FALSE);
}
```

Building Embedded Web Applications Slide 16

The next step is to search for the data field. We only have one field, but fields may not always arrive in the same order, so this process is identical for forms with multiple inputs.

First we search for the field called “lights”. If it is found, ptr will point to the value associated with that name. We then compare this value with another string. If they match we set the lights on, otherwise we turn them off.

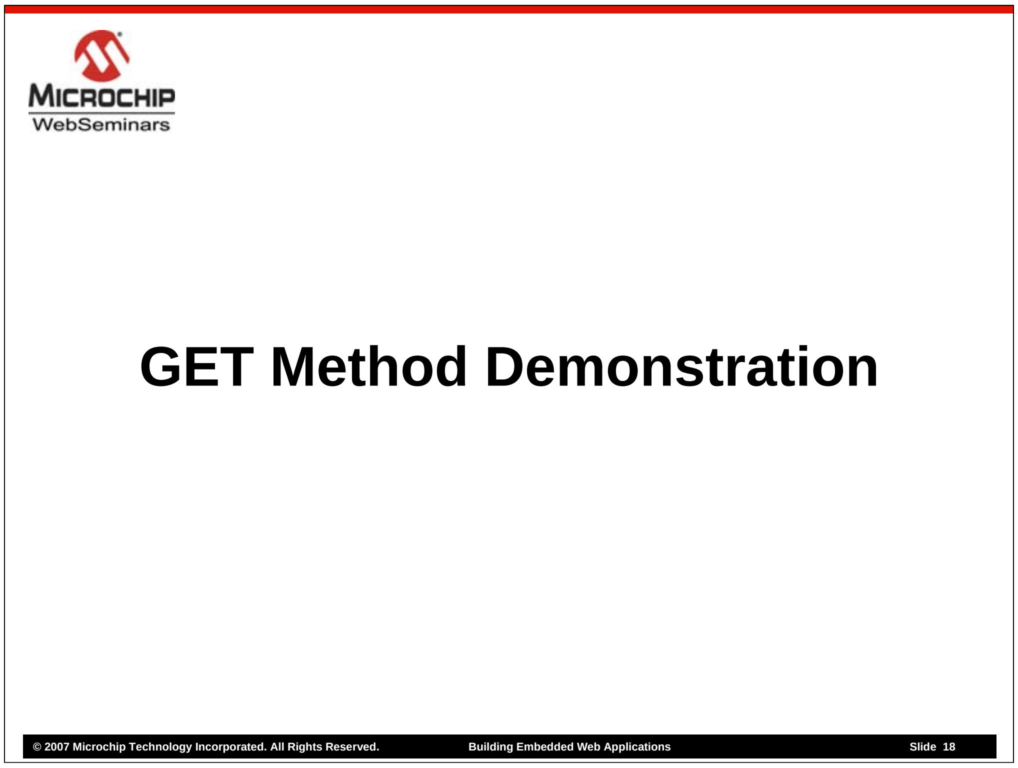


HTTPExecuteGet()

```
    // Indicate that we're finished
    return HTTP_IO_DONE;
}
```

That's all we need to do in order to turn on the lights. The last step is to return HTTP_IO_DONE to indicate that our callback is complete and should not be called again.

It is worth noting that you could return HTTP_IO_WAITING at this point. When that flag is sent, the server will return control to the main loop and will invoke your callback again at a later point before it continues with this request. This is useful if you wanted to wait for some asynchronous process to complete before continuing, but is outside the scope of this presentation.



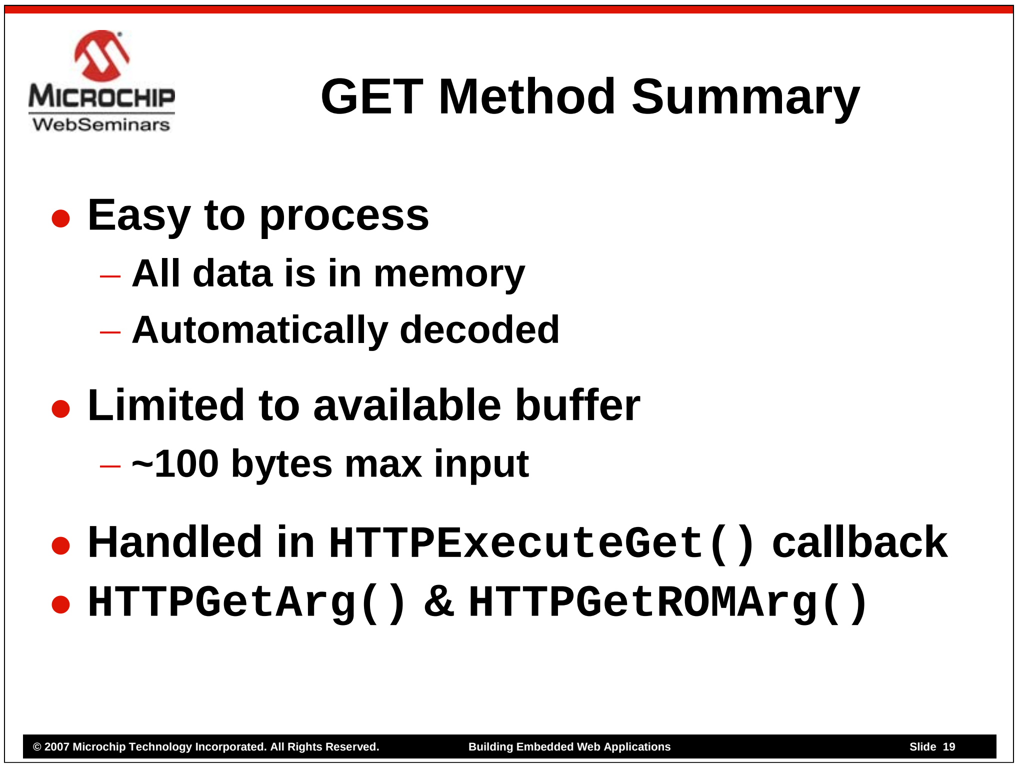
GET Method Demonstration

Building Embedded Web Applications Slide 18

Now that our implementation is complete, let’s try it out.

I’ll start by accessing the device from my web browser. If I navigate to the “Lights” page, you’ll see the form and notice that the lights are currently indicated to be off. If I choose the On selector and click Set, the form will submit this value to the device and the LEDs will illuminate. I can now navigate back to the Status page from the previous seminar. When I return to the Lights form, the On indicator is already selected. If I choose “off” and click Set again, the LEDs on my development board will turn off.

It looks like this portion is working, so our job is complete!



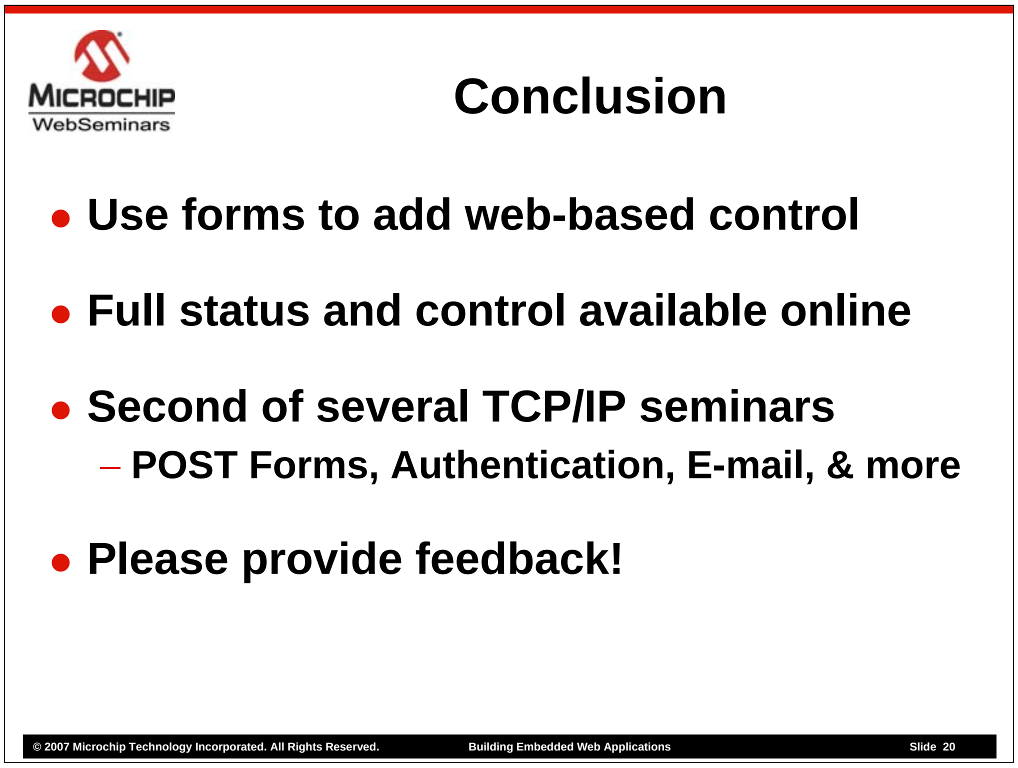
GET Method Summary

- Easy to process
  - All data is in memory
  - Automatically decoded
- Limited to available buffer
  - ~100 bytes max input
- Handled in `HTTPExecuteGet()` callback
- `HTTPGetArg()` & `HTTPGetROMArg()`

Building Embedded Web Applications Slide 19

You've just learned how to process data submitted via the GET method. This method is simplest because all the data is in memory at once, and it is automatically decoded. However, it is only appropriate for inputs totaling less than 100 bytes.

Forms submitted via GET are handled in the HTTPExecuteGet() callback. HTTPGetArg() and its associated HTTPGetROMArg() function are the provided as an easy way to parse this data and find desired values.



Conclusion

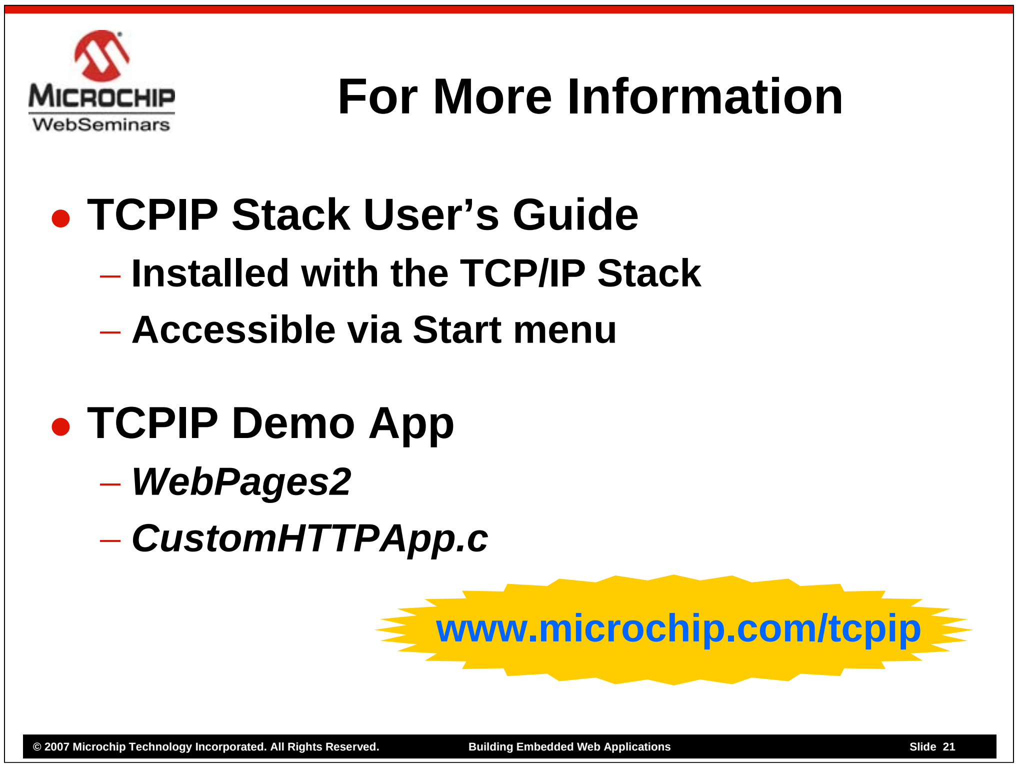
- Use forms to add web-based control
- Full status and control available online
- Second of several TCP/IP seminars
  - POST Forms, Authentication, E-mail, & more
- Please provide feedback!

© 2007 Microchip Technology Incorporated. All Rights Reserved. Building Embedded Web Applications Slide 20

The tools presented today should provide everything needed to add web-based control to your own application. You can now build a complete online status and control system for any embedded device!

As mentioned at the beginning, this is one of several TCP/IP seminars. The next session will explain how to receive larger amounts of data using the POST method. Additional seminars are also being developed to explain how to add authentication, send e-mails, build a data logger, and explain a variety of network-related topics.

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

- TCPIP Stack User's Guide
  - Installed with the TCP/IP Stack
  - Accessible via Start menu
- TCPIP Demo App
  - *WebPages2*
  - *CustomHTTPApp.c*

www.microchip.com/tcpip

© 2007 Microchip Technology Incorporated. All Rights Reserved. Building Embedded Web Applications Slide 21

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 in more detail how everything works.

Finally, our online TCPIP and Ethernet design center has many more resources to get started with your own project.

Thanks for watching!