|  |  |  |
| --- | --- | --- |
| Cell | **Visuals** | **Audio** |
| 1 | Shot of Alan  TEXT ON SCREEN:  Download the manual and solution projects at:  [www.cypress.com/training/wicedwifi-101](http://www.cypress.com/training/wicedwifi-101) | Hi, I’m Alan Hawse. I am Senior Vice President of Technical Staff for Solutions and Software at Cypress Semiconductor. In this video I’ll talk about JSON and how WICED makes it easy for you to read data from a JSON document. |
| 2 | TEXT ON SCREEN:  JSON: JavaScript Object Notation | First, what is JSON? It stands for JavaScript Object Notation. It is an open-standard format that uses human-readable text to transmit data. It is by far the most commonly used format for exchanging data in the cloud so it is important to know how to use it. |
| 3 | TEXT ON SCREEN:  {  “name” : “Alan” ,  “iot expert” : true,  “children”: [“Anna”,”Nicholas”],  “address” : {  “number” : 201,  “street” : “East Main Street”,  “city” : “Lexington”,  “state” : “Kentucky”  }  } | JSON supports double precision floating point values, strings, Booleans, arrays, and key/value pairs. As an example, the following JSON document shows information about me.  First is my name which is a string, then whether or not I am an IoT expert, which is a Boolean set to true. Then I have an array with the names of my two children, Anna and Nicholas. Finally, I have my work address which contains a number represented as a floating point value, and the rest of the address represented as strings.  Note that carriage returns and spaces (except within the strings themselves) don’t matter. |
| 4 | Shot of Alan | There are 2 JSON parsers in the WICED – cJSON and JSON\_parser.  The first parser, cJSON, is an open source project that was created by a very clever programmer named Dave Gamble. He described: “cJSON aims to be the dumbest possible parser that you can get your job done with. It's a single file of C, and a single header file.“  The bottom line is that it is super lightweight and easy to use.  cJSON reads in the entire document at once and then lets you access the data with various API functions. This is the simpler of the two parsers and it is the one we will focus on here.  JSON\_parser does not read the entire document at once. It is a more complicated to use than cJSON but it is useful in situations where you have a very large document that can’t practically be read all at once. |
| 5 | TEXT ON SCREEN:  {  “i2cleds” :  {  “1” : “on”  “2” : “off”  “3” : “on”  “4” : “off”  }  “gpioleds” :  {  “1” : “on”  “2” : “on”  }  } | In this case we are going to use JSON to specify LEDs to turn on and off on our shield board. 2 of the LEDs are connected directly by GPIOs while 4 LEDs are controlled by writing over I2C to the PSoC on the shield.  The JSON will look like this. We use one keymap for the I2C LEDs and one keymap for the GPIO LEDs. In this example, we will turn 4 LEDs on and 2 LEDs off. |
| 6 | SCREEN CAPTURE: Show README file. | You can look at the README file inside libraries/utilities/cJSON/README for a description of the API and some useful examples. And you can find all of the original source code on github at https://github.com/DaveGamble/cJSON |
| 7 | SCREEN CAPTURE: Show project creation process. Show project h file. | We will start with a new project called 04/04\_cjson. Remember that I am keeping all of the chapter 4 projects in the 04 directory  To use the library, we need to add one line to the make file as shown here to include the library functions. |
| 8 | SCREEN CAPTURE: Show project  C file. | In the C file, we need to include cJSON.h.  Next, we will add a constant char array to hold the JSON data. Normally you would be receiving this from the cloud, but for now I will just hardcode the json that I am parsing.  Note that we need to escape the quotes within the JSON with a backslash since we don’t want the C compiler to interpret them. Unforunately that makes is a little bit messy to look at  In the main application, we write to the I2C to allow us to control the LEDs on the shield and then we read the JSON data.  Once we have the JSON data, there are functions that parse out the sections and then the individual entries for the GPIO controlled LEDs and the I2C controlled LEDs. Finally, the corresponding LEDs are turned on or off based on the requested states. |
| 9 | TEXT ON SCREEN:  cJSON\_Parse  cJSON\_GetObjectItem | The key library functions are:  cJSON\_Parse which reads in the file, and cJSON\_GetOpbjectItem which traverses through the JSON hierarchy one level at a time to get the value you are looking for. The structures used by these functions are pointers to structures of type cJSON.  These structures contain pointers to the next/previous items, the type of data, and the data itself. |
| 10 | SCREEN CAPTURE: Show abbreviated programming process.  VIDOE: Show LEDs on the kit lit up. | Now program the project and see what it does. Look – 2 of the I2C LEDs and both GPIO LEDs turn on, just like we expected.  If you want, you can try changing the JSON to turn on/off different LEDs. |
| 11 | Shot of Alan | If you don’t have the shield you can still run the project, but it will only control the 2 LEDs on the baseboard. Note that one LED on the baseboard is active high while the other is active low so the JSON above will turn on one LED and will turn off the other. |
| 12 | TEXT ON SCREEN:  Cypress Developers Community  community.cypress.com  Show video of email and twitter windows. | As always, you can post your comments and questions in our Wifi developer community or you are welcome to email me at alan\_hawse@cypress.com or tweet me at @askioexpert with your comments, suggestions, criticisms and questions. |