EMBEDDED OPERATING SYSTEMS

Embedded Linux on Beaglebone Black

JSON: What?

- JSON
  - **J**ava**S**cript **O**bject **N**otation
  - Light-weight text based open standard
  - For **human-readable** Data Interchange
  - Extended from the JavaScript scripting language
- Specified by Douglas Crockford
  - Is now an RFC (4627)
- Official Internet media type
  - ***application/json***
- Filename extension
  - .json

JSON: Uses

- Writing JavaScript based applications
    - Includes browser extensions and websites
- Serializing and transmitting structured data
    - Over a network connection
- Transmitting data
    - Between server and web applications
- Used by web services and APIs
    - To provide public data
- Supported by most modern programming languages

JSON: Syntax

- Object
  - Collection of Data as **Key / Value** pairs
    - Key / Value pairs separated by colon “:”
- Keys are always strings
  - Unique within an **object**
- Values can be
  - Strings, numbers, booleans, **null**, objects, arrays
  - Introduces the concept of **nesting**
- Delimitations
  - Curly braces hold **objects**
  - Square braces hold **arrays**

JSON: Example-1

- 1 object
  - **Key** is “book”
  - Whose **value** is an array

- The 2 objects have
  - Key-value pairs, with keys:
    - “id”
    - “language”
    - “edition”
    - “author”

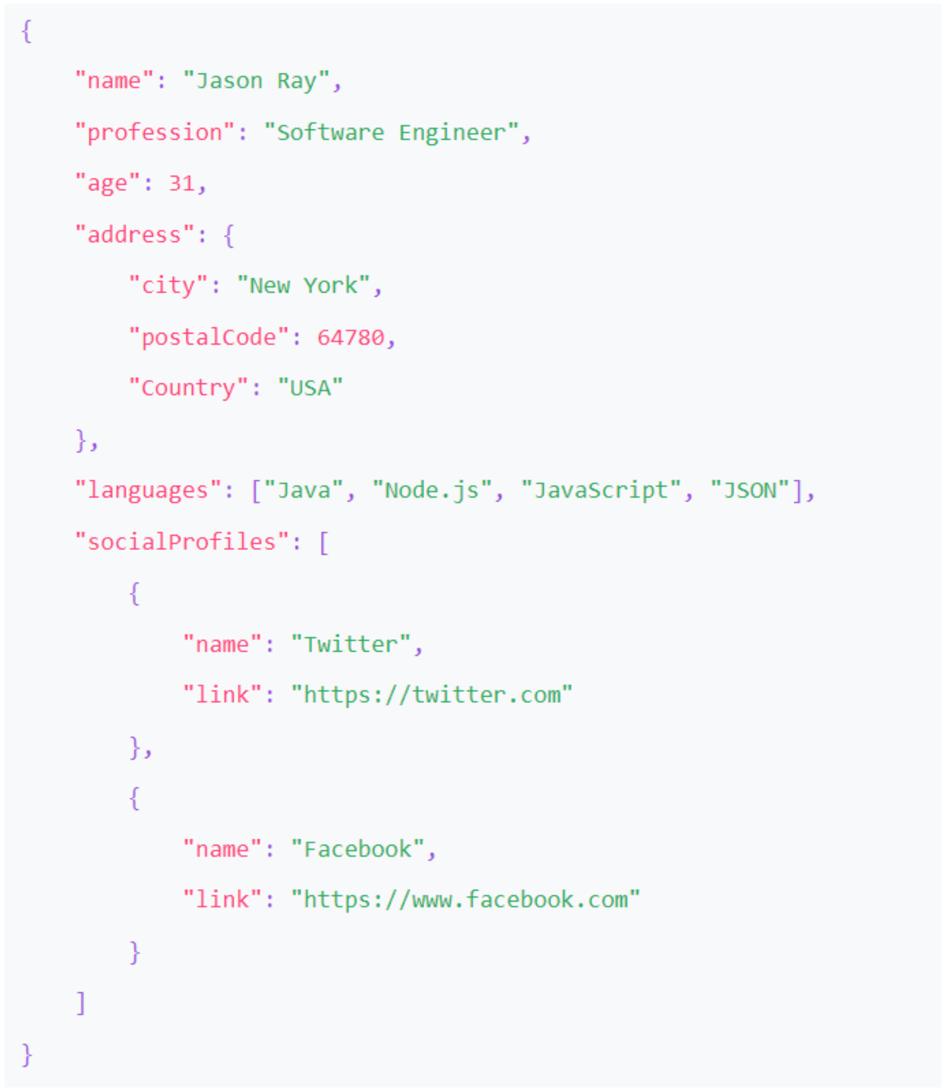
```
{
   "book": [

      {
         "id": "01",
         "language": "Java",
         "edition": "third",
         "author": "Herbert Schildt"
      },

      {
         "id": "07",
         "language": "C++",
         "edition": "second",
         "author": "E.Balagurusamy"
      }

   ]
}
```

JSON: Example-2



```
{
    "name": "Jason Ray",
    "profession": "Software Engineer",
    "age": 31,
    "address": {
        "city": "New York",
        "postalCode": 64780,
        "Country": "USA"
    },
    "languages": ["Java", "Node.js", "JavaScript", "JSON"],
    "socialProfiles": [
        {
            "name": "Twitter",
            "link": "https://twitter.com"
        },
        {
            "name": "Facebook",
            "link": "https://www.facebook.com"
        }
    ]
}
```

JSON vs. XML

- JSON is less verbose than XML
  - XML uses up more characters for delimitations
- JSON is easier to parse
  - Can be parsed in JavaScript
  - XML needs a separate parser
- JSON supports arrays
  - Not supported in XML

XML vs. JSON: Comparison

```xml
<databases>
    <database>
        <name>MySQL</name>
        <type>RDBMS</type>
    </database>
    <database>
        <name>MongoDB</name>
        <type>NoSQL</type>
    </database>
    <database>
        <name>Neo4j</name>
        <name>Graph DB</type>
    </database>
</databases>
```

```json
{
    "databases": [
        {
            "name": "MySQL",
            "type": "RDBMS"
        },
        {
            "name": "MongoDB",
            "type": "NoSQL"
        },
        {
            "name": "Neo4j",
            "type": "Graph DB"
        }
    ]
}
```

cJSON: What?

- Ultra-light-weight JSON parser
  - In ANSI C
- Open-source, MIT License
- Github website:
  - https://github.com/DaveGamble/cJSON
- Installation on Ubuntu

  $ sudo apt install libcjson1 libcjson-dev

  Linking flags: -lcjson

cJSON API: Structs

- Header:

```
#include <cjson/cJSON.h>
```

- The **cJSON structure** – *represents a JSON value*

```
typedef struct cJSON
{
    struct cJSON *next;
    struct cJSON *prev;
    struct cJSON *child;
    int type;               // bit flag representing cJSON type
    char *valuestring;
    int valueint;           // direct write is deprecated
    double valuedouble;
    char *string;
} cJSON;
```

cJSON types

- Internal cJSON data types
  - *cJSON_Invalid*
  - *cJSON_False*
  - *cJSON_True*
  - *cJSON_NULL*
  - *cJSON_Number*
  - *cJSON_String*
  - *cJSON_Array*
  - *cJSON_Object*

cJSON type creation

- Datatype creation functions
  - *cJSON_CreateObject()*
  - *cJSON_CreateNull()*
  - *cJSON_CreateTrue(), cJSON_CreateFalse()*
  - *cJSON_CreateNumber()*
  - *cJSON_CreateString()*
  - *cJSON_CreateArray()*
- Deletion
  - *cJSON_Delete()*

cJSON type check functions

- Input is a cJSON *, return is a Boolean
  - *cJSON_IsInvalid()*
  - *cJSON_IsFalse()*
  - *cJSON_IsTrue()*
  - *cJSON_IsNULL()*
  - *cJSON_IsNumber()*
  - *cJSON_IsString()*
  - *cJSON_IsArray()*
  - *cJSON_IsObject()*

cJSON object functions

- *cJSON_AddItemToObject()*
- *cJSON_DetachItemFromObjectCaseSensitive()*
- *cJSON_ReplaceItemInObjectCaseSensitive() – using a key*
- *cJSON_GetArraySize() – for objects too!*
- *cJSON_GetObjectItemCaseSensitive() – access to an item in an object*
- *cJSON_ArrayForEach() – macro for iteration*
- *cJSON_AddNullToObject()*

cJSON array functions

- *cJSON_AddItemToArray() – appends at the end*
- *cJSON_InsertItemInArray() – inserts at an index*
- *cJSON_ReplaceItemInArray() – replace in place, using index*
- *cJSON_DetachItemFromArray() – deletes and gives a pointer*
- *cJSON_GetArraySize() – get the size of an array*
- *cJSON_GetArrayItem() – get item at an index*
- *cJSON_ArrayForEach() – macro for iteration*

cJSON parsers and helpers

- *cJSON_Parse()*
- *cJSON_Print()*

cJSON exercise

- *cjson-writer.c*
- *cjson-reader.c*
  - Write and read back a simple JSON object file

- *cjson-array-writer.c*
- *cjson-array-reader.c*
  - Write a complex JSON file with arrays
  - Read and parse for support for a certain feature

THANK YOU!