Firebase via CURL and JSON-C

By Eddie Marchioni, Jamie Epp, and Kieran McCormick

Instructions for Unix 

This guide uses code from the `libcurl` Example Sources.

Description:

This project covers:

- The basics of contacting a `Firebase Realtime Database Instance`
- How to build and incorporate `CURL` and `JSON-C` into a project compiled for the `Beagle Bone Green`
- How to use `CURL` and `JSON-C` to contact a `Firebase Realtime Database Instance`.

Firebase: 

The first step in our quest to success is understanding basic interactions with a `Firebase Realtime Database Instance`.

If you have used `Firebase` before then you will know its a great `NOSQL` database.

Using `Firebase` is incredibly simple and getting data is as easy as sending a `GET` request to a `URL`.

An example `URL` would be `https://example.firebaseio.com/data.json`

You could browse to this `URL` and get the JSON stored at that location, or your could send a simple `CURL` command.

e.g.

```
$ curl https://example.firebaseio.com/data.json
{"eddie":{"alert":true,"data":6,"ownerName":"Eddie","seed":1510801622183,"datal":1}}
```

The `JSON` returned by the previous command gives us a great view of the data. Below is our database!

```
{
  "eddie" : {
    "alert" : true,
    "data" : 6,
    "ownerName" : "Eddie",
    "seed" : 1510801622183,
    "datal" : 1
  }
}
```

Above is the data stored at `data` in our database.

Updating the database is as easy as using a `PUT` method. As we can see above, our `JSON` object has a `data.eddie` with a `data.eddie.datal`. Updating this data is as easy as:

```
$ curl -d 2 -X PUT https://example.firebaseio.com/data/eddie/datal.json
```

As we can see below our `data.eddie.datal` value has changed from `1` to `2`, wow that was easy!

```
$ curl https://example.firebaseio.com/data.json
{"eddie":{"alert":true,"data":6,"ownerName":"Eddie","seed":1510801622183,"datal":2}}
```

Now that we have a firm grasp of how to access and manipulate our database from the command line we need to figure out how do the same from our `Beagle Bone` program. Lucky for us `CURL` actually provides a great library for just such an occasion.

CURL: 

`CURL` releases: https://curl.haxx.se/download.html

Just like we saw above, we want to use `CURL` to access our data. Smart cookies will have noticed that we are accessing vai the `HTTPS` protocol. This, *of course*, means that we need to have an `SSL` library available for `CURL` during our build.

Ensure the following package is installed on your `Beagle Bone`:

```
# sudo apt-get install libssl-dev
```

Now that our prerequisite is in place we can start our build process.

First, download the latest `CURL` release source to your `Beagle Bone`.

Once downloaded and extracted, run the following commands from the top of the source directory:

```
# ./configure
```

> `Note` The above command should display the following:
>
> ```
> Protocols: DICT FILE FTP FTPS GOPHER HTTP HTTPS IMAP IMAPS
> POP3 POP3S RTSP SMB SMBS SMTP SMTPS TELNET TFTP
> ```
>
> As you can see, `HTTPS` is supported.
>
> If you see the following, however, `HTTPS` is not supported and you will not be able to contact `Firebase`. Ensure that `libssl-dev` is installed before you rerun `./configure`.
>
> ```
> Protocols: DICT FILE FTP GOPHER HTTP IMAP POP3 RTSP SMTP TELNET TFTP
> ```

Now run:

```
# make
# make test # optional
# make install
```

We can now start copying our built files:

```
# mkdir /mnt/remote/curl-bbg
# mkdir /mnt/remote/curl-bbg/libs
# mkdir /mnt/remote/curl-bbg/curl
# cp -rf lib/.libs/ /mnt/remote/curl-bbg/libs #The library
# cp -rf include/curl/ /mnt/remote/curl-bbg/curl #The headers
```

Don't forgot that we built our `libcurl` against `libssl-dev`.

Our new, amazing library has dependencies built for our `arm-linux-gnueabihf` compiler.

If we tried to incorporate `libcurl` into our projects on our host machines our cross compiler would tell us it couldn't find the following dependencies for `libcurl`:

- libssl.so.1.0.0
- libcrypto.so.1.0.0
- libz.so.1

The first step is to determine which, if any of these we already have on our `Beagle Bones`

For example, in `/usr/lib/arm-linux-gnueabihf/` on my `Beagle Bone`, I found the following:

- /usr/lib/arm-linux-gnueabihf/libssl.so.1.0.0
- /usr/lib/arm-linux-gnueabihf/libcrypto.so.1.0.0
- /usr/lib/arm-linux-gnueabihf/libz.so

Copying these files to the following directory on my host machine almost resolved my issue:

```
/usr/arm-linux-gnueabihf/lib/
```

As we saw before, my cross compiler asked for `libz.so.1` but I foolishly provided `libz.so`

The fix for this is, of course renaming the file:

```
$ mv /usr/arm-linux-gnueabihf/lib/libz.so /usr/arm-linux-gnueabihf/lib/libz.so.1
```

We are ready to incorporate our new library!

JSON-C: 

`JSON-C` GitHub repo: https://github.com/json-c/json-c

Incorporating `JSON-C` will be much more straight forward that `CURL`.

First run:

```
# git clone https://github.com/json-c/json-c.git
# cd json-c
# sh autogen.sh
```

followed by:

```
# ./configure
# make
# make check # optional
# make install
```

We can now start copying our built files:

Our libraries can be attained via

```
# mkdir /mnt/remote/json-c-bbg
# mkdir /mnt/remote/json-c-bbg/libs
# cp -rf ./.libs/ /mnt/remote/json-c-bbg/libs #The library
```

Our headers can be found like this:

```
# ls -h ./*.h
./arraylist.h       ./json_object.h           ./math_compat.h
./bits.h            ./json_object_iterator.h  ./printbuf.h
./config.h          ./json_object_private.h   ./random_seed.h
./debug.h           ./json_pointer.h          ./snprintf_compat.h
./json_config.h     ./json_tokener.h          ./strdup_compat.h
./json_c_version.h  ./json_util.h             ./strerror_override.h
./json.h            ./json_visit.h            ./strerror_override_private.h
./json_inttypes.h   ./linkhash.h              ./vasprintf_compat.h
```

So we will copy them over as follows:

```
# mkdir /mnt/remote/json-c-bbg/json
# cp ./*.h /mnt/remote/json-c-bbg/json #The headers
```

We are now ready to use `JSON-C`

Building with Make

Now that we have our headers and libraries on our shared, we can put them in our projects and link against them.

First, lets copy them over:

```
# cp -rf ~/cmpt433/public/curl-bbg/ ./
# cp -rf ~/cmpt433/public/json-c-bbg/ ./
```

Lets assume our project directory looks like this:

```
.
├── Makefile
├── curl-bbg
│   ├── curl
│   └── libs
└── jsonc-bbg
    ├── json
    └── libs
```

This means we can do the following in our `Makefile`

We can add these to our `CFLAGS`

```
-I${CURDIR}/curl_bbg
-I${CURDIR}/jsonc_bbg
```

And these to our `LFLAGS`

```
-L${CURDIR}/curl_bbg/libs/
-L${CURDIR}/jsonc_bbg/libs/
```

with a trailing `-lcurl` and `-ljson-c` of course.

We are now ready to code!

Implementation

Using `CURL` and `JSON-C` is super easy

To get started, the following tutorials are super helpful. `CURL` provides a ton of example snippets, the ones most useful to us are the Get In Memory and HTTPS tutorials.

The first is important as it shows us how to send a GET via `HTTPS`, the second shows how the write callback function to store the data from the response into a chunk of memory.

https://curl.haxx.se/libcurl/c/https.html
https://curl.haxx.se/libcurl/c/getinmemory.html

Once you have combined these tutorials you will see that we attain the `chunk` struct. This struct contains the information returned from the `GET`.

When we use this to get the `JSON` from `Firebase` we can start to look at how to actually work with our data through `C`.

We can then use this data as follows:

```
char *str = chunk.memory;
```

This string can then be used by `JSON-C`:

```
struct json_object *jobj = json_tokener_parse(str);
```

Once we have mastered this we can look at the HTTP PUT tutorial. This tutorial includes a callback to write the response of the `PUT` in memory. We, however, don't need this as there is no response from `Firebase` to `PUT`.

https://curl.haxx.se/libcurl/c/httpput.html

From this tutorial we can infer that the following is possible:

```
const char *message= json_object_to_json_string_ext(jobj, JSON_C_TO_STRING_PLAIN);

curl_easy_setopt(curl, CURLOPT_POSTFIELDS, message);
```

I bet you never thought sending and receiving `JSON` from `Firebase` via `C` could be this easy!