# Chapter 7c: Cloud Connectivity using HTTP 1.1

## Objective

At this end of Chapter 7c you will understand

* The HTTP 1.1 Protocol
* How to create an HTTPS connection using TLS
* How to use HTTP(s) to read & write data to the Cloud using RESTful APIs
* How to create a JSON message and send it using HTTP(s)
* Using other WEBAPIs

## Time: 4 Hours

## Fundamentals

http1.1 versus http 2.0

### HTTP 1.1

HTTP 1.1 is a application layer, plain-text, client-server protocol. It starts with the client opening a TCP socket to the server (or a TLS TCP socket to the server). The client then sends three things

1. Client Request Start Line
2. Headers (one or more in the form of “headername: headervalue”
3. Content (as many bytes as required e.g. a file or an html page or a json document)

The server will then respond with a

1. Server Response
2. Header
3. Content

The client can then

1. Close the connection
2. Leave the connection open to possibly send another request (the Server will eventually close the connection after a timeout of unspecified length … generally in the range of seconds

**Client Request Start Line**:

The client request start line has three

1. The HTTP Method
2. The Request Target
3. The version of HTTP
4. A “\r\n”

An example legal client request line is

GET /ask HTTP/1.1

**HTTP Method**

Get

Put

Post

patch

Options

Head

Headers

Required

optional

idempotent

URI

http authentication

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT

### The “Content-Type” Header a.k.a MIME Types

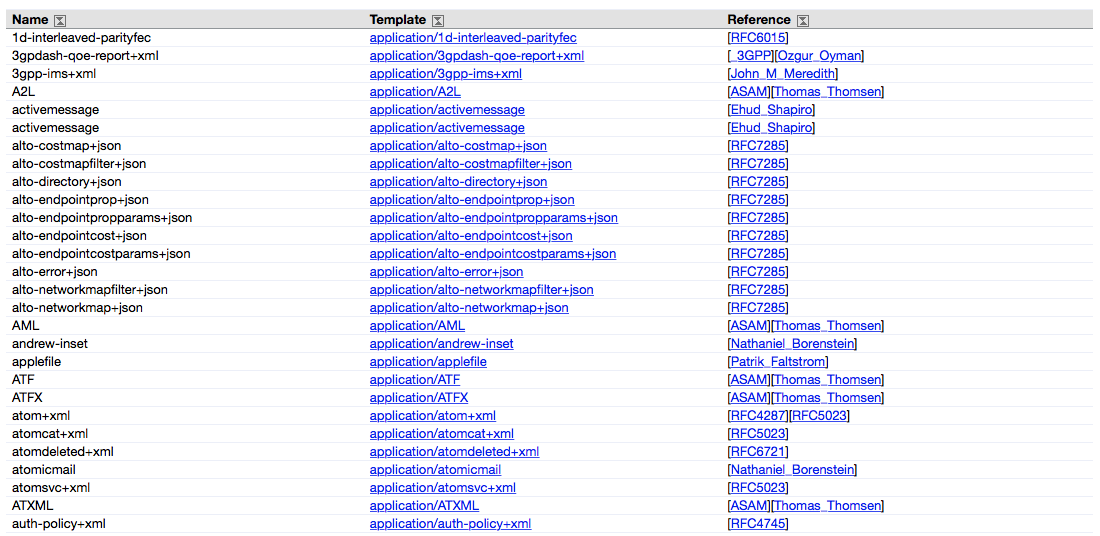
The legal values of the “Content-Type” header are also knows as “MIME Types”. MIME (an old acronym that means Multipurpose Internet Mail Extension) types are specified by the [IANA](https://www.iana.org/) and can be found on their [website](https://www.iana.org/assignments/media-types/media-types.xhtml). Some of the types that are probably useful for IoT applications include:

application/json

application/xml

text/plain

The list runs to 100’s of possible types



### Curl

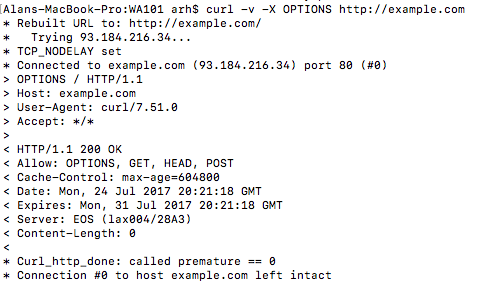
Curl (Client for URLs or See URL) is a Unix (Linux, MacOS, Cygwin) utility for sending and receiving HTTP requests. Curl is a handy tool to help you figure out what an HTTP website is doing so that you can build your WICED program to do the same thing. Curl will let you create HTTP requests with all the commands (GET, POST, PUT, …), any headers you want, plus any content that you want. As with most Unix utilities it is completely out of control with options

For example, if you want to see what options are available on a website you can type the command “curl -v -X “OPTIONS” http://example.com”. This example will build an HTTP message that looks like this:

OPTIONS / http/1.1

HOST somewebsite.com

The website will then reply with the HTTP options that it supports and you will see the output on the terminal (because of the -v)



Curl support both http and https. If you specify the root certificate using the --cacert option, curl will validate the certificate before proceeding with the http transaction.

Some of the useful options are:

|  |  |
| --- | --- |
| **Option** | **Explanation & Example** |
| -v | Verbose: all the http request and response will be echo’d to the screen |
| curl -v http://httpbin.org/get |
| -X “command” | Curl will execute the specified HTTP command GET, POST, PUT, DELETE, OPTIONS, TRACE, CONNECT, HEAD. If you use PUT, POST you need to specify the content by adding --data |
| curl –v –X “OPTIONS” http://httpbin.org/get |
| -H “headername:headervalue” | Adds a header to the HTTP request. You can have multiple –H to add multiple headers. If you specify a header that CURL does automatically e.g. “Content-Type:” it will be overridden by specifying this option. |
| curl -v -H “x-some-custom: someValue” http://httpbin.org |
| -d “data”  --data “data” | Specifies the data for a PUT, POST. CURL will automatically add the “Content-length:” header and “Content-Type: application/x-www-form-urlencoded” header. |
| curl -v -X "PUT" -H "content-type: application/json" -d "{asdf}" http://httpbin.org/put |
| -o filename | Send output to filename. This only sends the content, not the headers to the file |
| curl –o blah.json http://httpbin.org/get |
| --head | Curl will make the method HEAD. You will need to use the –v to see the headers because there will be no content sent back by the http server |
| curl –v --head http://httpbin.org/get |
| --cookie “value” | This will add the header “Cookie: value” to your header |
| curl –v --cookie “name=arh” <http://httpbin.org/get> |
| --cacert server\_cert.pem | Verify the certificate of the https connection with the certificate.pem root ca. In the example below, if the httpbin.pem does not match the root certificate received from httpbin.org. |
| curl --cacert httpbin.pem https://httpbin.org/get |
| --cert client\_cert.pem | Send client\_cert.pem to the HTTPS server to verify the client identity |
| curl --cert client\_cert.pem https://httpbin.org/ |

This [link](https://curl.haxx.se/docs/httpscripting.html#The_HTTP_Protocol) takes you to a useful tutorial using Curl with HTTP.

### WICED HTTP Libraries

### REST & RESTful APIs

Representational State Transform (REST)

REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

https://en.wikipedia.org/wiki/Representational\_state\_transfer

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\_arch\_style.htm

http://www.restapitutorial.com/lessons/whatisrest.html

Use and HTTP action (GET, POST, PUT, DELETE, PATCH) to a URI (URL)

#### RESTful APIs

Uniform interface

HTTP actions (Get, Put, Post, Delete)

URIs are resource names

Stateless

Each message is self descriptive… the server doesn’t maintain client state … implies state is kept on client and the requests are complete

Cacheable

Server responses are cacheable (implicitly, explicitly or negotiated)

Client-Server

Assume that the system is disconnected

HTTP is the interface between the client and server

Layered System

Client can’t assume direct connection (can be multiple layers between)

Scalability

Code on Demand (optional)

Logic can be transferred to the client

Web API Directory

<https://www.programmableweb.com/category/all/apis>

<https://www.wunderground.com/weather/api>

Twitter

Google cloud platform & google translate <https://cloud.google.com/translate/docs/translating-text>

URL options

Document options

API key

HTTPS

### SOAP

### JSON

## XML

## Httpbin.org

## Initialstate

## AWS

## Exercise(s)

### Example.com

Write a TCP socket program to send an HTTP request to example.com and print the result HTML to the debug UART

### Use webapi to convert board temperature from C to F using JSON

1. <http://webservices.daehosting.com/services/TemperatureConversions.wso?op=CelsiusToFahrenheit>

## CURL

## Save an API Key in the DCT

## Related Example “Apps”

|  |  |
| --- | --- |
| **App Name** | **Function** |
| http\_sever\_sent\_events | starts, pings gateway, then starts AP |
| httpbin\_org | Use HTTPS to get data from httpbin.org |
| https\_client | Use HTTPS to get data from google HTTPS server and print it to the screen |
| http\_server | WICED Station with an HTTP Server running |

## Known Errata + Enhancements + Comments