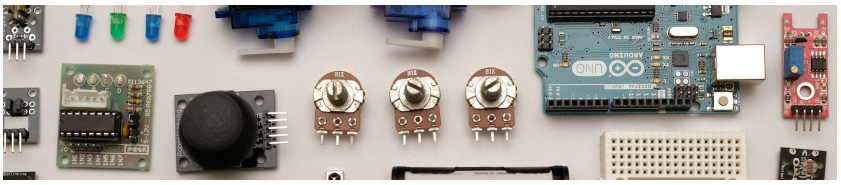
Using the Arduino NTPClient Library

johnwargo.com/posts/2021/using-the-arduino-ntpclient-library/



Posted: July 18, 2021 | | **Categories:** Internet of Things (IoT), Arduino

As I mentioned in my previous post, I have a version of Andy Doro's Word Clock running in my house. I had some issues with the real-time clock (RTC) keeping time correctly, so I decided to rebuild the clock using a Wi-Fi enabled device and the Network Time Protocol (NTP) to keep my clock's local time synchronized with reality. I'll write more about that project once I've completed it.

As I started working on my updated project, I discovered that the Arduino team maintains (well, used to maintain) an NTP Client library called NTPClient (https://github.com/arduino-libraries/NTPClient) that would work for my project. The library is pretty easy to use, however it's woefully under-documented. There's a very simple readme in the project repository with an overly simple sample sketch and a vague reference to only one of the functions exported by the library.

Sadly, there are a couple of pull requests (PR) in the repo for documentation changes, but those aren't getting any love from the team. I thought about proposing some updates to the docs in order to clearly document how the library works, but realized they'd likely not get published. However, I may go back and submit them later.

As I tried to learn what the library did, I studied the source code and I now have a better understanding of the inner workings of the library. The purpose of this post is to share some of what I learned.

Note: The library expects the device running the sketch (and using the library) to have a valid network connection (typically over Wi-Fi). You can only retrieve the time from a network time server (using NTP) if you can connect to the server over the network. So any sample code you see using the library will connect to Wi-Fi before doing anything else related to the NCPClient library.

To use the library in your sketch, you must add to libraries:

```
#include <NTPClient.h>
#include <WiFiUdp.h>
```

Then, you must initialize two variables:

```
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);
```

The NTPClient object supports several initialization options (multiple constructors). The default one shown in the example code uses default values for several configuration options. No matter which approach you use, you must provide a Wi-Fi UDP object to the constructor as shown in the example.

UDP stands for User Datagram Protocol which is a communications protocol that enables network devices to exchange messages. Rather than designing its own communication protocols between the client and server, it just uses a standard one. You don't have to do anything with UDP except to create the ntpUDP object as shown in the example and pass it to the NTPClient constructor.

The default constructor also configures the client to use the default NTP server pool (described below) and sets an update interval of 1 minute (more on that later as well).

If you look at the sample application in the NTPClient repository's readme file, you'll also see this code in the sample:

```
// You can specify the time server pool and the offset, (in seconds)
// additionally you can specify the update interval (in milliseconds).
// NTPClient timeClient(ntpUDP, "europe.pool.ntp.org", 3600, 60000);
```

This highlights another supported constructor for the NTP client. In this case, it shows how you can specify a different time server pool, a time offset, and an update interval. Unfortunately, the text doesn't explain what any of this means, so let me see if I can fill in the blanks:

Time Server Pool: To make it easier for applications to retrieve the current time using NTP, the community created a 'big virtual cluster of timeservers' online. Rather than having to know the specific address/name for a time server, you can point your client application to a pool of servers and through Internet Magic get connected to any of the servers in the default pool. You can also specify a specific pool like they do in the example depending on your location (for example specifying the pool for Europe). You can read more about Time Server pools at https://ntppool.org/en/. Unless you have a specific need to specify a pool, I'd just leave things at their default, but I'm not an expert on this topic.

Time Offset: Depending on the application and the time zone you're in, you may want to set a time offset as well. The time offset specifies the number of seconds by which to adjust the retrieved time. The 3,600 seconds shown in the example is one

hour (60 seconds times 60 equals an hour) telling the NTPClient to add an hour to every retrieved time value. The reason this option exists is to enable you to adjust the time for your current time zone. For example, I'm in US Eastern Time (GMT-4) which means I'm 4 hours behind Greenwich Mean Time (GMT) and the time offset for my location would be -14,400 (-4 * 60 * 60).

Update Interval: This parameter seems to indicate that you can specify how frequently the NTPClient updates its time over the network, but that's not the case. The NTPClient doesn't automatically update the time from the network. What this parameter does is control how frequently the NTPClient can update the time, and this is where it gets a little weird (and why I wrote this article); I'll explain this more throughout the remainder of the article.

In your sketch's `setup` function, you'll initialize the NTPClient using:

```
timeClient.begin();
```

Then, in your sketch's loop function or any other place in your sketch, you'll update the time from the network using:

```
timeClient.update();
```

Then, with the time updated, you can retrieve it using `timeClient.getFormattedTime()` or `timeClient.getEpochTime()`. At this point, you'll update something in your sketch or do something interesting with the hardware, perhaps even update the time value stored in a realtime-clock (RTC).

This is where the update interval comes into play; I'm going to refer to it as UpdateInterval going forward. When you executed the update, you probably expected that the client would connect to the network and do its stuff right away. That's not what happens. It only updates the time over the network if it has been at least UpdateInterval milliseconds since the last update. By setting UpdateInterval (it defaults to 1 minute) you set guardrails for how frequently the library will update the time when the sketch specifically asks it to.

Alright, so that last paragraph probably made you want to ask at least two questions: "How do I force an update so I can get the time whenever I want it?" and "What time will I get if I ask for it before the UpdateInterval completes?"

Sadly, I have the answers to those questions only because I studied the source code, not because this is documented anywhere. Sigh.

Alright, to the first question: How do I force an update so I can get the time whenever I want it? To force an update (ignoring UpdateInterval) you simply call:

```
timeClient.forceUpdate();
```

Internally this is the method that update() calls once it checks to see if UpdateInterval milliseconds has passed since the previous update. Easy, right?

To the second question: What time will I get if I ask for it before the UpdateInterval completes? Aaah, excellent question - that's the question that got me looking at the library source code to begin with.

When your sketch calls `timeClient.update()` before UpdateInterval milliseconds has passed, the library returns the previous time value adjusted by the number of milliseconds that passed since the library last updated the time.
So, if that's the case, what's the difference?

I'm not really sure, but I doubt it's much. The Arduino device running the sketch probably has a good reckoning of how much time passed (in clock ticks) since the last time the library checked, however it is an approximation. If you have unlimited network capacity, you can probably check the network time as frequently as you want (although I saw an issue in the NTPClient repository mentioning that the default 60 second interval violates the NTP Pool Server terms of service (https://github.com/arduino-libraries/NTPClient/issues/123)). For my particular application, I'm going to update an RTC using NTP once a day (immediately after midnight). I can't imagine I'd need any more accuracy than that.

Next Post: Implementing Custom Pipes in Ionic

Previous Post: Word Clock Time Issues

If this content helps you in some way, please consider buying me a coffee.

Header image: Photo by Robin Glauser on Unsplash.