C# ALPACA TEMPLATE GUIDE

Abstract

Understand how to build your own Alpaca device using the ASCOM C# template.

Peter Simpson

Contents

Introduction to the C# Alpaca Template

Introduction

The ASCOM C# Alpaca template creates an Alpaca device using Microsoft ASP.NET and Blazor technologies. The generated Alpaca device provides supports Alpaca discovery out-of-the box, making substantial use of the ASCOM.Alpaca.Razor NuGet package to keep device implementation small and enable new features to be added in future.

Initially the template implements a safety monitor device, but you can easily configure the template to serve any ASCOM device type. Furthermore, you can serve multiple devices of the same device type as well as multiple devices of different device types.

We use the “Blazor Server” model rather than “Blazor WebAssembly (WASM)” because this provides a simple model with a single executable running on the host device from which communications to the hardware can be managed

Getting started

The template is created from the command line using a **`dotnet new`** command, but before this will work you need to install the NuGet package containing the template.

Installing the template generator

To install the template:

- Download the latest template: https://www.nuget.org/packages/ASCOM.Alpaca.Templates
- Start a command prompt in the folder where you downloaded the package.
- Install the template: **`dotnet new install .\ASCOM.Alpaca.Templates.x.y.z.nupkg`** where x, y and z are the version numbers from the downloaded file. Include any release candidate text that follows the z parameter.

On completion you should see a message similar to this:

```
The following template packages will be installed:
   D:\Downloads\ascom.alpaca.templates.0.5.0-alpha02.nupkg

Success: ASCOM.Alpaca.Templates::0.5.0-alpha02 installed the following templates:
Template Name       Short Name  Language  Tags
------------------  ----------  --------  ----
ASCOM Alpaca Razor  alpacacs    [C#]      Web
```

Removing the Template Generator

If no longer required, the template can be removed as follows:

- Start a command prompt
- Uninstall the template: **`dotnet new uninstall ascom.alpaca.templates`**

On completion you should see a message similar to this:

```
Success: ASCOM.Alpaca.Templates::0.5.0-alpha02 was uninstalled.
```

Template help and build options

A summary of use and options is shown by the command: `dotnet new alpacacs -h`

Note: **`alpacacs`** is an abbreviation that avoids the need to type **`ASCOM.Alpaca.Templates`**

The help text includes details of options such as driver id and server name that can be set from the command line when the Alpaca device is generated from the template.

The following options can be set from the command line when the template is generated:

- -n <devicename> - The name of the solution/project that will be generated
- -o <outputfolder> - The folder in which the project will be generated (defaults to device name)
- -dr <driverid> - Name of the file used to store configuration information
- -m <manufacturer> - Your name or company
- -p <port> - The IP port on which the Alpaca device will start
- -s <servername> - A friendly name for the server

Note: Use double quotes around parameter values that contain spaces.

Creating your Alpaca device

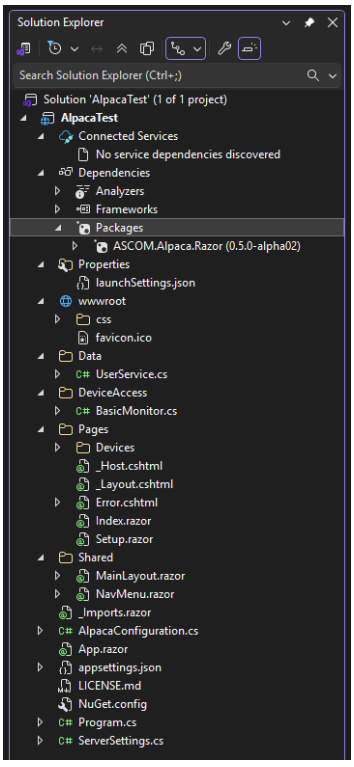
The Alpaca device is created with a command similar to:

- `dotnet new alpacacs -n DeviceName -dr DeviceName.Alpaca -m "Your name" -p 34567 -s "YourName's Alpaca Server" -o C:\Documents\DeviceName`

The project can then be opened in Visual Studio (2022 or later) or an editor of your choice.

Project structure

The project structure looks like this (see *Appendix 1 - Project structure* for a description of each file):



Optimal implementation order and testing

To aid developers, ASCOM publishes the Conform Universal tool that exercises each API member independently and reports on conformance with the relevant interface standard. You can download Conform Universal from this page: https://ascom-standards.org/AlpacaDeveloper/Index.htm . Conform is a robust tool designed to operate in the development environment where devices may be partially implemented or exhibit bugs.

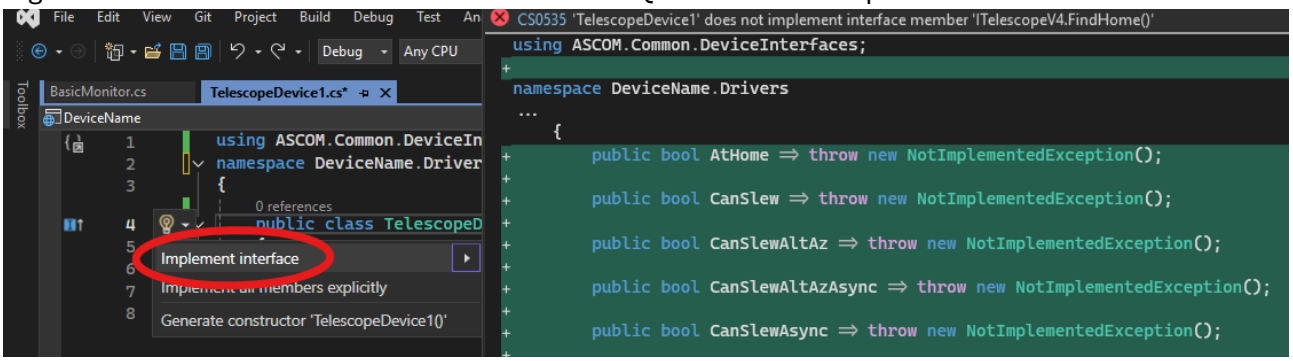
The best approach to creating your Alpaca device to is to implement in the following order:

- **Establish connectivity to your device**
  - Get the low-level routines for communicating with your device operating robustly, including establishing and tearing down the physical link.
  - Implement the Connected, Connect, Disconnect and Connecting members
- **Implement read-only members**
  - These appear as properties in the device class you added. Many are straightforward to implement and will enable you to get experience in working with the template.
- **Implement functional members**
  - These appear as methods and usually implement more complex functionality.

How to add a new device

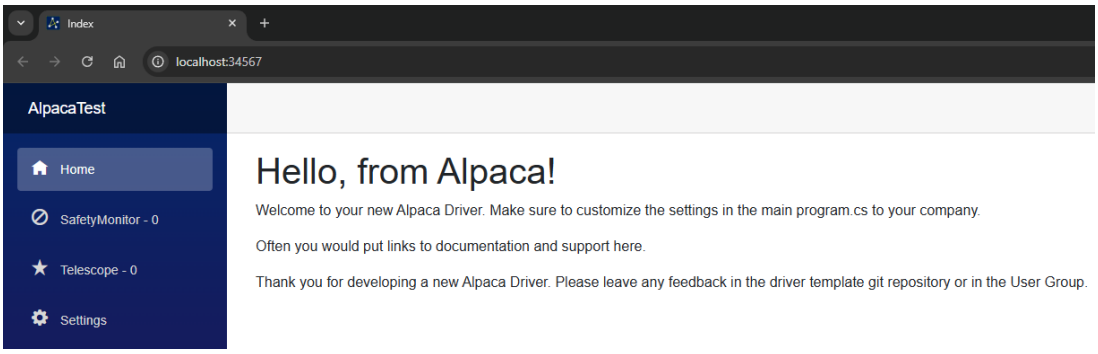
New devices are added as classes to the DeviceAccess folder and then the configuration in **`program.cs`** is updated to load the new class when the device starts. In this example a telescope device is added, which implements ASCOM interface ITelescopeV4.

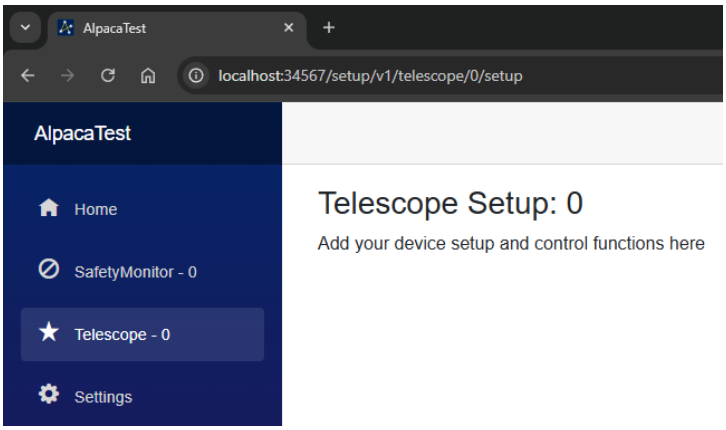
- Add a new class to the DeviceAccess folder with an appropriate name e.g. TelescopeDevice1
- Add a using statement:
  - `using ASCOM.Common.DeviceInterfaces;`
- Add a reference to the latest ascom interface to the class statement:
  - `public class TelescopeDevice1 : ITelescopeV4`
- Right click on the interface statement and use the Quick Action to implement the interface



- Add an entry in **`Program.cs`**, under the **`LoadSafetyMonitor`** line, to load the telescope class when the Alpaca device starts:
  - `DeviceManager.LoadTelescope(0, new DeviceAccess.TelescopeDevice1(), "My Alpaca telescope", ServerSettings.GetDeviceUniqueId("Telescope", 0));`
  - Parameter 1 is the Alpaca device number that will be used in the URL to access the device e.g. http://localhost:12345/api/v1/telescope/**0**/name
  - Parameter 2 is the class that controls this device
  - Parameter 3 is the name of the device as it will appear through the discovery API.
  - Parameter 4 is a unique ID for this device. We recommend using the **`GetUniqueDeviceId`** function to return a consistent GUID.

- The template provides Setup GUI pages for all device 0 devices of all ASCOM device types. This means that you only need to add a Setup page to the Pages/Devices folder if you are supporting 2 or more devices of the same device type.
- You can now compile and run your code. A console application will start (the Razor server application) together with a browser interface to the Alpaca device's management configuration URL. You will see configuration for the Alpaca device itself and for the configured devices:





Implementing the hardware interface

Due to the wide variety of physical communication channels available this section is necessarily high-level.

A good approach is to implement the physical hardware communication inside a static class, located in the DeviceAccess folder, and to call this as needed from the Connected / Connect /Disconnect and other members of the interface.

If hardware responses are expected to take more than 1 second, implement the communications round-trip asynchronously e.g. by using the asynchronous methods in the .NET **`HttpClient`** class or by using async await to handle text transmission and reception through the **`SerialPort`** class.

How to view Alpaca protocol messages

The Alpaca protocol uses an HTTP/REST implementation and network messages between clients and devices can be viewed with WireShark as described in *Appendix 3 – Installing and using WireShark*.

How to use a serial port to communicate with hardware

If your device is attached to a real serial port or a virtual port exposed through USB hardware, you will need to add Microsoft's **`System.IO.Ports`** NuGet package to your project. After this you can communicate with using the **`SerialPort`** component.

The following basic program shows how to list available COM ports and send text to the COM port.

```
using System.IO.Ports;

namespace ComTest
{
    internal class Program
    {
        static void Main(string[] args)
        {
            try
            {
                // Get a list of available serial ports
                string[] portNames = SerialPort.GetPortNames();

                if (portNames.Length == 0)
                {
                    Console.WriteLine("No serial ports found.");
                    return;
                }

                foreach (string portName in portNames)
                {
                    Console.WriteLine($"Found serial port: {portName}");
                }

                // Select the first available port and connect at 9600 baud
                SerialPort sp = new SerialPort(portNames[0], 9600);
                sp.Open();

                // Write some arbitrary text to the port
                for (int i = 0; i < 10; i++)
                {
                    Console.WriteLine($"Writing to {portNames[0]}, Cycle: {i}");
                    sp.WriteLine("Arbitrary text sent to the COM port".ToString());
                    Thread.Sleep(1000); // Wait for 1 second
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Exception: {ex.Message}\r\n{ex}");
            }
        }
    }
}
```

This program should be created as a .NET 8 or later project and published to your target of choice e.g. Windows or Linux or ARM.

Serial ports on Windows are usually named **`COMx`** where x is an integer from 1 upwards. On Linux they have a variety of names e.g. **`/dev/ttyUSB0`** and **`/dev/ttys0`**. If you receive a permissions exception when running the program above on Linux you may need to add your user to the dialup group:

- Add the user to the dialup group: **`sudo gpasswd --add ${USER} dialout`**
- Change the user's group to the dialup group: **`newgrp dialout`**

How to use TCP/IP to communicate with hardware

The following is a basic example of how to use the .NET **HttpClient** component to retrieve a response from an HTTP server, in this example, the OmniSimulator running on the local device:

```
using System;
using System.Net.Http;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        using HttpClient client = new HttpClient();

        try
        {

            string url = "http://localhost:32323/setup"; // Target URL
            HttpResponseMessage response = await client.GetAsync(url);

            response.EnsureSuccessStatusCode(); // Throws if not 2xx

            string responseBody = await response.Content.ReadAsStringAsync();
            Console.WriteLine($"Response received:\r\n{responseBody}");
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine($"Request error: {e.Message}");
        }
    }
}
```

How to add a new persisted configuration setting

Configuration settings are defined as properties in the **ServerSettings** class and are automatically saved when you write to the property. Setting values are persisted to a file stored as follows:

- Windows: **C:\Users\%USERNAME%\.ASCOM\Alpaca\{Program.DriverId}**
- Linux: **$HOME/.config/ascom/alpaca/{Program.DriverId}**

This is an example of implementing a boolean value, other types are implemented similarly:

```
/// <summary>
/// Gets or sets a value indicating whether authentication is enabled.
/// </summary>
/// <remarks>The value is retrieved from or stored in the application profile settings.</remarks>
internal static bool UseAuth
{
      get
      {
           // Retrieve the value. Return false if the value has not yet been set
           if (bool.TryParse(Profile.GetValue("UseAuth", false.ToString()), out bool result))
           {
                return result;
           }

           // Return false if the stored value cannot be parsed to a boolean value
           return false;
       }
       set
       {
            // Persist the value to the backing file
            Profile.WriteValue("UseAuth", value.ToString());
       }
}
```

See the **ServerSettings.cs** file for examples of how to implement string, boolean, integer and enum types.

How to add a configurable setting to the web setup UI.

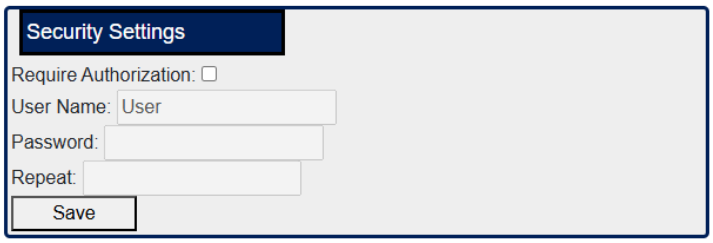
The template's web pages use the Razor markup language to create UI elements; an example page is shown in *Appendix 2 – Example Razor setup page*. This shows how to create string, boolean and integer input fields and to retrieve and store values using the **`ServerSettings`** class.

Successful compilation of this page requires that you have added three properties (**`ExampleBool`**, **`ExampleInt`** and **`ExampleString`**) to the **`ServerSettings`** class following the template in section *How to add a new persisted configuration* setting.

How to configure security

NOTE: *We highly recommended that you never expose an Alpaca device to a hostile environment, particularly not directly to the Internet via a static IP address. Remote access should always be accomplished via a VPN or other similar technology.*

The driver template offers several security features including HTTP Basic Auth support via the "**`Require Authorization`**" checkbox in the "**`Security Settings`**" section on the "**`Settings`**" tab. Here you can set a username and password that must be provided by the client.



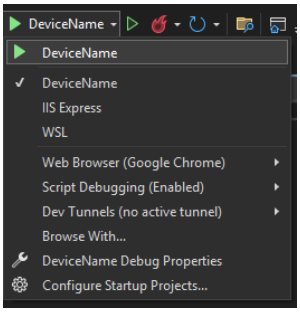
Once enabled, access is allowed only when a correct username and password are provided.

To use more complex access controls, revise the **`UserService`** class in the **`Data`** folder to implement your desired security mechanic. The default class created by the template supports a single basic auth user whose settings are stored unencrypted in the server settings file.

The template supports both HTTP and HTTPS using self-signed and user supplied certificates through the standard built-in .NET Kestrel server options. If you plan to use SSL, we recommended that you set up a web proxy with an SSL host certificate from a trusted certificate authority.

Appendix 1 - Project structure

The broad purposes of files created by the template are:



- **Packages** – Lists NuGet packages used in the project
- **launchsettings.json** – Configuration for various ways of running the application for testing (stand-alone or as a service running under IIS, which browser to use etc.). Devices built with the template are designed to run **stand alone**, see opposite, please do not select IIS or WSL.
- **wwwroot/CSS** – Contains cascading style sheet (CSS) files that format the user interface pages displayed in the browser. You generally don't need to change these if you retain the default style.
- **wwwroot/favicon.ico** - The favicon appears in browser tabs.
- **Data/UserService** – The UserService validates user access. See here for more information.
- **DeviceAccess -** This folder contains a class for each device available from the Alpaca server. See *Implementing the hardware interface* for information on how to add further devices.
- **Pages/Devices** – The Devices folder holds basic configuration pages for all device types
- **Pages/_Host.cshtml** – HTML page that acts as the host page for the Blazor single page application. You should only need to revise this in advanced scenarios.
- **Pages/_Layout.cshtml** – HTML page setting out the overall structure of the rendered page including common content e.g. CSS scripts that are available in every page of the application.
- **Pages/Error.cshtml** – Error page for use in the development environment.
- **Pages/Index.razor** – Home page for the application
- **Pages/Setup.razor** – Device's top-level configuration page
- **Shared/MainLayout.razor** – Overall layout for common elements of the browser page including the page header, navigation and page content areas. Usually, you won't need to change this file.
- **Shared/NavMenu.razor** – This creates the navigation menu including items like device setup and individual configuration pages for all available devices. Usually, you won't need to change this file.
- **_imports.razor –** Common Blazor using statements that should be available on all razor pages
- **AlpacaConfiguration.cs** – Defines configuration values for the Alpaca device itself e.g. IP port, manufacturer name, server version and location that will be loaded from and persisted to permanent storage. Define device specific configuration values in ServerSettings.cs rather than in AlpacaConfiguration.cs.
- **App.razor** – The root component of the application. Usually, you won't need to change this file.
- **Appsettings.json** - The appsettings.json file in ASP.NET Core is a centralized location for storing fixed application configuration settings such as database connection strings, API keys, and other environment-specific values. Usually, you won't need to change this file.
- **License.md** – The copyright license file for your device. The MIT license is provided by default.
- **NuGet.config** – Specifies sources of NuGet updates. By default, this includes the main NuGet repository and the ASCOM MyGet repository that holds updates ahead of public release to NuGet.
- **Program.cs** – Entry point for the application, see comments in file for further information.

- **ServerSettings.cs** – Class that defines common server level and device specific configuration values. Contents of this class are loaded automatically when the application starts and persisted by the **Save** buttons in the setup dialogue.

Appendix 2 – Example Razor setup page

```
@page "/setup/v1/Telescope/{InstanceID:int}/setup"

<h3>Telescope Setup: @InstanceID</h3>

@* Check whether this is a defined device and if so display the UI *@
@if (ASCOM.Alpaca.DeviceManager.Telescopes.ContainsKey(InstanceID))
{
    @* Create a group of configurable values *@
    <fieldset style="padding-left:12px">

        @* Example text box *@
        <label style="margin-top:12px">Example string: </label>
        <input type="text" @bind="ExampleString" >
        <br />

        @* Example check box *@
        <label style ="margin-top:12px">Example checkbox:</label>
        <input type="checkbox" @bind="ExampleBool">
        <br />

        @* Example numeric input *@
        <label style="margin-top:12px">Example int:</label>
        <input type="number" @bind="ExampleInt" min="1" max="65535" style="width:20ch;">
        <br />

        @* Display a Save button and status label that colours green for success messages *@
        @* and red for failure messages *@
        <button @onclick="SaveDriverSettings" style="min-width:12ch;margin:12px">Save</button>
        <label style="color:@StatusColour;"><b>@Status</b></label>

    </fieldset>
}
else
{
    @* Not a defined device so create an error message *@
    <h3>This device does not exist and cannot be configured</h3>
}

@* Start of C# code *@
@code {

    /// <summary>
    /// Parameter visible to clients calling this class
    /// </summary>
    [Parameter]
    public int InstanceID { get; set; }

    /// <summary>
    /// Example boolean setting that defaults to the value retrieved from the ServerSettings class
    /// </summary>
    private bool ExampleBool { get; set; } = ServerSettings.ExampleBool;

    /// <summary>
    /// Example integer setting that defaults to the value retrieved from the ServerSettings class
    /// </summary>
    private int ExampleInt { get; set; } = ServerSettings.ExampleInt;

    /// <summary>
    /// Example string setting that defaults to the value retrieved from the ServerSettings class
    /// </summary>
    private string ExampleString { get; set; } = ServerSettings.ExampleString;

    /// <summary>
    /// Current value of the save status message
    /// </summary>
    private string Status { get; set; } = "";

    /// <summary>
    /// Colour of the status message - green for success, red for failure
    /// </summary>
    private string StatusColour { get; set; } = "green";
```

```
    /// <summary>
    /// Save the driver settings and display a message indicating success or failure
    /// </summary>
    /// <remarks>
    /// Note use of the async modifier to ensure that this method does not block the UI thread
    /// and prevent UI updates appearing
    /// </remarks>
    private async void SaveDriverSettings()
    {
        try
        {
            // Save the settings to the ServerSettings class
            ServerSettings.ExampleBool = ExampleBool;
            ServerSettings.ExampleInt = ExampleInt;
            ServerSettings.ExampleString = ExampleString;

            // Display a success message
            Status = "Driver Settings Saved OK";
            StatusColour = "green";

            // Notify the application that the UI's Status field has changed so the UI will be updated
            StateHasChanged();

            // Wait for 2 seconds and then clear the status message.
            // Note that this is done asynchronously to ensure the UI remains responsive
            await Task.Run(async () =>
            {
                await Task.Delay(2000);
                Status = "";
            });
        }
        catch (Exception ex)
        {
            Status = $"Settings not saved: {ex.Message}";
            StatusColour = "red";
        }

        // Notify the application that the UI's Status field has changed so the UI will be updated
        StateHasChanged();
    }
}
```

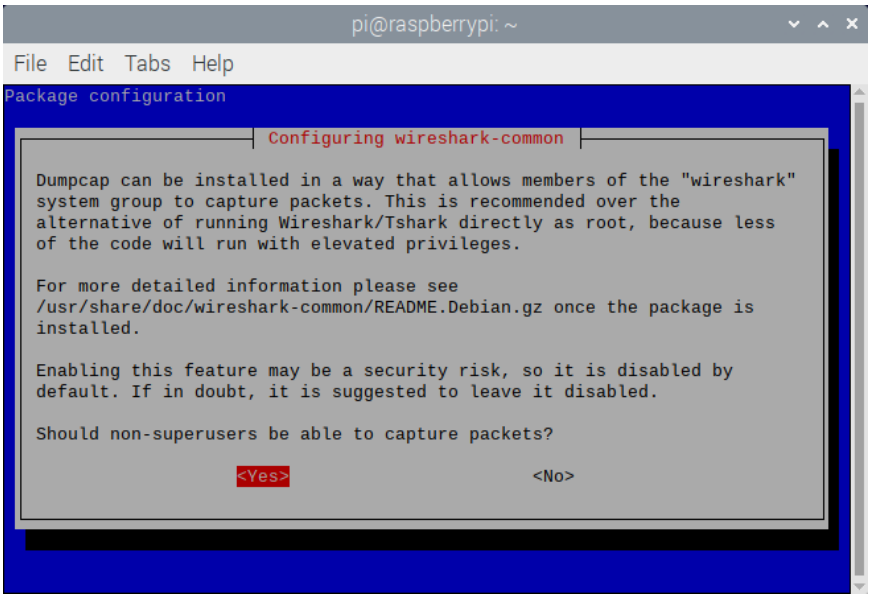
Appendix 3 – Installing and using WireShark

Installing Wireshark and Setting Privileges on a Raspberry Pi

To install it on the Pi, you need about 100MB. In a shell

```
pi@raspberrypi:~ $sudo apt install wireshark
```

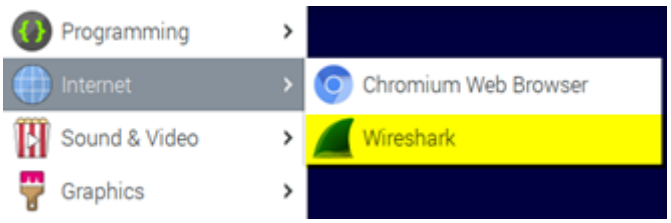
During installation you'll see this



Make sure to answer `<Yes>` to this so you don't have to start Wireshark with root privs. But there is more, note it says you still need to be a member of the "wireshark" group. Once the installation completes, add yourself (typically you are user "pi"):"

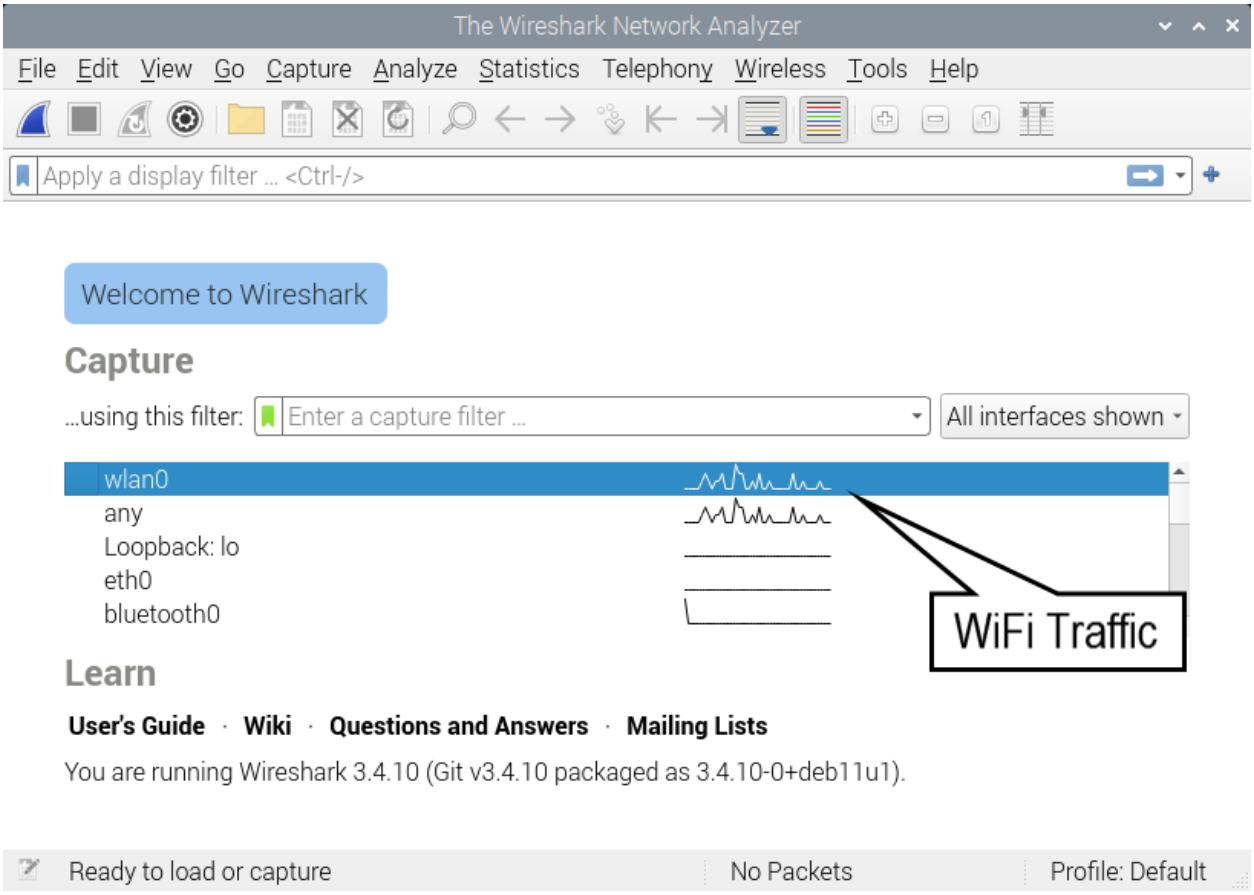
```
pi@raspberrypi:~ $sudo usermod -a -G wireshark pi
```

Now log out and back in to join the "wireshark" group. You will find Wireshark in the Berry menu under Internet.



Checking the Wireshark Installation on the Raspberry Pi

Now start Wireshark and be sure you see the network interfaces, indicating that you have successfully allowed non-root capturing and joined the "wireshark" group. You should see this:



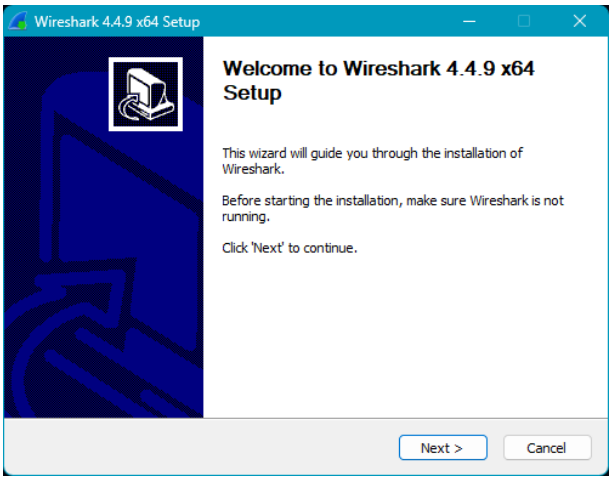
If you see wlan0 and traffic, then you're ready to use Wireshark. If you are on Ethernet, the traffic will be on eth0. Otherwise:

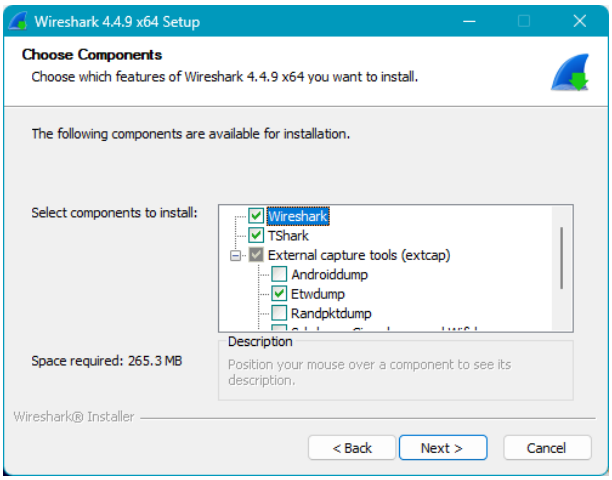
1. Did you answer Yes to the non-root capture? You can check by entering this command
   `pi@raspberrypi:~ $sudo dpkg-reconfigure wireshark-common`
   which will show the allow non-root dialog that appeared during installation. Answer `<Yes>`.
2. Did you log out and back in after adding yourself to the "wireshark" group?

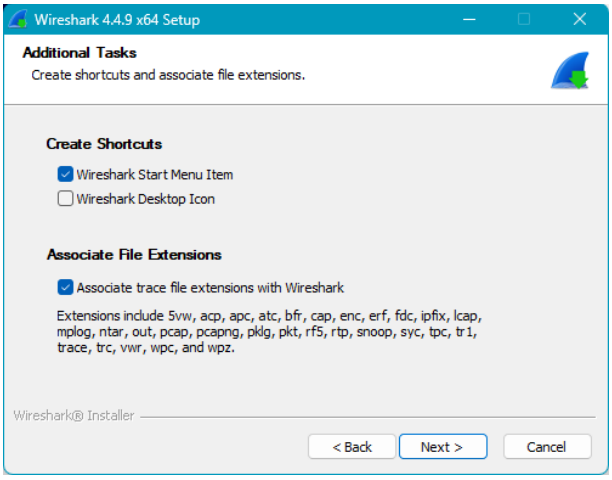
   Go back and repeat the installation steps till you see a Wireshark window with the physical interfaces as shown above.

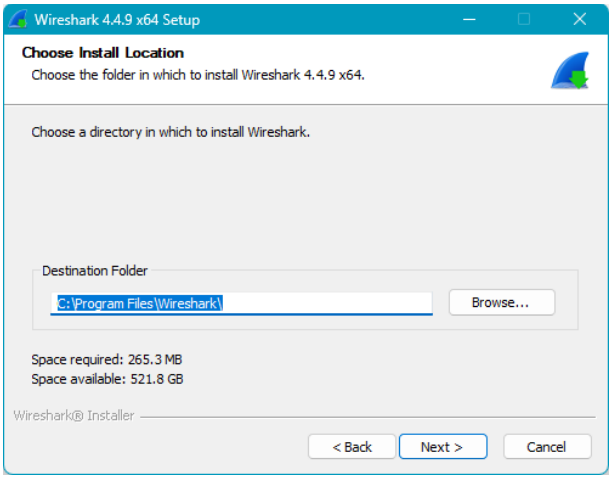
Installing WireShark on Windows

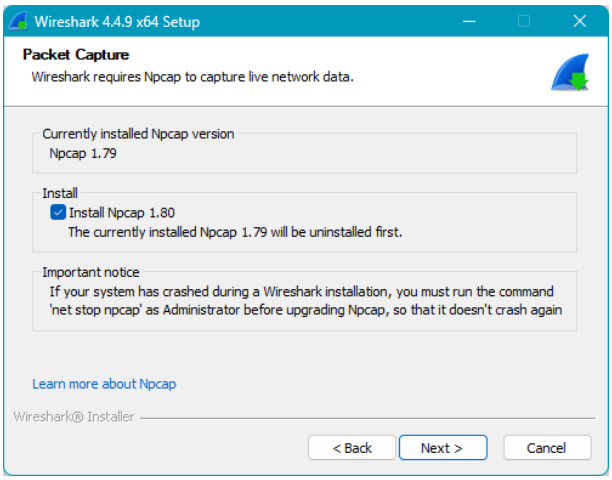
To install WireShark on Windows, download the latest release from https://www.wireshark.org/ and run the installer.

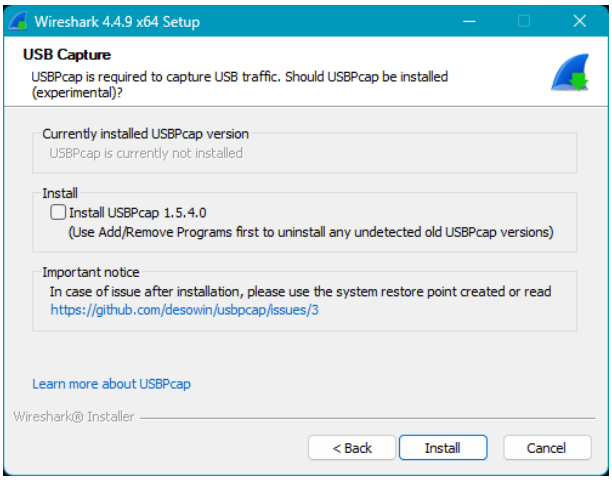


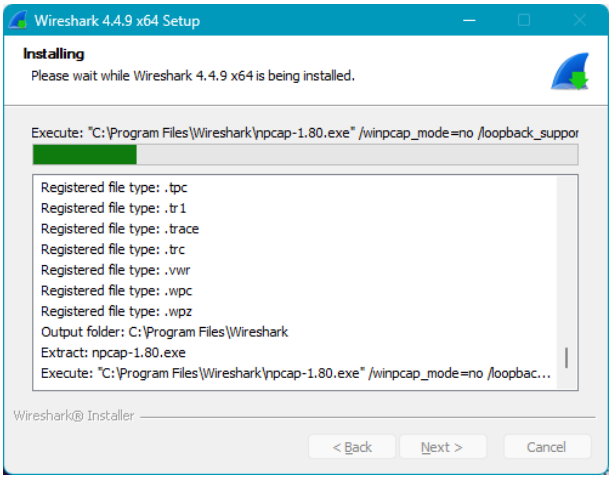


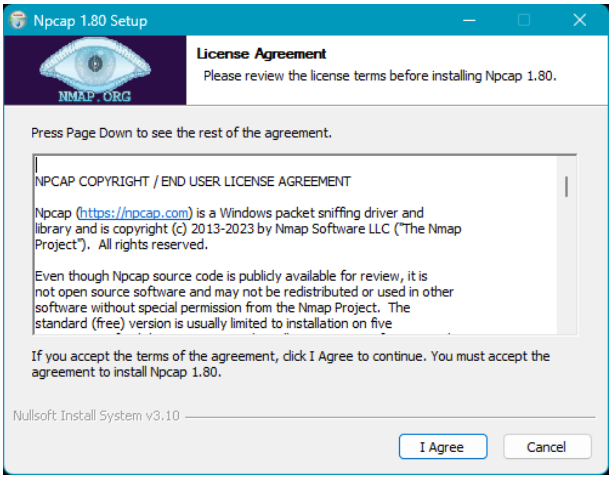


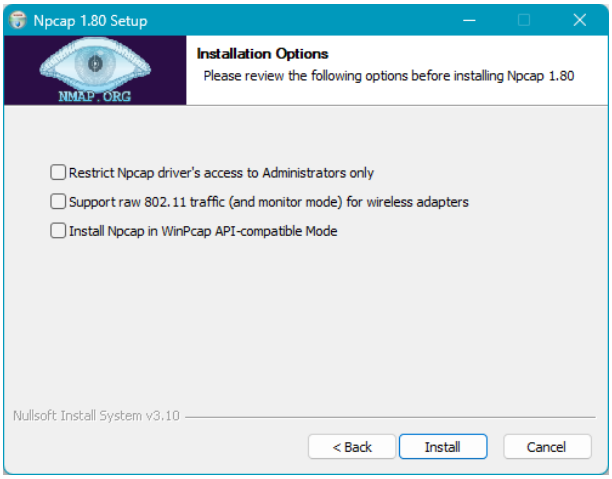


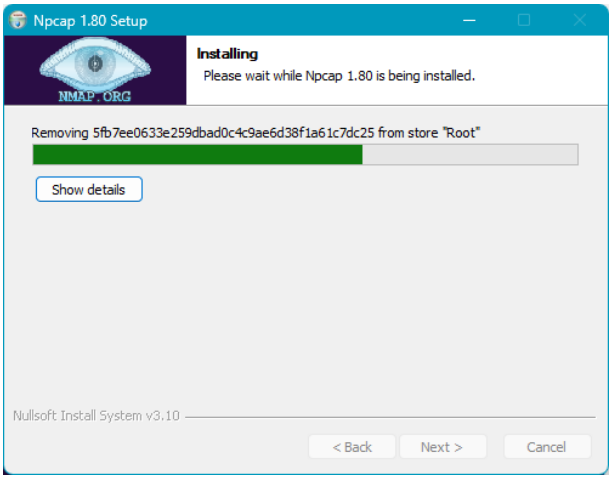


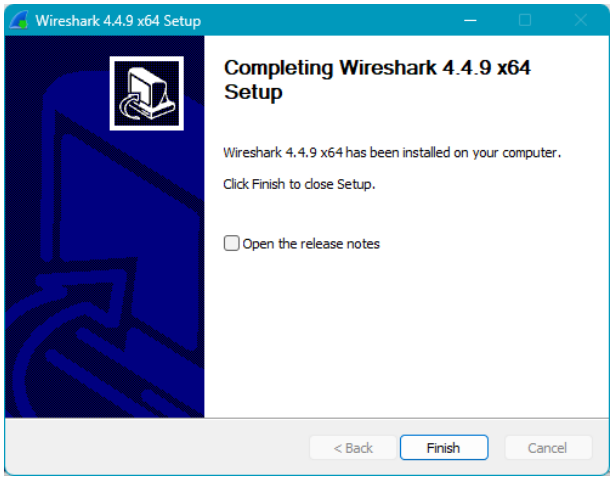



Wireshark 4.4.9 x64 Setup
Installing
Please wait while Wireshark 4.4.9 x64 is being installed.
Execute: "C:\Program Files\Wireshark\npcap-1.80.exe" /winpcap_mode=no /loopback_suppor
Registered file type: .tpc
Registered file type: .tr1
Registered file type: .trace
Registered file type: .trc
Registered file type: .vwr
Registered file type: .wpc
Registered file type: .wpz
Output folder: C:\Program Files\Wireshark
Extract: npcap-1.80.exe
Execute: "C:\Program Files\Wireshark\npcap-1.80.exe" /winpcap_mode=no /loopbac...
Wireshark® Installer
< Back
Next >
Cancel


Npcap 1.80 Setup
License Agreement
Please review the license terms before installing Npcap 1.80.
NMAP.ORG
Press Page Down to see the rest of the agreement.
NPCAP COPYRIGHT / END USER LICENSE AGREEMENT
Npcap (https://npcap.com) is a Windows packet sniffing driver and
library and is copyright (c) 2013-2023 by Nmap Software LLC ("The Nmap
Project"). All rights reserved.
Even though Npcap source code is publicly available for review, it is
not open source software and may not be redistributed or used in other
software without special permission from the Nmap Project. The
standard (free) version is usually limited to installation on five
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Npcap 1.80.
Nullsoft Install System v3.10
I Agree
Cancel

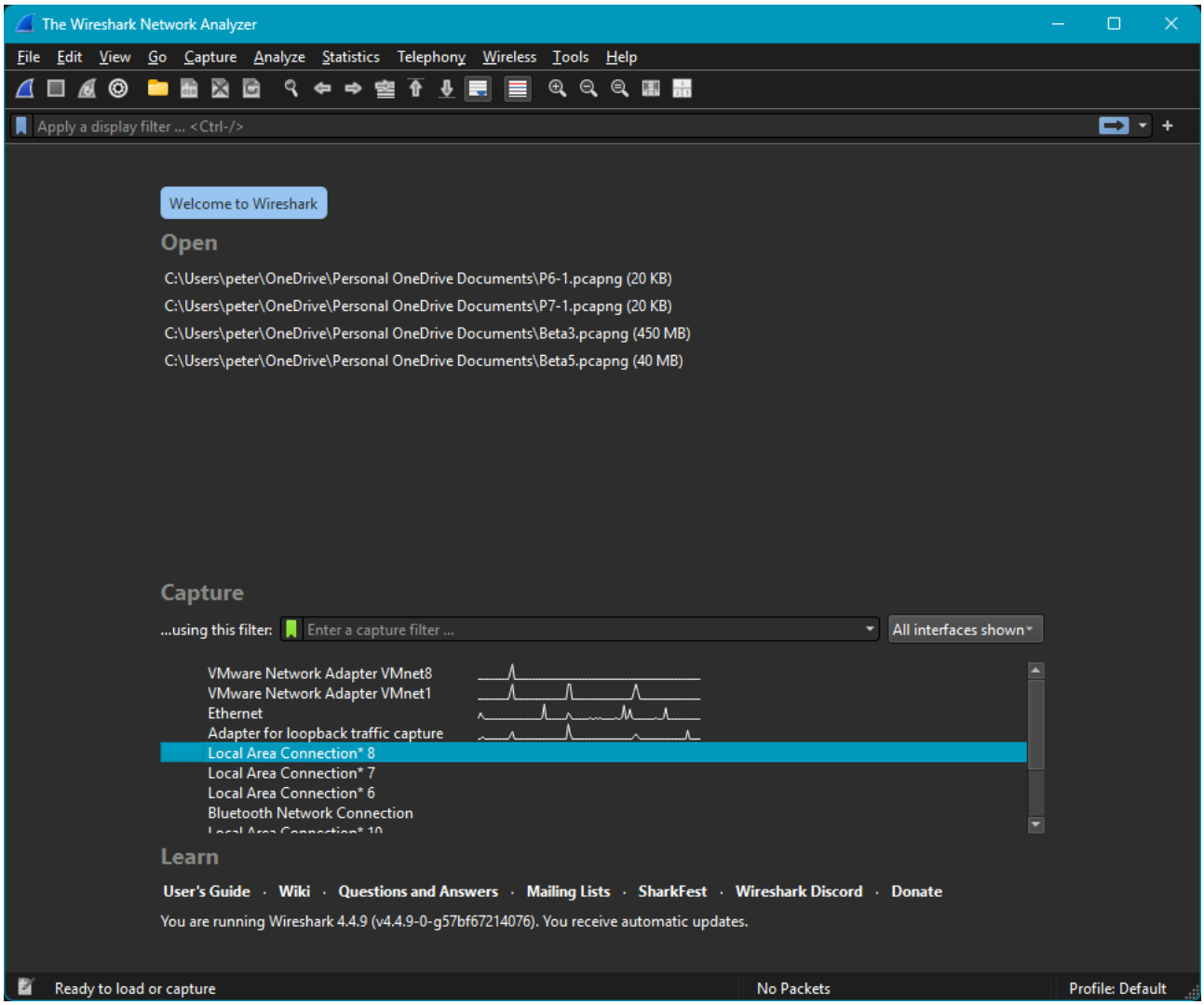

Npcap 1.80 Setup
Installation Options
Please review the following options before installing Npcap 1.80
NMAP.ORG
Restrict Npcap driver's access to Administrators only
Support raw 802.11 traffic (and monitor mode) for wireless adapters
Install Npcap in WinPcap API-compatible Mode
Nullsoft Install System v3.10
< Back
Install
Cancel


Npcap 1.80 Setup
Installing
Please wait while Npcap 1.80 is being installed.
NMAP.ORG
Removing 5fb7ee0633e259dbad0c4c9ae6d38f1a61c7dc25 from store "Root"
Show details
Nullsoft Install System v3.10
< Back
Next >
Cancel


Wireshark 4.4.9 x64 Setup
Completing Wireshark 4.4.9 x64 Setup
Wireshark 4.4.9 x64 has been installed on your computer.
Click Finish to close Setup.
Open the release notes
< Back
Finish
Cancel

Checking the Wireshark Installation on Windows

Start Wireshark and be sure you see the network interfaces, indicating that you have successfully installed WireShark. You should see something like this:



Setting up a Test & Learning Environment

You will need a client application and an Alpaca device for this. A range of clients are available including:

- Windows DeviceHub (via an Alpaca dynamic driver),
- SkySafari Plus or Pro on iOS
- Cartes du Ciel on any supported OS platform
- Conform Universal on any supported OS platform: https://github.com/ASCOMInitiative/ConformU/releases

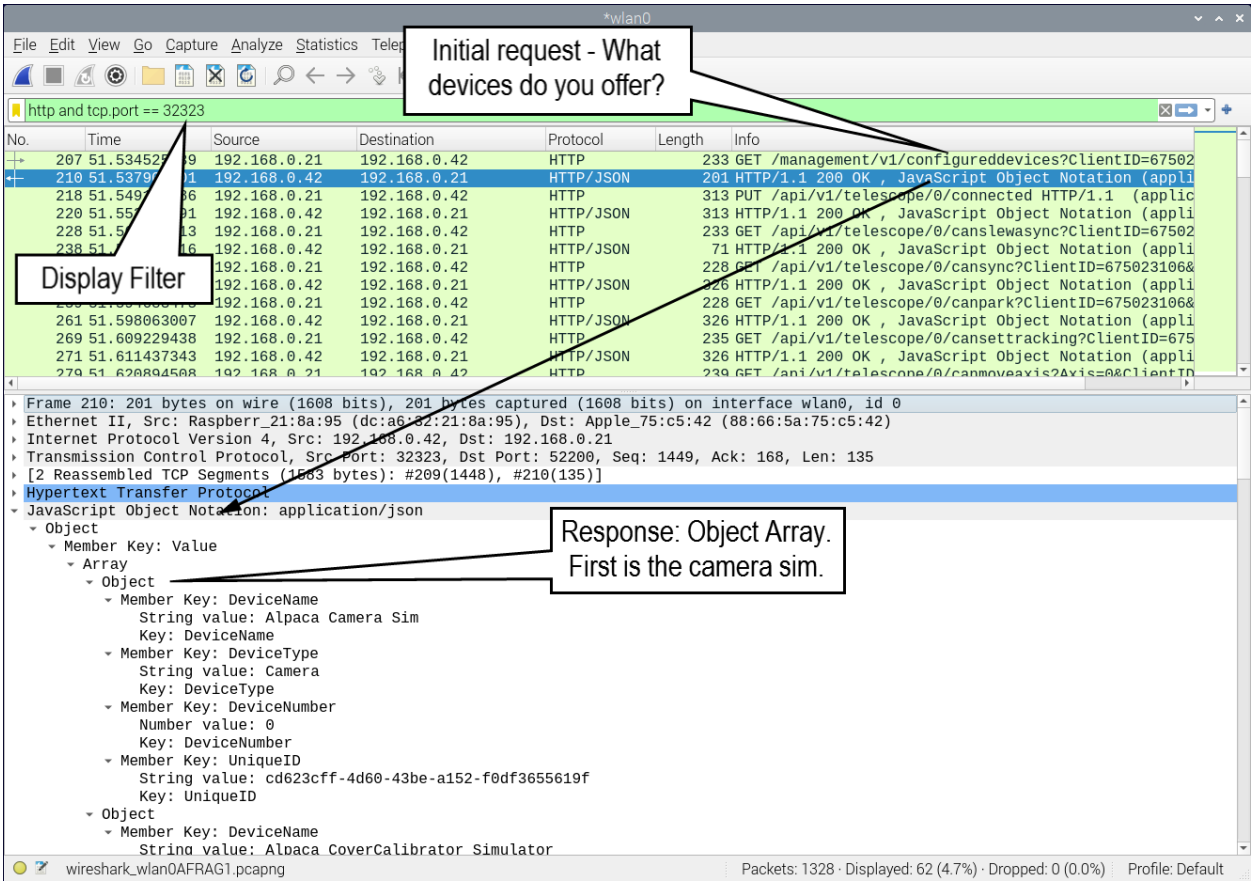
The OmniSim is the simplest Alpaca device to get going:

- Windows – If you are using Platform 7 the Omni-Simulators are installed with the Platform.
- Windows – If you are using Platform 6 install the latest production Omni-Simulators Windows release from here: https://github.com/ASCOMInitiative/ASCOM.Alpaca.Simulators/releases
- All other OS - Install the latest production Omni-Simulators release for your OS from here: https://github.com/ASCOMInitiative/ASCOM.Alpaca.Simulators/releases

Once you have an app (anywhere) talking to the simulated telescope on the Pi, you can use Wireshark to see the HTTP/REST traffic between them. Here we use SkySafari Pro on 192.168.0.21, and we're running Wireshark on a Pi that's also running the OmniSimulator, on 192.168.0.42.

Have a look at this packet capture of SkySafari's initial connect to the Telescope. The key to setting this up is the display filter, which limits the display to HTTP and port 32323 (the OmniSim's Alpaca port).

If you make a mistake in the display filter the background will turn reddish. Without the display filter you will see a lot of uninteresting (for our purposes) trash.



Look at the list of REST transactions. The first gets the list of devices as shown. Next you see a PUT of **true** to the telescope's **connected** endpoint, and this succeeds. Then it GETs some capability properties: **canslewasync**, **cansync**, **canpark**, etc.

It's beyond the scope of this document to be a tutorial on Wireshark. There are loads of videos on YouTube covering Wireshark. And there's always the PDF Wireshark User Manual.

More information on the Alpaca API can be found in the Alpaca API Reference document that can be downloaded from here: https://ascom-standards.org/AlpacaDeveloper/Index.htm

Appendix 4 – Document Change History

Version 1

- Original release

Version 2 – October 2025

- Changed the Drivers folder name to DeviceAccess to match the revised template.