ASCOM ALPACA

Concepts and Implementation

Abstract

In 2000, ASCOM brought to astronomy software on the Windows platform a standard way to communicate between programs and devices such as mounts, cameras, etc. The obvious next step is to extend ASCOM into the realm of distributed systems on multiple OS platforms while maintaining its language independence. The core element of any distributed system is its interface (API) design. While ASCOMs APIs are today mostly implemented through Windows/COM technology, their real value lies in their maturity, stability, and proven suitability to control most types of astronomical instruments.

In this paper we describe a new technology, ASCOM Alpaca, which specifies these proven abstract APIs using the Representational State Transfer (REST) concept and TCP/IP communication. We also present the ASCOM Remote set of tools, available today, that enable new REST based clients and drivers to interoperate with the significant existing community of COM based applications and drivers.

The new tool set proves that the design and implementation work in the real world, and for the first time, removes the limitation that ASCOM clients and drivers must be installed on the same PC.

Peter Simpson, Bob Denny

peter@peterandjill.co.uk, rdenny@dc3.com

Contents

Figures

1. ASCOM Alpaca

1.1 Background and Overview

Prior to 2001, the astronomy marketplace was shackled because application developers had to develop costly integration code for every hardware device and device manufacturers were limited in their growth by whether leading applications had support for their hardware.

In 2001, a set of APIs[1] called ASCOM[2] was published and implemented using cross-language COM services on the Windows OS. These APIs provide vendor neutral views of each hardware device type and ensure that application authors and hardware vendors only need to write code once and that applications will work with all hardware and that hardware devices will work with all applications.

The standardized APIs provide access to astronomical instruments in a way that masks the individual uniqueness of each instrument (e.g. telescope mounts, cameras, focusers, etc.). Thus, applications can freely use many different variations of a device type with no special understanding or program logic for the various device types out there. For example, planetariums can steer telescopes via the ASCOM Telescope API without any specific knowledge of the mount's operation.

The ASCOM APIs have opened up the astronomy software community to innovation and competition, reduced support cost, reduced time to market and increased market coverage for both application authors and hardware vendors.

For many years the ASCOM interfaces have proven themselves to be robust, mature, stable and suitable to the mission and have provided application developers and device makers with interoperation with little need for revision or expansion. The fundamentals are clearly solid and the ASCOM APIs mitigate the risk of low / slow adoption by other developers and suppliers that inevitably comes with inventing new unique interfaces.

1.2 ASCOM APIs in RESTful Form – Alpaca APIs

Application and device manufacturers see a need to expand beyond Windows to embrace platforms such as Mac OS, Android and Linux. In response, the ASCOM Initiative has redefined the ASCOM standard APIs using modern industry-accepted standards, namely OpenAPI[3] and REST[4] with JSON[5] payloads. These APIs are collectively known as ASCOM Alpaca and enable:

- Client applications on Mac and Linux to take advantage of the mature, stable, and already suitable ASCOM APIs.
- Creation of self-contained devices that directly present ASCOM standard APIs over Ethernet or Wi-Fi removing the need for a host operating system such as Windows or MacOS.
- Hosting of drivers and engineering tools on MacOS or Linux if desired.

The ASCOM abstract APIs are mature, stable, and enable astronomy software and device makers to eliminate a large part of their design risk in transitioning to the Mac OS and Linux platforms and of creating distributed networked systems.

[1] Application Programming Interfaces (APIs)
[2] Astronomy Common Object Model (ASCOM)
[3] OpenAPI Initiative
[4] Representational State Transfer (REST)
[5] Javascript Object Notation (REST context)

Originally the ASCOM APIs were only available on Windows through Microsoft's COM technology and enabled clients to use devices attached to the same PC.

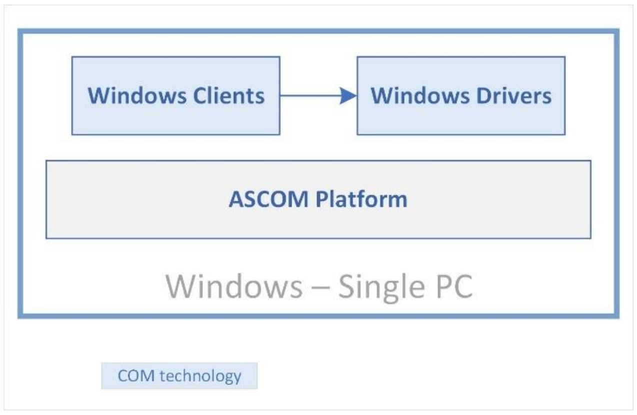


Figure 1 - Original ASCOM Configuration

ASCOM Alpaca enables heterogenous configurations utilising any hardware and operating system, so long as they are network capable.

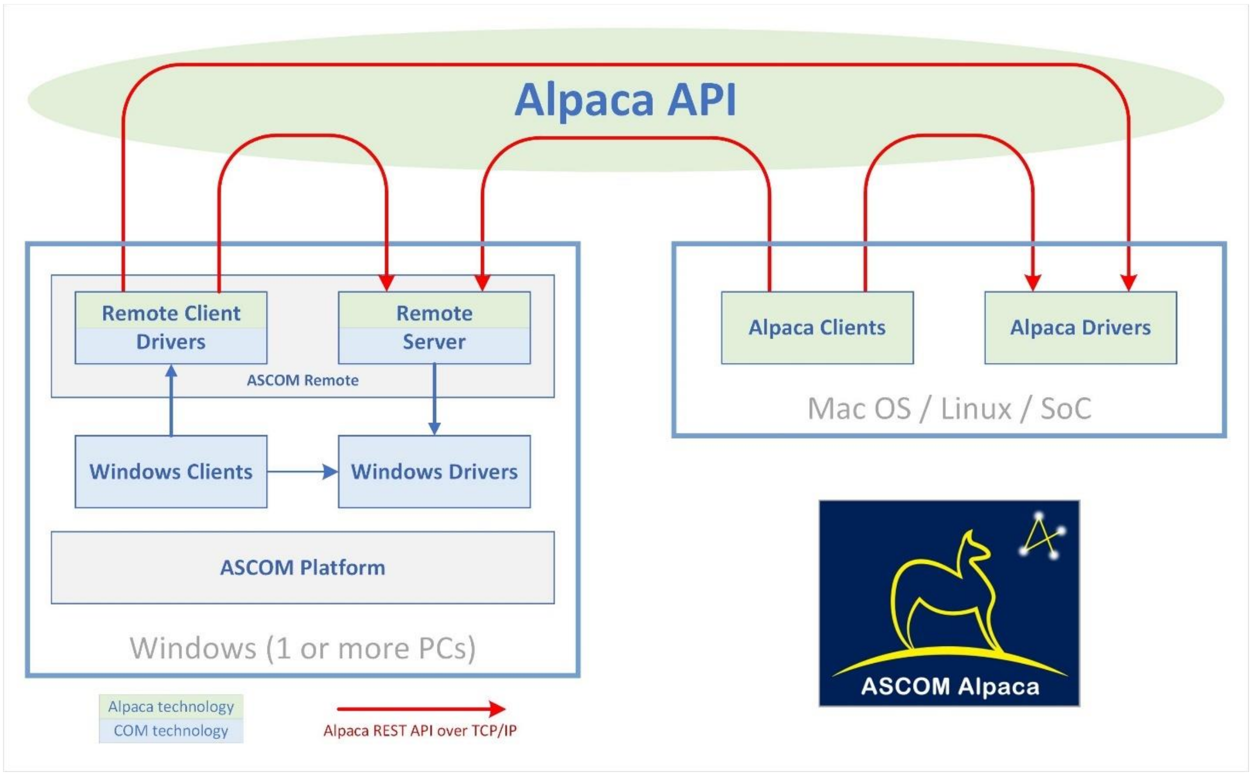


Figure 2 – Heterogenous configurations possible through ASCOM Alpaca

1.3 Transition and Migration

New applications and devices can be coded from the outset to use the new REST based ASCOM Alpaca APIs, however, they will be of limited value if they can only communicate with other Alpaca based applications and drivers. What is needed is transition technology to enable Alpaca applications and drivers to communicate with the huge existing base of Windows COM based applications and drivers to provide time to make the switch without disadvantaging consumer or commercial interests.

ASCOM principal developer Peter Simpson has developed, and made available Open Source, a complete set of tools that allow transparent operation of today's COM-based applications and drivers across TCP/IP networks. Any COM-based ASCOM driver can now accessed by a COM-based ASCOM client program anywhere in the world, over the internet, using the new Alpaca APIs. Section 3 of this document describes this set of tools, called **ASCOM Remote**.

ASCOM Remote not only provides a useful remoting service to today's COM based ASCOM clients and drivers, it also *proves* that the ASCOM Alpaca implementation faithfully implements the standardized ASCOM abstract interfaces and API. Finally, ASCOM Remote provides both a reference implementation of the new ASCOM Alpaca APIs, and a smooth path to migration for customers and manufacturers.

Through ASCOM Remote it is possible *today* to build a focuser that implements the ASCOM Alpaca Focuser API , perhaps based on a micro-board processor such as the Raspberry Pi or Arduino, and it will instantly be compatible with *today's* FocusMax, Optec FocusLock, MaxIm DL AutoFocus, AstroPhotoTool, CCDAutoFocus (Images Plus), Sequence Generator Pro, and ScopeFocus-ASCOM focusing software!

2. The ASCOM Alpaca API Specification

The Alpaca API expresses the proven ASCOM APIs in platform-neutral and language independent forms as REST resources using established technical standards such as TCP/IP and JSON. Exploitation can now be be ubiquitous because there are REST implementation libraries for virtually every language and platform.

The Alpaca APIs are functionally identical to the COM-based ASCOM APIs while adding cross-platform and distributed operation to the already powerful cross-language features and exploiting the mature, proven, ASCOM device APIs.

2.1 ASCOM Alpaca API Documentation

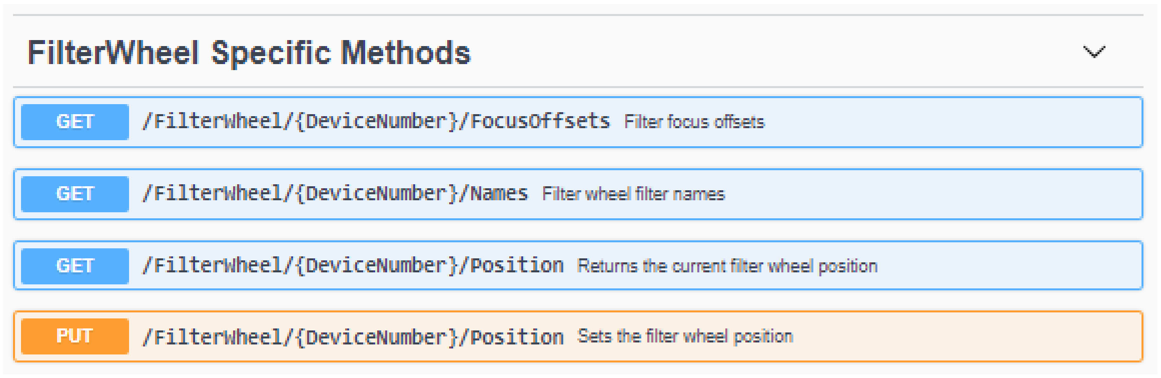
The ASCOM Alpaca APIs are documented using the Swagger OpenAPI toolset and are available through a URL on the ASCOM Standards web site. The ASCOM API is fully documented here:

https://www.ascom-standards.org/api

To start exploring go to the above API URL and click a grey Show/Hide link to expand one of the sets of methods and then click the blue GET or orange PUT methods for detailed information on that API call. Note that this documentation, along with the companion ASCOM Remote Server Management API is documented in the next section.

Anyone who is familiar with the ASCOM COM based APIs will immediately feel at home with the functionality available through ASCOM Alpaca. Full details on how to construct, send and decode ASCOM Alpaca API calls are given in the companion ***ASCOM Alpaca API Reference*** document.

Here is an example of the top level documentation view of the methods unique to the ASCOM Filter Wheel API:



FilterWheel Specific Methods

GET /FilterWheel/{DeviceNumber}/FocusOffsets Filter focus offsets

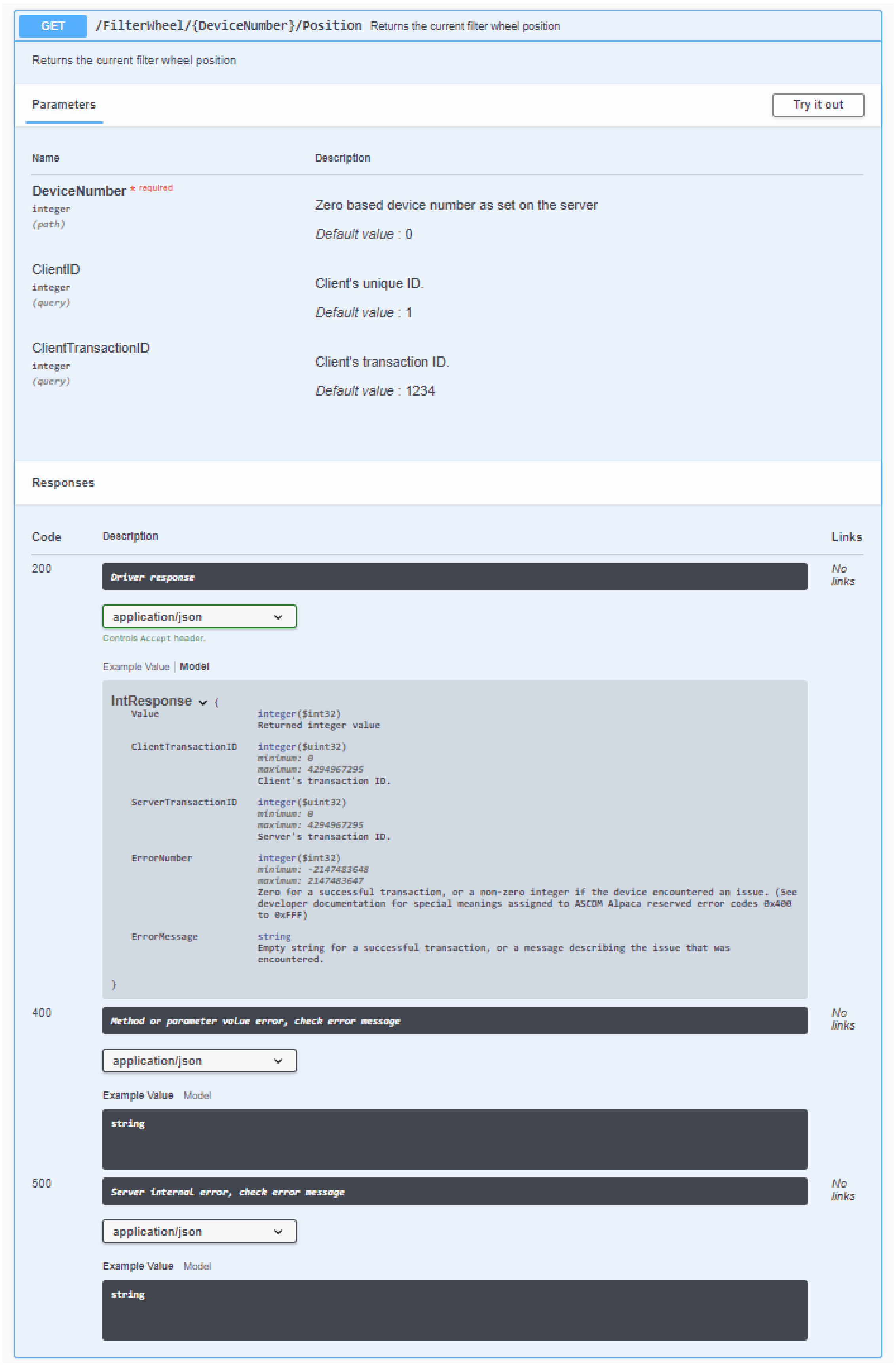
GET /FilterWheel/{DeviceNumber}/Names Filter wheel filter names

GET /FilterWheel/{DeviceNumber}/Position Returns the current filter wheel position

PUT /FilterWheel/{DeviceNumber}/Position Sets the filter wheel position

Figure 3 – Filter wheel API Swagger Documentation - Top level

The next figure shows the detail available to developers who wish to develop Alpaca devices and drivers.



GET `/FilterWheel/{DeviceNumber}/Position` Returns the current filter wheel position

Returns the current filter wheel position

Parameters

Try it out

Name	Description
DeviceNumber * required integer *(path)*	Zero based device number as set on the server *Default value* : 0
ClientID integer *(query)*	Client's unique ID. *Default value* : 1
ClientTransactionID integer *(query)*	Client's transaction ID. *Default value* : 1234

Responses

Code	Description	Links
200	*Driver response*	*No links*

application/json

Controls Accept header.

Example Value | Model

```
IntResponse {
    Value                  integer($int32)
                           Returned integer value

    ClientTransactionID    integer($uint32)
                           minimum: 0
                           maximum: 4294967295
                           Client's transaction ID.

    ServerTransactionID    integer($uint32)
                           minimum: 0
                           maximum: 4294967295
                           Server's transaction ID.

    ErrorNumber            integer($int32)
                           minimum: -2147483648
                           maximum: 2147483647
                           Zero for a successful transaction, or a non-zero integer if the device encountered an issue. (See
                           developer documentation for special meanings assigned to ASCOM Alpaca reserved error codes 0x400
                           to 0xFFF)

    ErrorMessage           string
                           Empty string for a successful transaction, or a message describing the issue that was
                           encountered.
}
```

Code	Description	Links
400	*Method or parameter value error, check error message*	*No links*

application/json

Example Value Model

```
string
```

Code	Description	Links
500	*Server internal error, check error message*	*No links*

application/json

Example Value Model

```
string
```

Figure 4 – Filter wheel API Swagger Documentation – FilterWheel.Position detail

3. ASCOM Remote

3.1 Concept and Capabilities

ASCOM Remote uses the ASCOM Alpaca APIs to deliver three capabilities for networked devices:

1. ASCOM Windows applications and drivers can, for the first time, run on separate PCs
2. Applications on any platform can use the ASCOM Alpaca APIs directly, without any COM technology, to communicate with all of today's COM based ASCOM Windows drivers.
3. Today's ASCOM Windows applications can communicate with new drivers that use the ASCOM Alpaca APIs directly without using any COM technology.

ASCOM Remote comprises two components:

1. A set of ***Remote Clients*** that appear as local drivers to ASCOM Windows applications. These receive COM based commands from the local application and translate them into ASCOM Alpaca calls, which are sent to networked devices that expose the ASCOM Alpaca API.
2. A ***Remote Server*** that can host up to 10 COM based ASCOM drivers and present these through the ASCOM Alpaca interfaces.

This graphic shows a COM client using a *Remote Client* driver to access COM and native drivers hosted on remote devices:

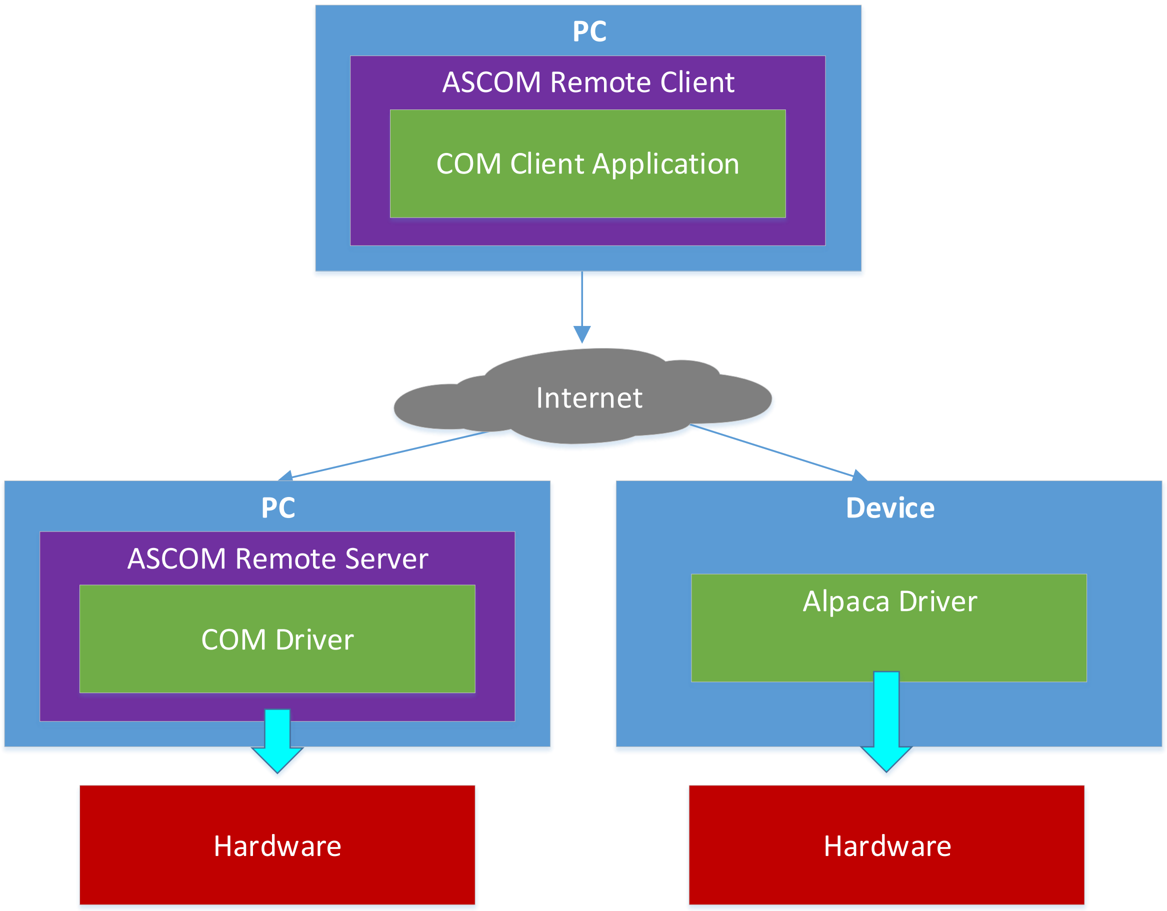


Figure 5 - COM Client accessing a remote COM or non-COM driver

This graphic shows non-Windows applications using current COM based ASCOM drivers hosted by the *Remote Server*:

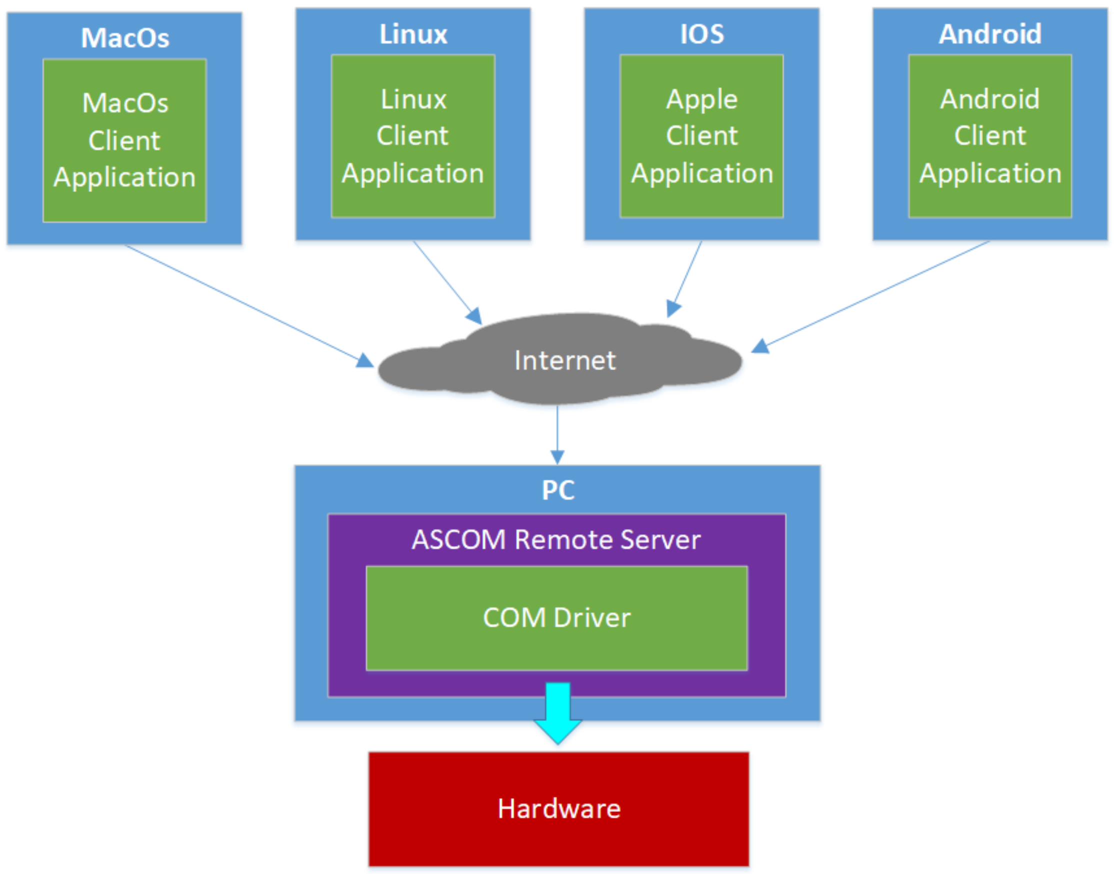


Figure 6 - Non-Windows applications accessing a remote COM based driver

3.1.1 ASCOM Remote Client

This appears to client applications as a driver of the required device type e.g. Telescope. This component translates the client's COM requests into the ASCOM Alpaca API standard, handles authentication, encryption and transport of the command to the remote host that houses the remote driver.

The remote client can be configured through its driver Setup Dialogue with required access credentials and the remote device's URL or IP address and port.

3.1.2 ASCOM Remote Server

This component exposes an IP end point and translates incoming ASCOM Alpaca API requests back to COM requests before passing them to the configured drivers installed on that machine. The following figure shows a Windows COM application using a Remote Client driver to talk with a Remote Server that is hosting a Windows COM driver running on a remote PC.

The appendix in section 4 explains the difference between in-process and out of process COM drivers.

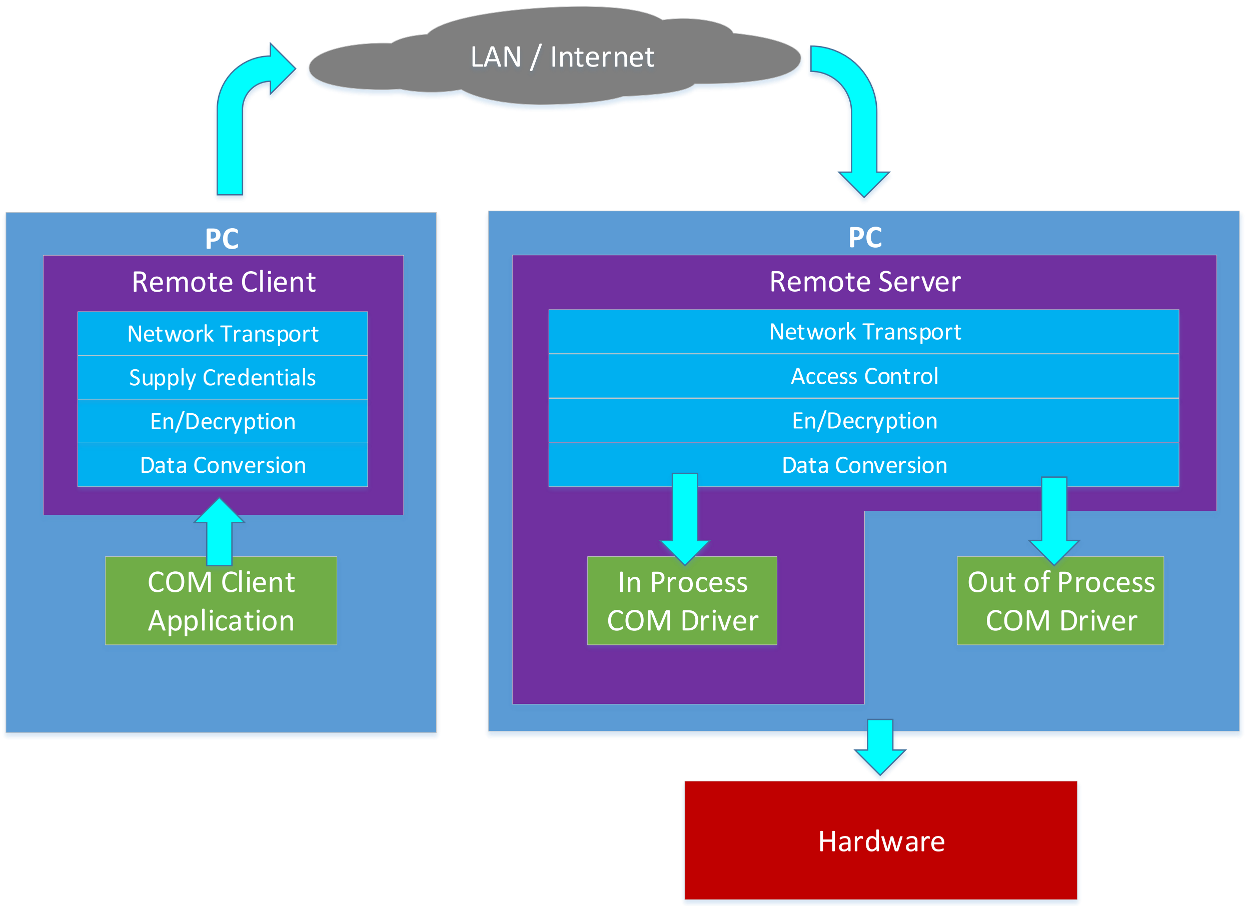


Figure 7 – ASCOM Remote facilitates a COM client to use a driver running on a remote PC

3.2 Security Concerns

You should think carefully before exposing observatory devices over the Internet. Network and device security are critical and effective solutions will need additional components.

The Remote Client and Remote Server applications support basic HTTP authentication and the Remote Client can initiate HTTPS connections, however, the Remote Server does not support HTTPS by design.

> ***For use over the Internet:*** *It is not recommended that the ASCOM Remote Server is connected directly to the Internet because it is not hardened for this use.*
>
> *Instead* ***front the Remote Server with a firewall and web server proxy / DMZ*** *that will isolate your internal LAN from Internet threats. The web server can also terminate incoming SSL connections, handle complex authentication if required and forward permitted HTTP connections to the ASCOM Remote server.*

This approach has been adopted because creating a device that can withstand the hostile environment of the Internet is a major undertaking and this capability is already provided by well-maintained components including firewalls and web servers such as Apache and NGINX.

3.3 Example – COM Application uses COM Driver on Local Network

This figure shows an internal LAN use case, where data does not need to travel over the Internet or be protected in any way. The client application runs on one PC and accesses a driver hosted within the ASCOM Remote Server running on another PC. This remote PC could even be one of the new class of mini PCs that are small enough and light enough to be fitted directly to a Telescope mount.

In this configuration, HTTP is used on the trusted internal network.

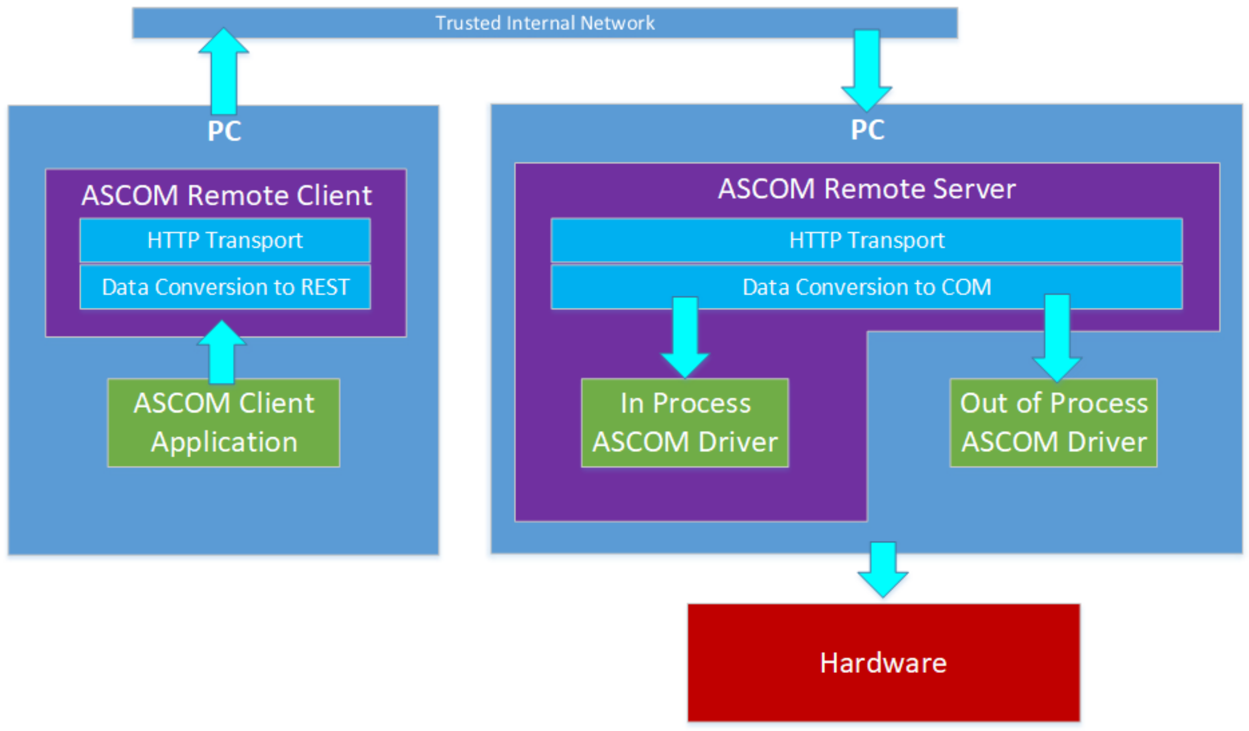


Figure 8 - COM application using a COM driver on the local network

3.4 Example - COM Application uses Alpaca Device on Local Network

This figure shows one of today's ASCOM compliant applications such as TheSkyX, ACP, Maxim DL or SGP using an Alpaca device over the network through the Remote Client driver.

In this scenario the Remote Server isn't required, and the Alpaca device will likely be a network enabled micro-controller or Linux based device such as the Raspberry Pi. Neither COM nor Windows are required on the Alpaca device.

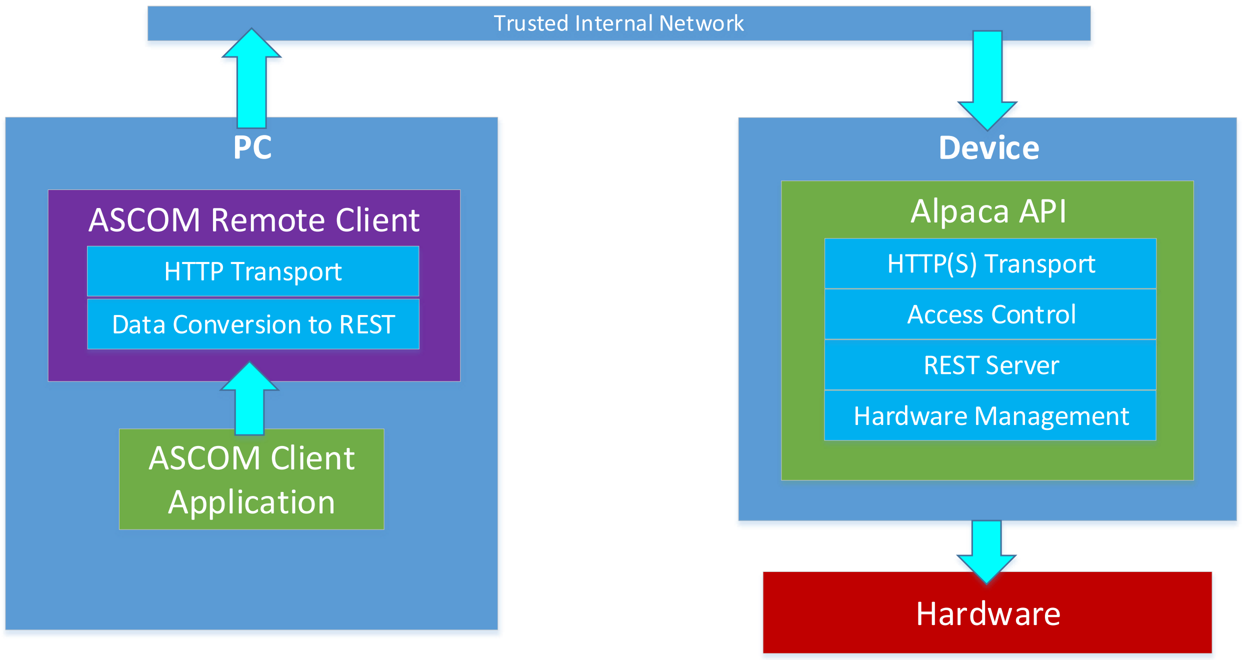


Figure 9 - COM application using an Alpaca device on the local network

3.5 Example – COM Application uses COM Driver on Remote PC

Use of HTTPS is considered good practice for communication over the Internet and, the recommended approach is to use a web server such as Apache or NGINX ahead of the ASCOM Remote Server to handle secure session termination and to proxy requests to the ASCOM Remote Server instance running on a local trusted network.

The web server instance will also support client authentication as well as handling the complex, ever evolving, SSL/TLS algorithms.

This diagram shows a configuration where remote devices are accessed over the Internet from a COM client application:

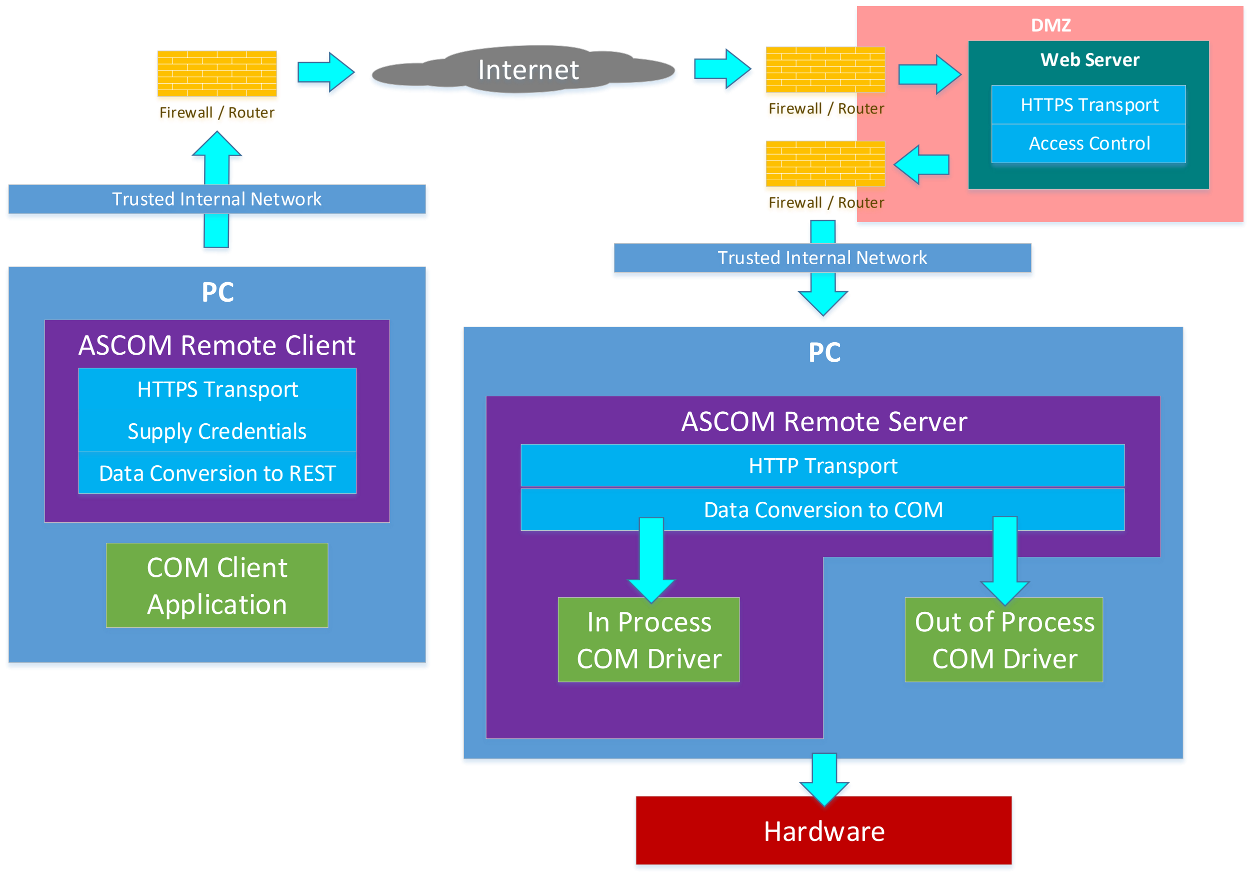


Figure 10 - COM application using a COM driver on a remote PC over the Internet

3.6 Example – COM Application uses Remote Alpaca Device

A key objective of ASCOM Alpaca is to enable new drivers to be developed that have no COM dependency and that communicate exclusively through the ASCOM Alpaca API.

This diagram illustrates the approach where a Windows COM client needs to communicate with a new networked Alpaca device:

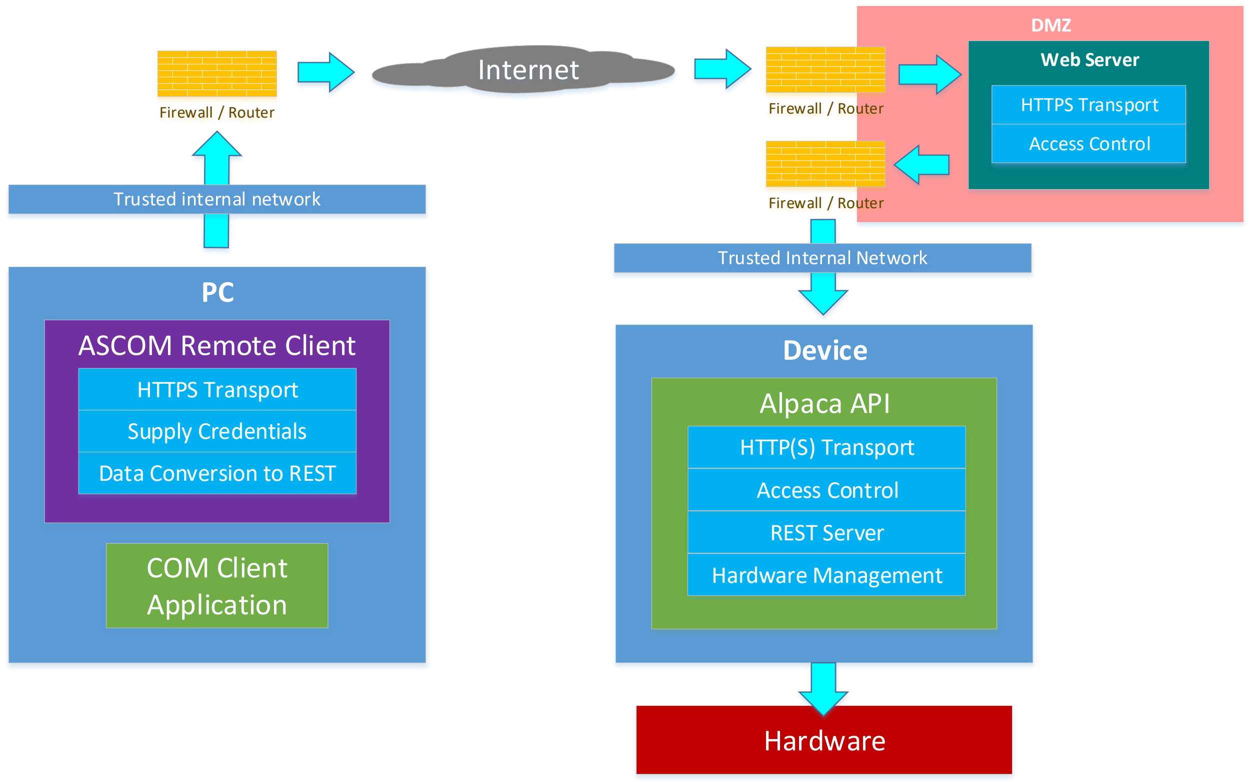


Figure 11 - A COM application using a remote non-Windows driver over the Internet

3.7 Example - Non-Windows Client uses Remote COM Driver

In time, native Alpaca clients will be developed and this example shows how these can use existing Windows COM drivers through the Remote Server.

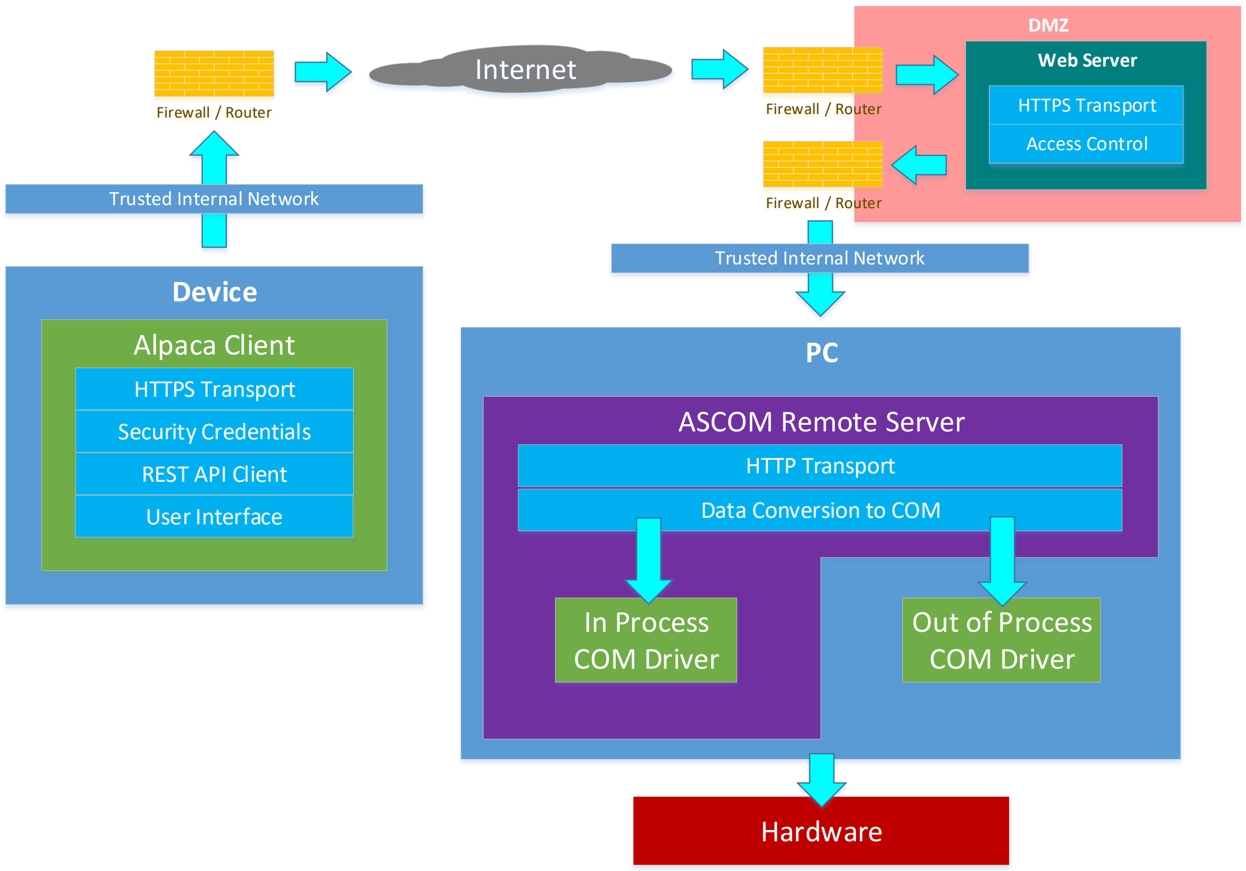


Figure 12 - Non-Windows client using a COM driver on a remote PC over the Internet

4. Appendix – COM Server Types

There are two different types of COM servers: "in process" and "out of process". The key difference is whether the COM server (the ASCOM driver) runs within the client application's process or outside it in an independent server process of its own as shown below.

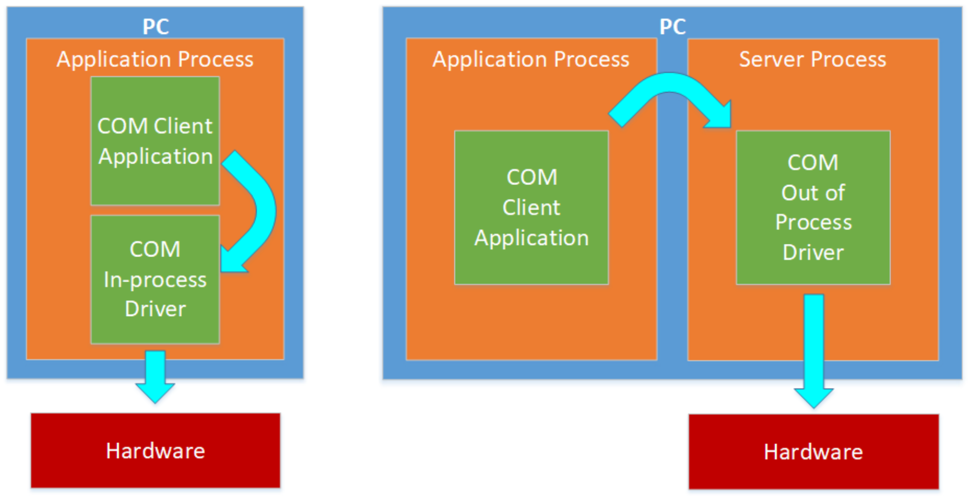


Figure 13 - In-process and out-of-process COM drivers

Drivers created and registered to COM as DLLs are "in process" types. Those that use ASCOM's Local Server pattern, and consequently have their own executable, are "out of process" types.