ASCOM REST

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 REST, 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

1. Contents

1. ASCOM REST

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 or not 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 standardised 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.

The ASCOM interfaces have proven themselves to be robust, mature, stable and suitable to the mission. For many years they 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.

Detailed explanations of the architecture and principles underlying the ASCOM APIs are given in section 0.

1.2 ASCOM APIs in RESTful Form

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 REST 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.

[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)

The ASCOM abstract APIs are mature, stable, and have been proven suitable for the mission over the last twenty years. Their use enables 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.

ASCOM REST enables heterogenous configurations utilising any hardware and operating system, so long as they have a TCP/IP stack.

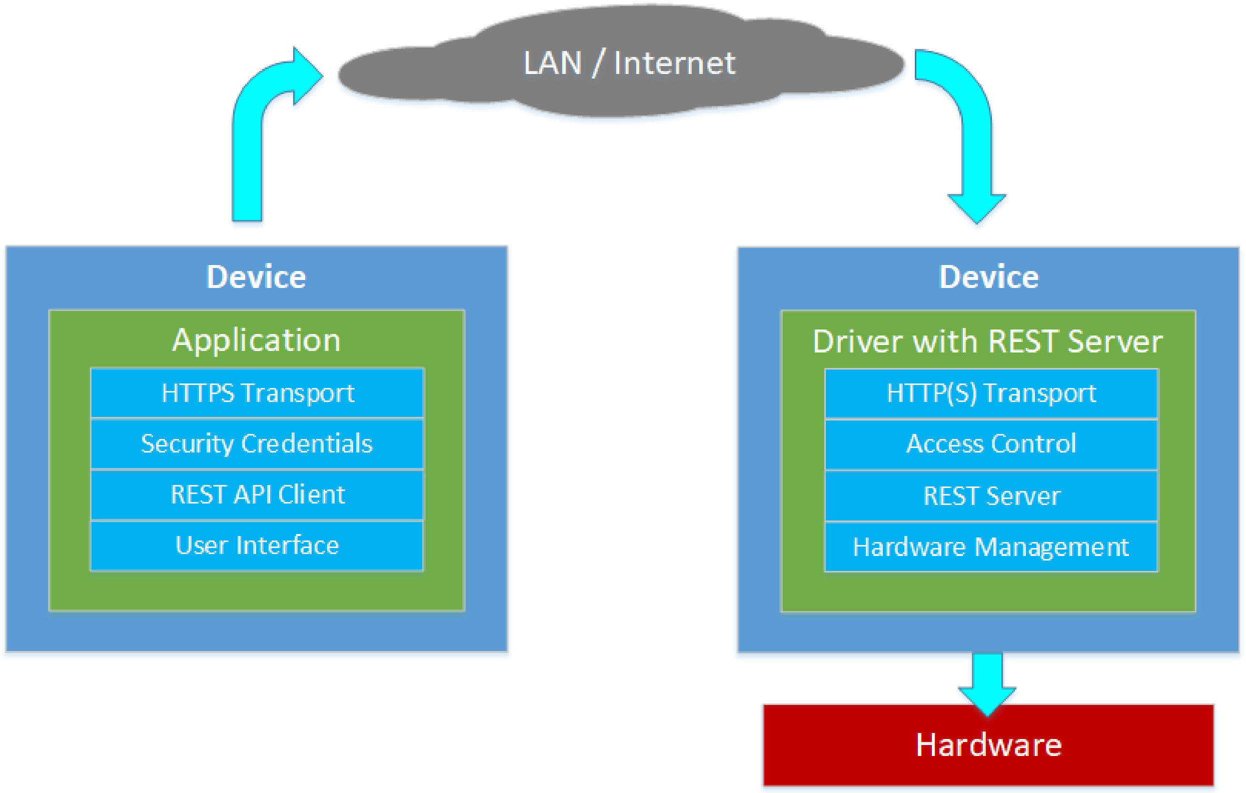


Figure 1 - Application to Driver communication using ASCOM REST

1.3 Transition and Migration

New applications and devices can be coded from the outset to use the new REST based ASCOM APIs, however, they will be of limited value if they can only communicate with other REST based applications and drivers. What is needed is transition technology to enable REST based 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 APIs across REST-based connections. Any device with a COM-based ASCOM driver can today interoperate with any COM-based ASCOM client program anywhere in the world, over the internet, using the new REST-based APIs. 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 REST implementation faithfully implements the standardized ASCOM abstract interfaces and API. Finally, ASCOM Remote provides both a reference implementation of the new ASCOM REST 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 Focuser API over REST, perhaps based on a micro-board processor such as the Raspberry Pi or Arduino, and 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 REST API Specification

In this section we see how modern advanced tools are used to express the proven ASCOM APIs as platform-neutral data representations and REST resources. ASCOM REST respects ASCOM's *language-independence* because it is specified using language and platform independent standards such as TCP/IP and JSON. Exploitation can be ubiquitous because there are REST implementation libraries for virtually every language and platform.

The REST 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 OpenAPI Initiative and Swagger

Describing to users how to interact with a RESTful API can be a daunting task. In the past, APIs have resorted to manually edited human readable documentation which is difficult to produce from separately created API implementation code and message structures. Enter the OpenAPI Initiative (OAI). This organization has formalized a vendor-neutral API description format based on a set of tools and specifications called Swagger, originally developed by SmartBear Software. SmartBear have now donated this technology to OAI and also provide free community-driven tools and services, along with commercial tools and services with which they run their business.

The long-term objective of OAI is to produce not only API documentation standards but also automated generated RESTFul implementation server stub and client code generators which are driven by the same API specification from which the documentation is produced. Currently Swagger Codegen provides generators and it freely available for open source projects. Follow this link to the Swagger page where Swagger Codegen is described to see the list of languages and environments for which code generators are already available. It should be noted that the "servers" would normally be the astronomical devices which implement the relevant ASCOM API.

2.2 ASCOM REST API Documentation

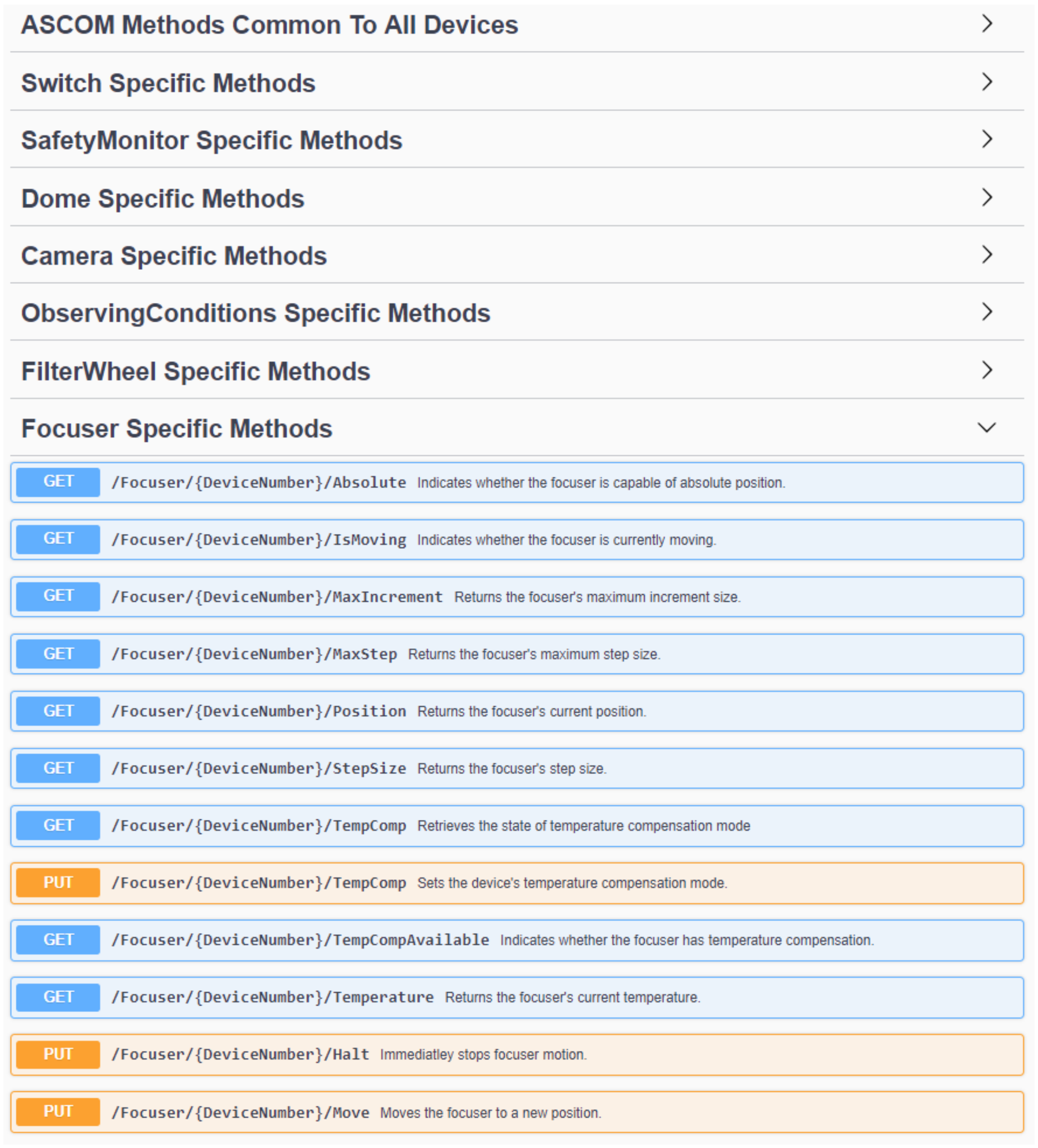
The ASCOM RESTful APIs are documented using the Swagger toolset and are available through a URL on the ASCOM Standards web site. The ASCOM API is fully documented here:

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 REST. Full details on how to construct, send and decode ASCOM REST API calls are given in the companion ***ASCOM REST API Reference*** document.

Here is an example of the OpenAPI/Swagger documentation of the ASCOM Focuser API looks like (in Swagger, a "method" is any REST resource including object properties and methods):



ASCOM Methods Common To All Devices

Switch Specific Methods

SafetyMonitor Specific Methods

Dome Specific Methods

Camera Specific Methods

ObservingConditions Specific Methods

FilterWheel Specific Methods

Focuser Specific Methods

GET `/Focuser/{DeviceNumber}/Absolute` Indicates whether the focuser is capable of absolute position.

GET `/Focuser/{DeviceNumber}/IsMoving` Indicates whether the focuser is currently moving.

GET `/Focuser/{DeviceNumber}/MaxIncrement` Returns the focuser's maximum increment size.

GET `/Focuser/{DeviceNumber}/MaxStep` Returns the focuser's maximum step size.

GET `/Focuser/{DeviceNumber}/Position` Returns the focuser's current position.

GET `/Focuser/{DeviceNumber}/StepSize` Returns the focuser's step size.

GET `/Focuser/{DeviceNumber}/TempComp` Retrieves the state of temperature compensation mode

PUT `/Focuser/{DeviceNumber}/TempComp` Sets the device's temperature compensation mode.

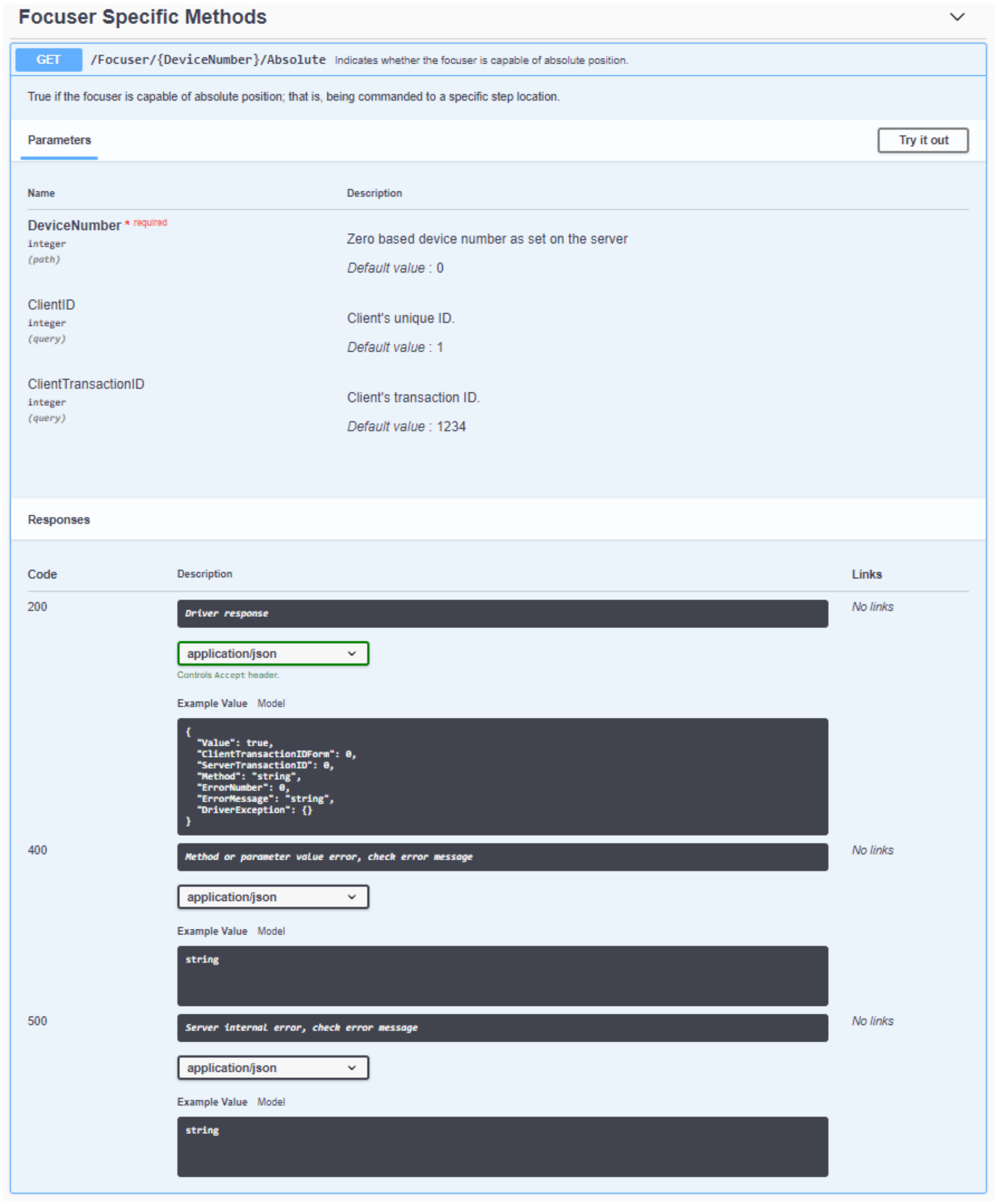
GET `/Focuser/{DeviceNumber}/TempCompAvailable` Indicates whether the focuser has temperature compensation.

GET `/Focuser/{DeviceNumber}/Temperature` Returns the focuser's current temperature.

PUT `/Focuser/{DeviceNumber}/Halt` Immediatley stops focuser motion.

PUT `/Focuser/{DeviceNumber}/Move` Moves the focuser to a new position.

Figure 2 - Swagger Documentation - Top level



Focuser Specific Methods

GET /Focuser/{DeviceNumber}/Absolute Indicates whether the focuser is capable of absolute position.

True if the focuser is capable of absolute position; that is, being commanded to a specific step location.

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* application/json Controls Accept header. Example Value Model	*No links*
400	*Method or parameter value error, check error message* application/json Example Value Model `string`	*No links*
500	*Server internal error, check error message* application/json Example Value Model `string`	*No links*

Example value for 200:

```
{
  "Value": true,
  "ClientTransactionIDForm": 0,
  "ServerTransactionID": 0,
  "Method": "string",
  "ErrorNumber": 0,
  "ErrorMessage": "string",
  "DriverException": {}
}
```

Figure 3 - Swagger Documentation - Method detail

3. ASCOM Remote

3.1 Concept and Capabilities

ASCOM Remote uses the ASCOM REST APIs to deliver three new 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 REST 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 REST 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 REST calls, which are sent to networked devices that expose the ASCOM REST API.
2. A ***Remote Server*** that can host up to 10 COM based ASCOM drivers and present these through the ASCOM REST interfaces.

This graphic shows a *Remote Client* driver communicating with remotely hosted COM or native drivers on behalf of the COM client application:

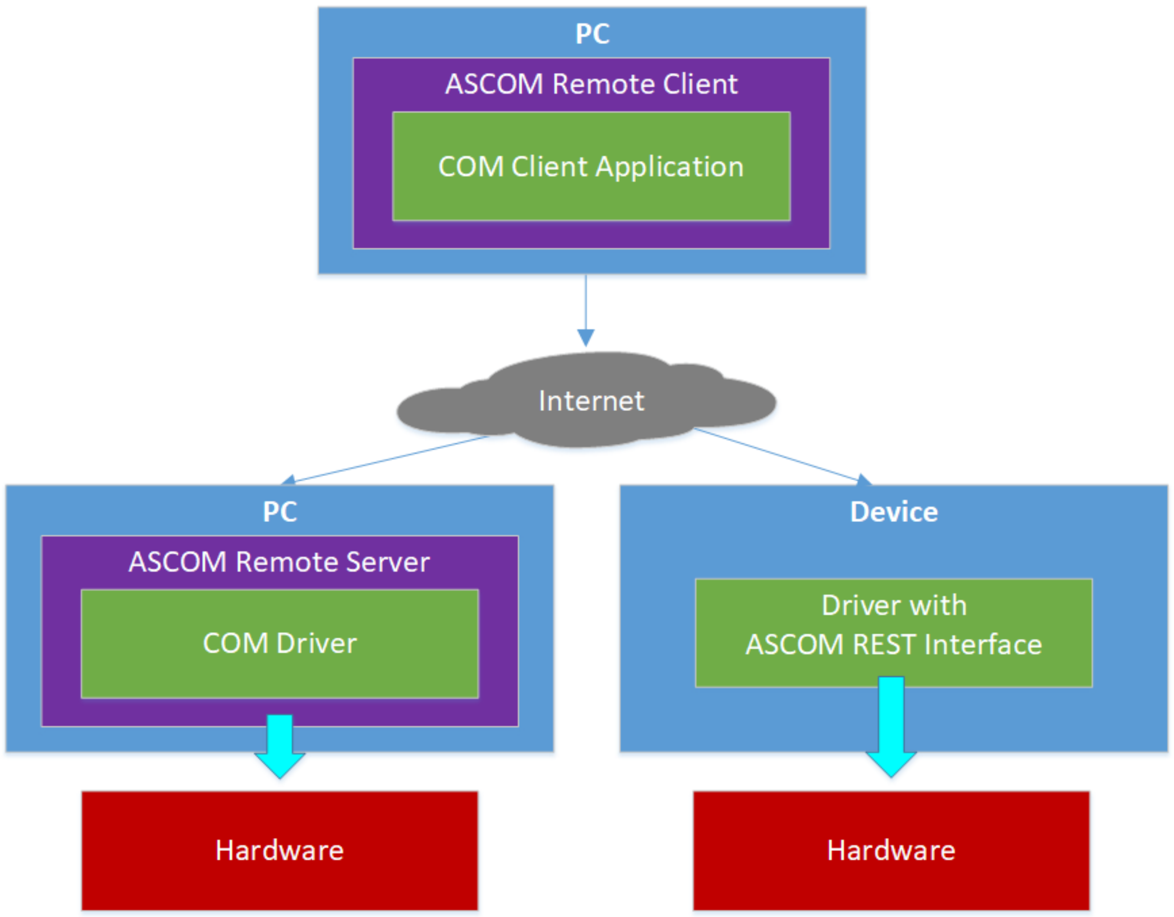


Figure 4 - COM Client accessing a remote COM or non-COM driver

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

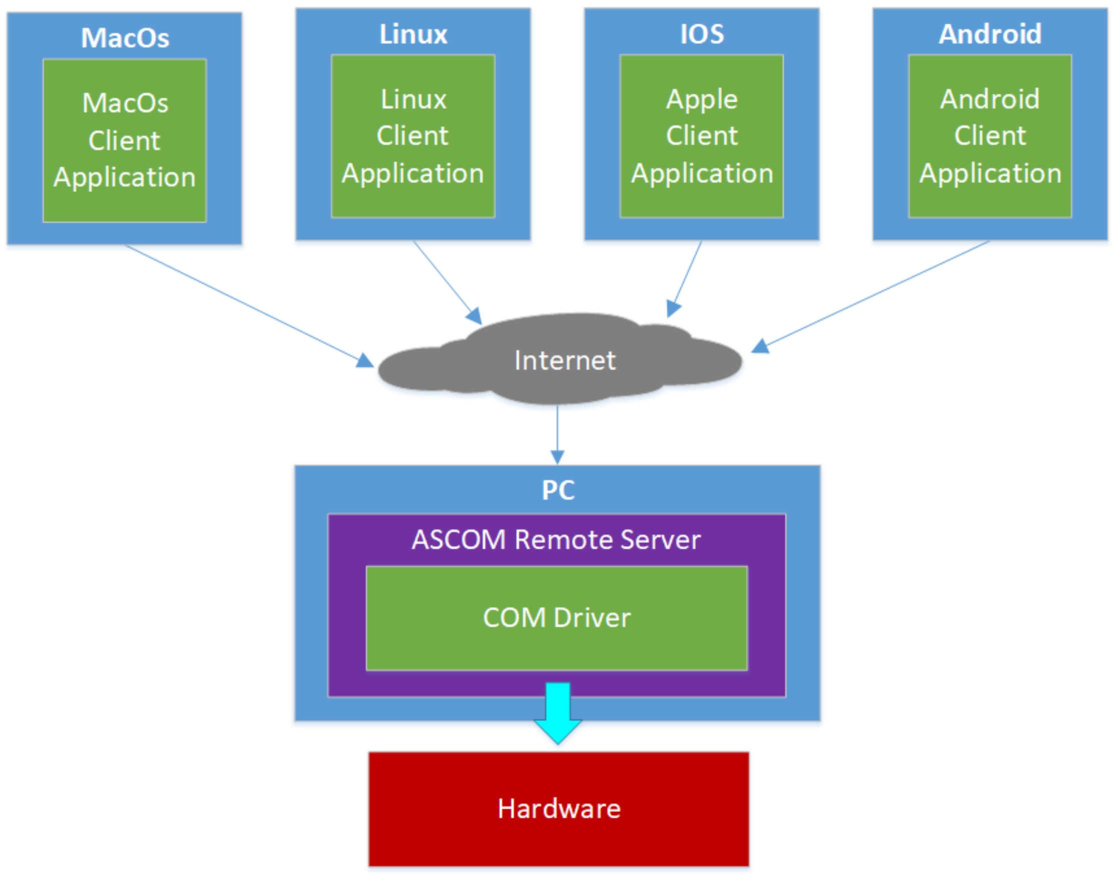


Figure 5 - Non-Windows applications accessing a remote COM based driver

3.1.1 COM Server Types

Some of the following graphics show 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.

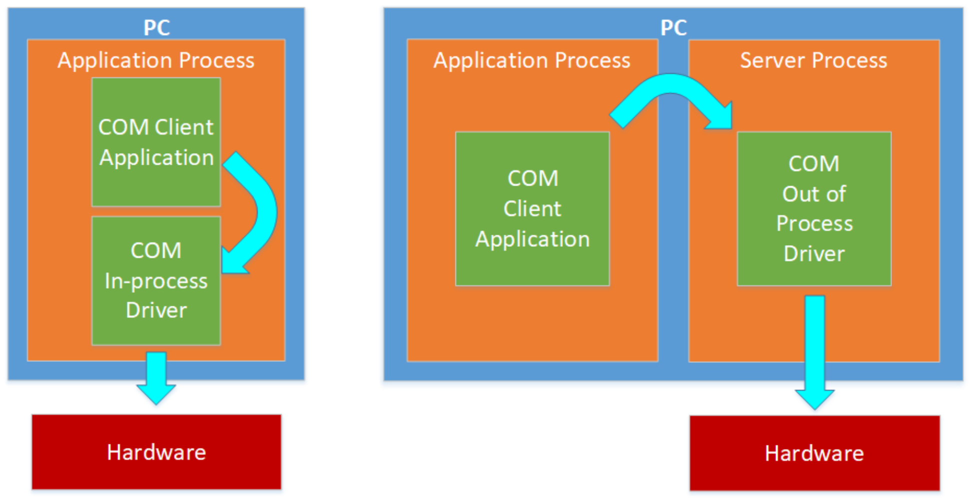


Figure 6 - In-process and out-of-process COM drivers

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.

3.1.2 High level solution architecture

Realising this conceptual model in practice requires:

Requirement	Approach to realisation
A platform neutral data representation of the ASCOM device APIs	**ASCOM REST interface specification**
A common communications protocol supported by many client devices	**http protocol over TCP/IP**
A means to encode the data so that it can be reliably transported by the communications protocol and be easily converted into meaningful data structures within the client.	**JSON data encoding**
A means to protect the data whilst in transit	**https (http over Transport Layer Security or its predecessor, Secure Sockets Layer)**
A means to control access to the remote devices	**http basic authentication**

3.1.3 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 REST 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.

3.1.4 ASCOM Remote Server

This component exposes an IP end point and translates incoming ASCOM REST API requests back to COM requests before passing them to the configured drivers installed on that machine.

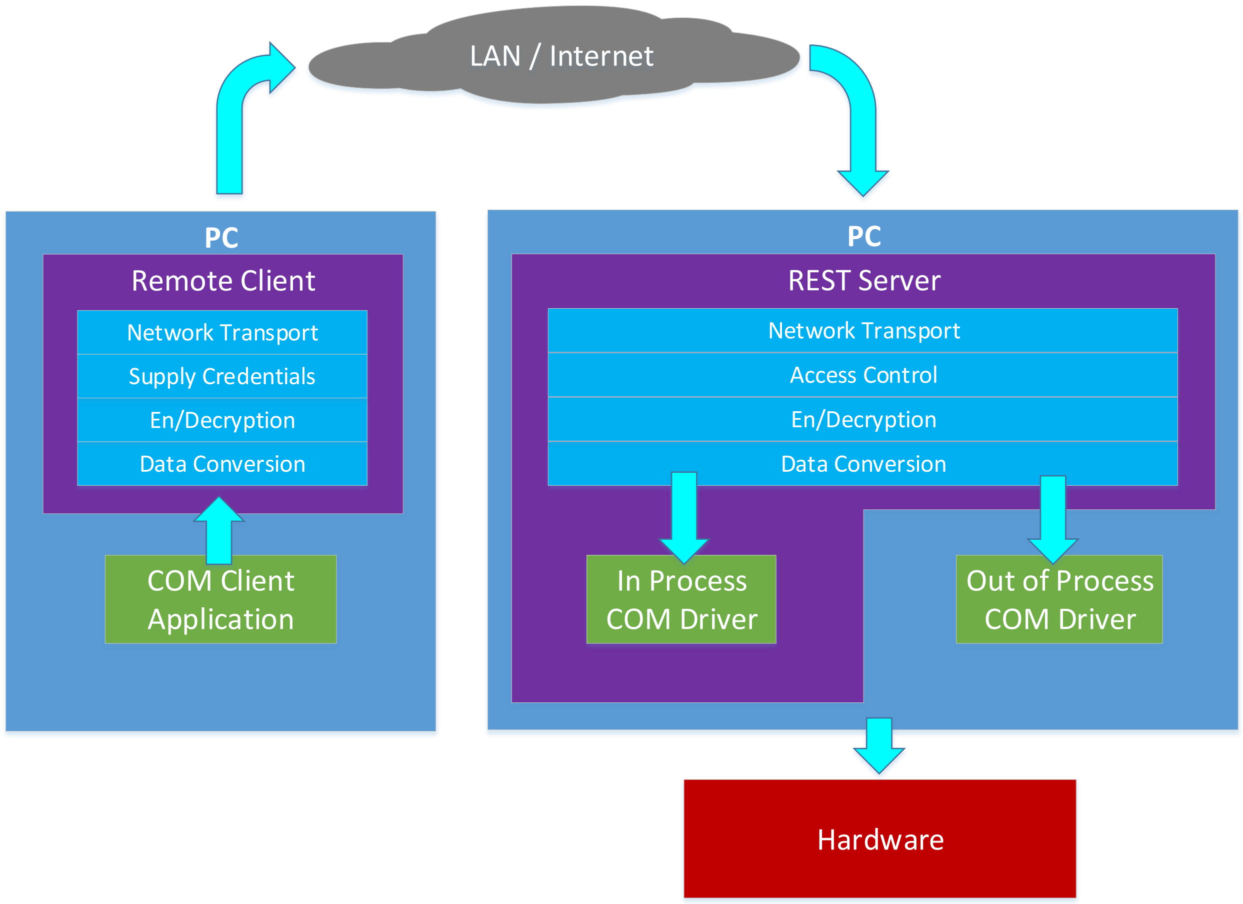


Figure 7 - Logical view of the ASCOM Remote client and server components

3.2 Security Concerns

Network and device security are important considerations and effective solutions can involve complex and architectures. The ASCOM Remote client and server applications support basic HTTP authentication and the Remote Client supports HTTPS connections. However, both applications rely on other dedicated appliances to provide fire walling and termination of SSL links.

> ***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.*

3.3 Example – COM Application uses COM Driver on Local Network

This graphic shows a physical realisation of an internal use case, where data does not need to travel over the Internet or be protected in any way. The client application runs one PC and accesses a driver hosted within the ASCOM Remote Server on another PC.

In this configuration, HTTP must be used internally on the trusted network because the REST Server cannot terminate SSL/TLS connections.

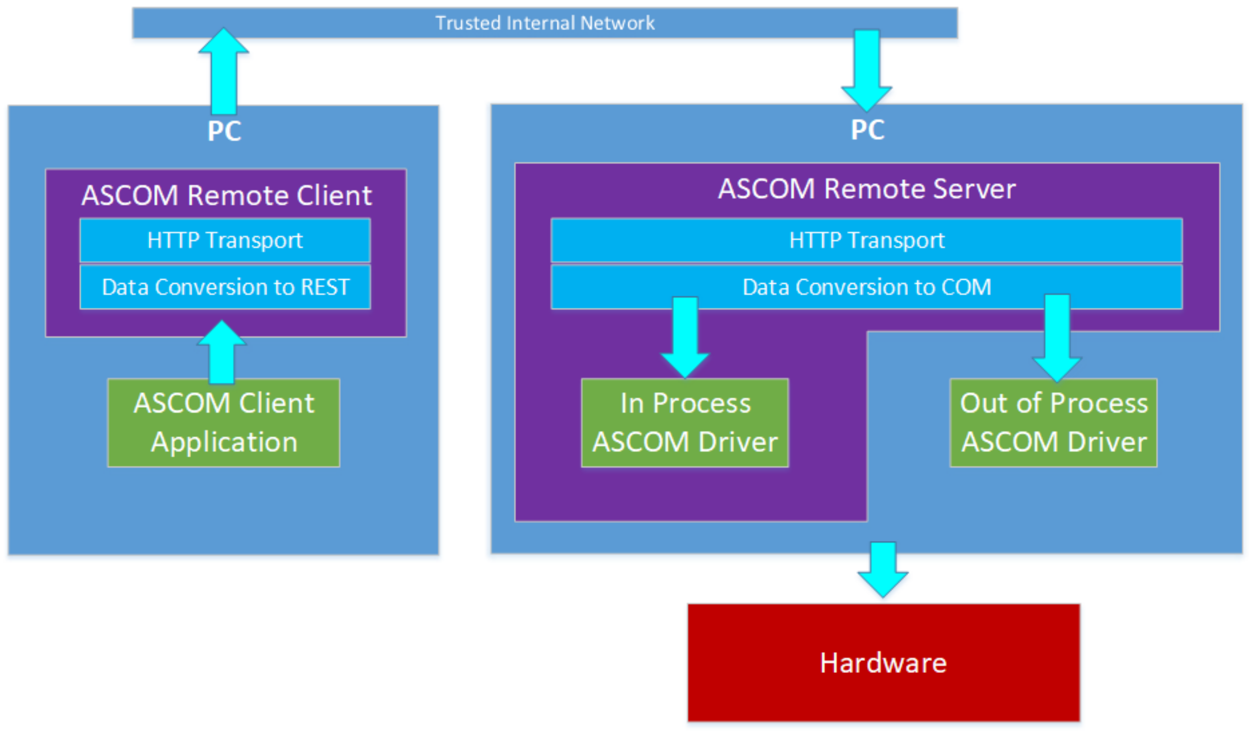


Figure 8 - COM application using a COM driver on the local network

3.4 Example – COM Application uses COM Driver on Remote PC

The REST Server supports HTTP connections but does not support HTTPS connections owing to the complexity of options in this area. An HTTP connection can be set up over the internet just as in the case of the internal network but of course there is no protection for data in transit and no authentication of the client by the server, so anyone could access the controlled device.

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

The Apache 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:

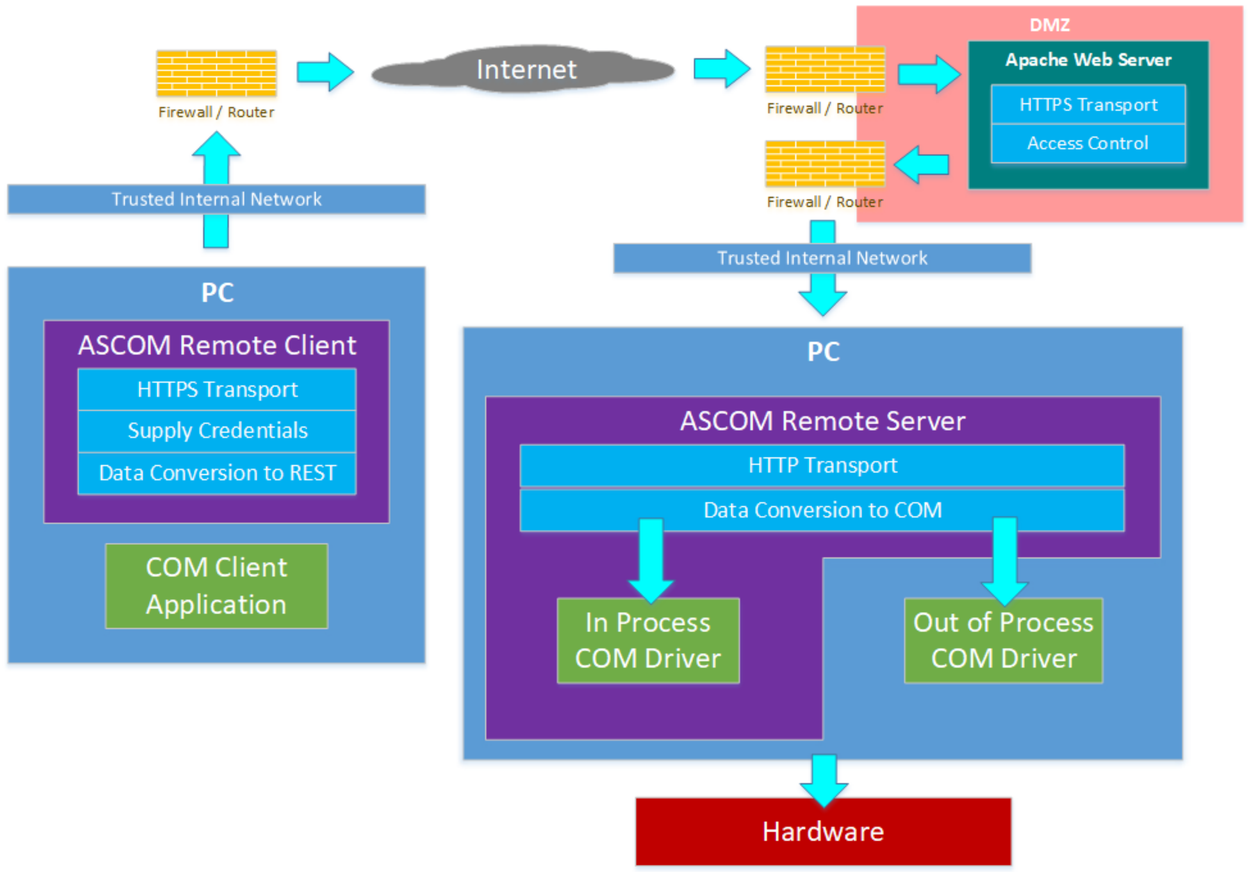


Figure 9 - COM application using a COM driver on a remote PC over the Internet

3.5 Example - Non-Windows Client uses Remote COM Driver

One of the key objectives of ASCOM REST is to enable non-Windows devices to access Windows based ASCOM drivers. This diagram illustrates the approach where the device client needs to:

- Connect to the remote access server
- Supply authentication credentials
- Issue appropriate HTTP GET and PUT commands
- Interpret the resultant JSON responses.

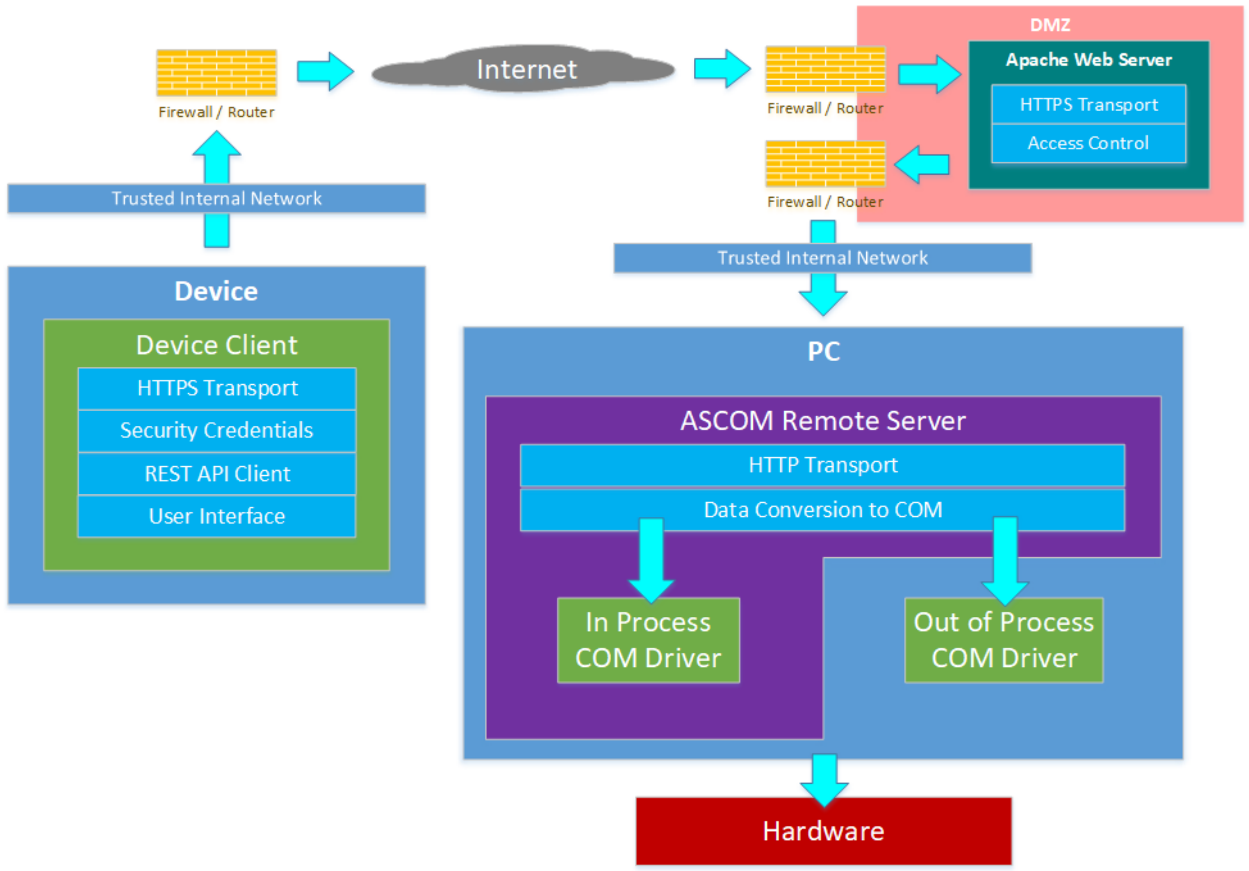


Figure 10 - Non-Windows client using a COM driver on a remote PC over the Internet

3.6 Example – COM Application uses Remote Non-Windows Driver

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

This diagram illustrates the approach where a Windows COM client needs to communicate with a new REST API only driver:

- Connect to the remote access server
- Supply authentication credentials
- Issue appropriate HTTP GET and PUT commands
- Interpret the resultant JSON responses.

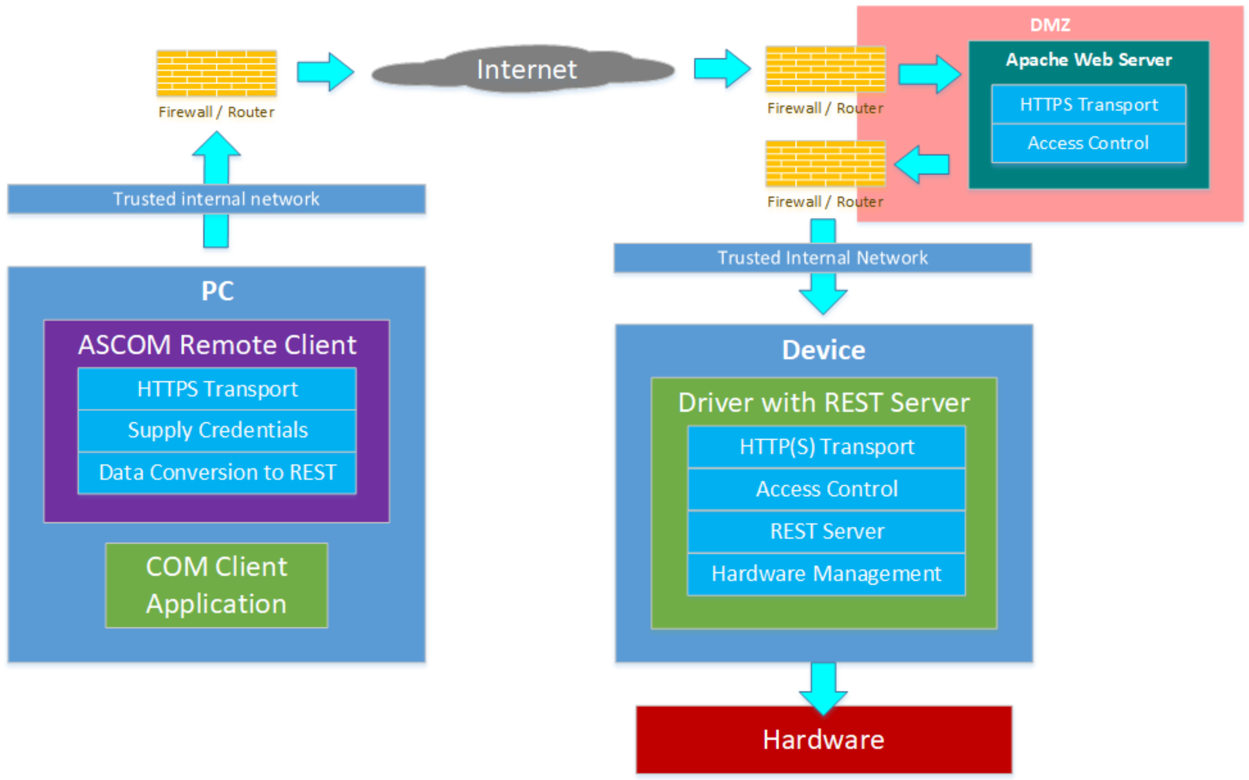


Figure 11 - A COM application using a remote non-Windows driver over the Internet