User Guide

Rob Morgan and Peter Simpson

2024

Table of Contents

Rob Morgan’s original document was updated in 2024 by Peter Simpson to describe Alpaca and changes in Platform 7.

What is ASCOM?

There are a number of items or entities associated to the acronym ASCOM. The first is to know that it stands for Astronomy Common Object Model. It was originally invented in late 1997 and early 1998 by Bob Denny, when he released two commercial programs and several freeware utilities that showcased the technology.

The ASCOM Initiative was formed as a small group of software developers from around the world to give oversight and continue the development effort of the ASCOM Platform. Their website can be found at ascom-standards.org.

The ASCOM Platform is a collection of programs to standardize the core functionality for normal operations and observing in astronomy hardware and bring these together in a common implementation for use in client software applications. ASCOM is a many-to-many and language-independent architecture, supported by most astronomy devices which connect to computers.

The first observatory to adopt ASCOM was Junk Bond Observatory, in early 1998. It was used at this facility to implement a robotic telescope dedicated to observing asteroids. The successful use of ASCOM there was covered in an article in Sky & Telescope magazine. This helped ASCOM to become more widely adopted. As of today the ASCOM platform is on its 6th version of major releases. Figure 1 shows the layer interactions with compliant astronomy instruments and compliant software architectures.

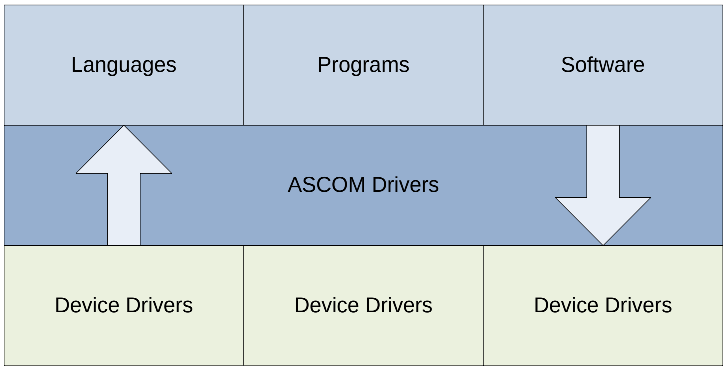


Figure 1

An ASCOM driver acts as an abstraction layer between the client and hardware thus removing any hardware dependency in the client, and making the client automatically compatible with all devices that supports the minimum required properties and methods. For example, this abstraction allows an ASCOM client to use an imaging device without needing to know whether the device is attached via a serial or network connection.

What are the ASCOM Interfaces?

The ASCOM Initiative has published a number of interface standards and guidelines that describe the ways in which ASCOM compliant client software and devices interact.

To be called "ASCOM compliant", a driver, component, or application must meet all applicable guidelines and standards. Only then can drivers, interfaces, or a component's packaging and user interface carry the ASCOM logo.

Interfaces are technical contracts that ensure that two pieces of software can communicate meaningfully with each other. They are the means by which the "abstraction layer" mentioned in the previous paragraph works.

Interfaces are particularly helpful to developers because they enable development software to report any interface requirements that are not met. E.g. Your application should provide this property, but it is missing. Or, this method will receive an integer parameter value from the client, but you have coded it to expect a string.

Why are Drivers Important?

Well, step back from astronomy for a minute. When you go out and buy a new printer, you can be virtually certain that it will work with all of the programs on your computer. Likewise, when you install a new program on your computer, you can be virtually certain that it can print to your existing printer, even if it's no longer in production. We take this for granted. Printers come with a disk that installs the driver for the printer. The driver takes care of all of the details for that particular printer, leaving all of your Windows programs with a common printer-agnostic way to send pages to paper.

OK, back to astronomy. Until ASCOM, each astronomy program that needed to control telescopes, focusers, and so forth had to include its own code for all of the different instrument types out there. Keeping up with new instruments, supporting old ones, and dealing with firmware revisions is a tremendous burden. Every astronomy software developer is faced with (re)writing code for every device he intends to support. Furthermore, astronomy device manufacturers are faced with having to beg an array of astronomy software vendors to support their device in the future, delaying adoption of their new devices.

ASCOM eliminates these problems. Most programs that need to control telescopes, focusers, etc., now expect a driver to be available for those instruments. For example, you may have several programs that need to control your telescope (planetarium, imaging software, alignment assistance tool). If there is a driver for your mount, you can be virtually certain that all of these programs can control it. To find out, you ask your mount maker "Does your mount have an ASCOM driver?" If so, you're all set. No more asking a bunch of software developers "Does your software support my mount?"

ASCOM defines a collection of properties and methods, known as an interface, that ASCOM compliant software can use to communicate with ASCOM compliant devices. By testing various properties, an ASCOM client application can determine what features are available for use.

Any language that supports access to Microsoft COM objects can be used to create ASCOM clients and drivers. Figure 2 shows a breakout of the typical layers used by the Clients, ASCOM, Manufacturer Driver, and hardware devices.

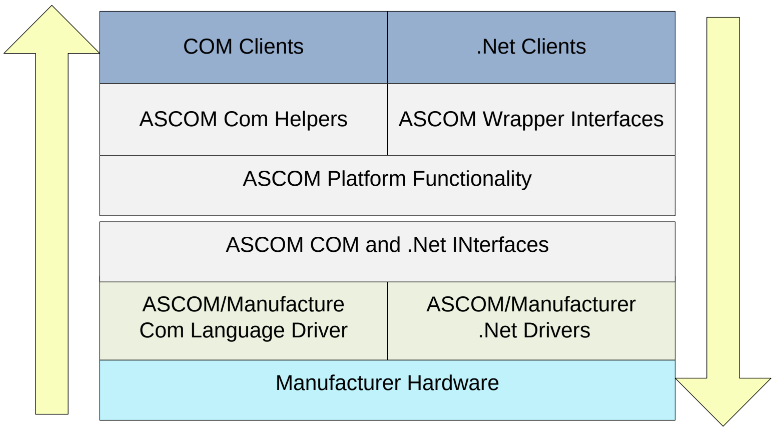

COM Clients
.Net Clients
ASCOM Com Helpers
ASCOM Wrapper Interfaces
ASCOM Platform Functionality
ASCOM COM and .Net INterfaces
ASCOM/Manufacture Com Language Driver
ASCOM/Manufacturer .Net Drivers
Manufacturer Hardware

Figure 2

A key ASCOM Initiative principle is to maximize compatibility with Microsoft's Component Object Model (COM). This enables a variety of development languages to be compatible with the ASCOM architecture. For the examples in this article, we've selected one compiled language, C#, and one scripting language, VBScript, from the many available development languages.

Why Use COM?

COM is built into Windows and only runs on Windows. Any language can use COM to display on the screen or write to a disk file but COM isn't just an I/O service. It is a Component Object service (hence the name which stands for Component Object Model). Within any Object-Oriented Programming language, one can define and create objects, then use their members while treating the object as a black box.

But Components are different. Components exist apart from the application’s code and are shared by all applications. Once loaded, they can be used exactly like objects created in the native language. The cool thing, though, is that one component can be used by any program in any language. This makes Components a natural choice for creating drivers, which after all, are things that must be usable by any program in any language.

How COM Works

When an application asks the Operating System (OS) for a component, it uses the ID of the component. IDs are system-wide. The location on disk of the component is not important to the application. The OS has an object broker that uses the ID to locate the component's code and activate it. Thereafter it is ready for use by the app. Multiple instances of a component can be used by different apps simultaneously.

The OS call that activates the component returns a reference to the activated component. The component reference is kept in a variable and used just like a reference to an object created in the app's native language. Thus the application’s code sees the component as identical to one of its own objects and uses it identically. Perhaps you can now see how powerful the component concept is, and why it was a natural choice for ASCOM drivers.

What is Alpaca?

ASCOM’s COM interfaces are an implementation of the ASCOM device interfaces using Microsoft’s component object model technology. Alpaca is an implementation of the ASCOM device interfaces using the HTTP network protocol, which enables clients and devices to communicate through wired and wireless networks.

Alpaca supports the same properties and methods that that are available in COM objects. Yes, the means of calling them is different, but the inputs and outcomes are the same because they both work to the same interface technical contract.

Alpaca clients use network broadcasts to discover Alpaca devices on the network and subsequently use point to point communication to use the devices. Alpaca enables device manufacturers to create “driverless” devices based on low-cost micro-controllers to reduce cost and simplify deployment.

Alpaca has 100% interoperability with traditional COM clients and drivers. The ASCOM Platform provides all the means necessary for traditional COM clients to transparently find and use Alpaca devices and for Alpaca devices to find and use traditional COM drivers.

Why use Alpaca?

Alpaca is operating system agnostic because it is based on HTTP and is implemented on almost every operating system, so Alpaca clients and devices can run on all Linux, operating systems, MacOS, iPadOS and Android as well as on Windows.

In many cases Alpaca devices provide improved reliability and flexibility over traditional COM drivers because they use ethernet, which is more robust than serial and USB links and operates over much longer distances.

If you are not a Windows fan, an ASCOM compliant observatory can now be built without any use of Windows.

What ASCOM is not

ASCOM Platform is not meant to implement all the features of any one particular manufacturer’s hardware device or devices. It’s meant to standardize core functionality for normal operations and observing.

Some manufacturers may implement a driver that implements both the ASCOM standards and their own interfaces for specific features. This is perfectly acceptable and encouraged since the ASCOM Initiative cannot cover all possible variations of all the manufacturers.

ASCOM Initiative Mission Statement

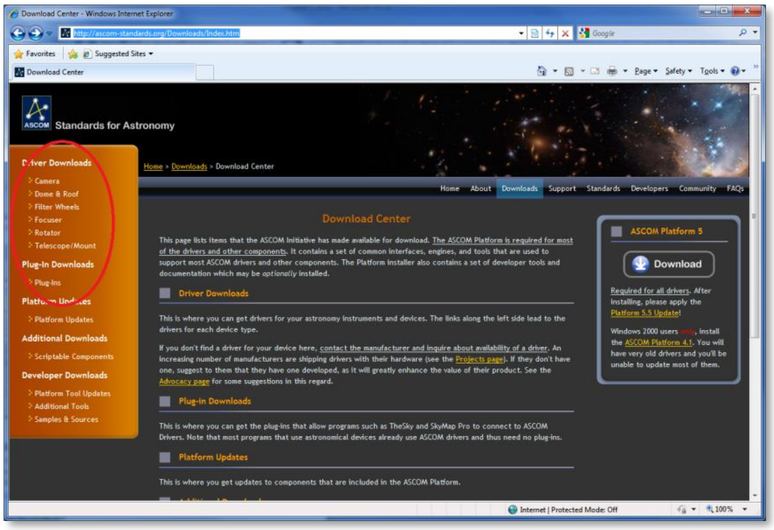
1. Establish a set of vendor-independent and language-independent interface standards for drivers that provide plug-and-play control of astronomical instruments and related devices.

2. Provide general requirements and guidance for quality and behavior of drivers.
3. Promote the use of these standard drivers from any astronomy-related software.
4. Ensure that drivers are usable from the widest possible variety of programs and languages, including Windows Active Script languages and Automation based tools.
5. Promote (but not absolutely require) open-source implementations of the drivers.
6. Promote script ability of astronomy software without standardizing application level interfaces (which would inhibit innovation).
7. Provide general requirements for quality and behavior of application scripting interfaces, aimed at making script writers' experiences consistent and robust.

The ASCOM Platform

Installing Drivers

Now that you have the platform installed you'll need driver(s) for the various equipment you have. These drivers will probably come the manufacturers and be marked for ASCOM compatibility but if not, you might find them here: https://ascom-standards.org/Downloads/Index.htm.



Download Center

This page lists items that the ASCOM Initiative has made available for download. The ASCOM Platform is required for most of the drivers and other components. It contains a set of common interfaces, engines, and tools that are used to support most ASCOM drivers and other components. The Platform installer also contains a set of developer tools and documentation which may be *optionally* installed.

Driver Downloads

This is where you can get drivers for your astronomy instruments and devices. The links along the left side lead to the drivers for each device type.

If you don't find a driver for your device here, contact the manufacturer and inquire about availability of a driver. An increasing number of manufacturers are shipping drivers with their hardware (see the Projects page). If they don't have one, suggest to them that they have one developed, as it will greatly enhance the value of their product. See the Advocacy page for some suggestions in this regard.

Plug-in Downloads

This is where you can get the plug-ins that allow programs such as TheSky and SkyMap Pro to connect to ASCOM Drivers. Note that most programs that use astronomical devices already use ASCOM drivers and thus need no plug-ins.

Platform Updates

This is where you get updates to components that are included in the ASCOM Platform.

ASCOM Platform 5

Required for all drivers. After installing, please apply the Platform 5.5 Update!

Windows 2000 users only, install the ASCOM Platform 4.1. You will have very old drivers and you'll be unable to update most of them.

Once you locate the drivers, go ahead and install each one that you need. Each may come with specific instructions, so be sure to read any included files after installing.

Platform Simulators

Simulators have been created for a number of devices that mimic the use of drivers and the client applications that access them. These simulators test each of the internal ASCOM drivers to insure integrity of the installed ASCOM platform. These simulators provide a convenient tool for application software developers to test their programs with known good drivers under controlled conditions. The simulators also serve driver developers as reference implementations of the driver standards. If there is a question about the behavior of a property or method, the behavior of the appropriate simulator serves as the reference.

Diagnostics Tool

The Diagnostics application will evaluate the current ASCOM platform installed on the local computer to create a log file that can be used for troubleshooting issues. This text file can then be sent to the ASCOM Initiative developers or others for evaluation of the issues.

Profile Explorer Tool

The Profile Explorer allows viewing of the ASCOM Profile. The ASCOM profile, figure five, stores information about the devices and drivers installed. Drivers are normally registered in the profile store during the setup process. ASCOM specifically stores the COM ProgID of the driver. All drivers must register with the ASCOM profile and may use the profile to store other configuration or runtime information.

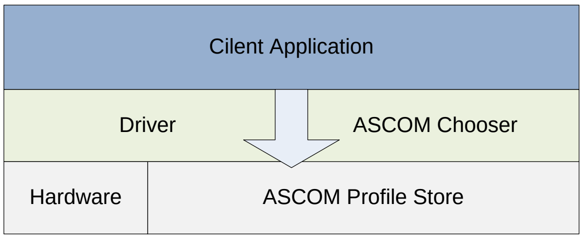
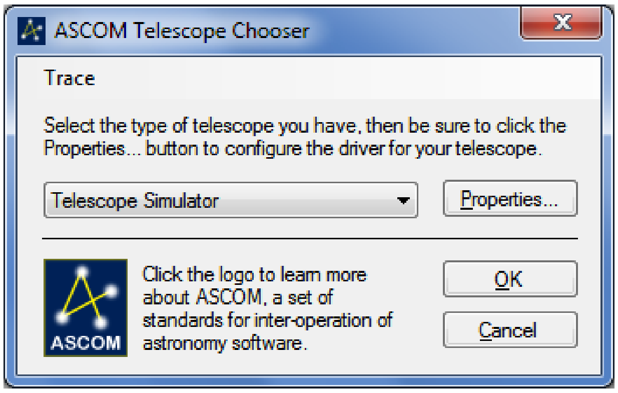


Figure 5

Chooser Tool

The Chooser provides a way for you to select the device to work with, within an application.



Select the type of telescope you have, then be sure to click the Properties... button to configure the driver for your telescope.

Click the logo to learn more about ASCOM, a set of standards for inter-operation of astronomy software.

Developer Information

Core Assemblies

All the listed assemblies are installed into the Global Assembly Cache (GAC) as part of the platform installation process.

- **ASCOM.Astrometry** - This encapsulates the Naval Observatory Vector Astrometry Software (NOVAS) and Kepler's Laws of Planetary Motion.
- **ASCOM.Attributes** - Used by the ASCOM LocalServer and the SettingsProvider to load settings. The LocalServer uses this to control which assemblies to load.
- **ASCOM.Controls** - This contains a common set of user interface elements for use by all developers.
- **ASCOM.DriverAccess** - This is a .NET assembly that provides high-level simplified access to ASCOM drivers for developers writing client applications. This provides automatic switching between the preferred early-binding interfaces and, for older drivers that don't support it, late-binding.
- **ASCOM.Exceptions** - This contains common exception classes used by the ASCOM platform and for internal exceptions. Drivers are permitted to directly throw this exception as well as any derived exceptions.
- **ASCOM.Interfaces** - Master interfaces are installed in a registered COM type library and a .NET primary interop assembly (in the GAC). For .NET, a registered master primary interop assembly (PIA) is provided. It

appears in the .NET References window, COM tab, as "ASCOM Master Interfaces for .NET and COM (V1.0)" (the same friendly name as seen in COM from OLEView etc.). Once referenced in a .NET project, it will show as ASCOM.Interfaces, the namespace containing the interfaces (e.g. ASCOM.Interface.ITelescope).

- **ASCOM.SettingsProvider** - SettingsProvider integrates the ASCOM Profile store with the Visual Studio settings designer and the application settings architecture. It is intended for use by driver developers and is incorporated into the VS template projects.
- **ASCOM.Utilities** - Contains things like the Serial, Profile, Chooser and other items like date and time conversions,

ASCOM Profile

The profile is the store for Driver and Device information. Used by the Chooser to locate the Windows registered ProgID of drivers. The profile can be access directly by the drivers or by using the Profile Explorer.

ASCOM does not mandate that all drivers have to use the Profile component to store their configuration information, but it does mandate that the Profile is used to register the device so that Chooser always knows where device registration information is located.

Choosing and Configuring the Driver

Since all focusers look the same to the application, it first has to give the user a chance to select the required focuser. To do this, the application uses a component that comes with the ASCOM Platform called the Chooser. Omitting the details, the app displays the Chooser and the user selects the type of focuser from a drop-down list.

Once chosen, the user then clicks a Properties button, which loads the driver into the Chooser and asks the driver to show its configuration window. There, the user sets the COM port that the focuser is connected to, as well as anything else the MicroGlide needs for one-time configuration.

The Chooser looks exactly the same regardless of which language the app is written in or which type of driver is being chosen. When the user finishes, they close the config window and the Chooser. At this point, the driver saves the settings entered by the user and disappears from the system. Thus, the user's settings are remembered and need not be entered again unless something changes.

.NET Client Toolkit

The client toolkit is a Visual Studio 2005 C# project that shows how to use the ASCOM .NET Client Toolkit. This console application contains code that shows how to access each type of driver, using the chooser and pre-selecting the simulator for each. Some info is printed out to the console window. This is a separate download from the platform

Driver Conformance Checker

This tool performs a comprehensive set of tests on a driver to determine its conformance with the relevant ASCOM interface standard. It also tests some aspects of driver behavior against the reference implementation. Use this tool to test your driver before each release (even pre-production)

COM Driver Guidelines

1. The driver must install and run on Microsoft Windows 7 or later with the latest service packs at the time of driver release. It should work on both 32- and 64-bit systems. Platform 6 does not support Windows 2000.
2. The driver must implement the published standard interface for the device type via a scriptable dispatch ("Automation") interface per the Microsoft Component Object Model (COM). Drivers should also implement "dual" interfaces which have both dispatch and early/VTBL binding (using the appropriate abstract standard interface that is part of the ASCOM Platform). See Driver Development Notes.
3. The driver must never "extend" the standard interface (add private members - properties and/or methods). If private members are desired, they must be exposed through a separate non-standard interface.
4. The driver must never display a modal window which requires user interaction to dismiss. All errors must be raised/thrown back to the client.
5. The driver must use the Helper component's Profile.Register() method for ASCOM registration. It is recommended that drivers also use the Helper's Profile object for storage of their persistent configuration, state data, etc., as well as the Helper's Serial object for serial port I/O. The Helper components are part of the ASCOM Platform and serve to isolate drivers from changes in Platform architecture. They also make development easier by providing high level functionality commonly needed by drivers.

6. Prior to release, the driver must pass the Conform tests using the current/latest version of the Conformance Checker test tool.
7. The driver must be delivered as a self-contained installer. It is unacceptable to ask users to copy files, edit the registry, run BAT files, etc. See Creating a Driver Installer.

There are a number of help files available on the ASCOM website for each type of driver that is supported. Within each are the properties and methods that are considered the standards.

Scriptable Components and Programs Guidelines

1. The product must run (at a minimum) on Microsoft Windows 7, Vista, and XP with the latest service packs at the time of driver release. It should work on both 32- and 64-bit systems. Windows 2000 is not supported in Platform 6.
2. The name ASCOM and/or the ASCOM logo must never be displayed for products which are in experimental, beta, preview, or any other such state. Only production products which are available and supported by the vendor or author are eligible.
3. Property and method names should be user friendly (e.g., SlewToCurrentObject instead of slw_curob). Use of so-called "Hungarian" notation is specifically discouraged. These interfaces may be used by scripters and should be user-friendly.
4. Wherever practical, property and method names should be consistent with existing ASCOM standard interfaces. For example, a property that implements equatorial right ascension should be called RightAscension, as used in the Telescope standard interface.
5. The product must implement scriptable dispatch ("Automation") interface(s) via the Microsoft Component Object Model (COM), and use only automation-compatible data types (see the data type requirements below).
6. Errors within your product must raise Automation exceptions (via IErrorInfo). The error info must contain both an error number that is based on FACILITY_ITF and an informative error message in English and optionally other languages. Optionally, methods which do not return values should return VARIANT_BOOL indicating success or failure. This allows clients to determine status while ignoring exceptions (e.g. On Error Resume Next and try/catch). IErrorInfo support is essential to providing client writers with the behavior they rely on. Very few of these

people manually test return values for errors with 'if' logic. They depend on their client environment to pop a meaningful alert box (or catch an exception with try/catch) when things go wrong.

7. Components must be 100% usable from an automation client without user interaction. For example, it is not permitted to require a user to dismiss an error alert window when the program is being controlled through a component automation interface. On the other hand, it is permitted to require the user to use a program's configuration features to set preferences. Another example of non-compliant behavior is a component server whose behavior changes or stops depending on whether it is a foreground or background window. The point of this requirement is to assure that, when used from a script, the program will never hang awaiting some user action such as a window shuffle, or clearing an error message box or selector dialog. Raise an exception and return to the client for handling the error or establishing the selection.
8. The product must be delivered as a self-contained installer. It is unacceptable to ask users to copy files, edit the registry, etc. See Creating a Driver Installer.
9. Executable components must self-register when first started and must support the command line options /REGSERVER and /UNREGSERVER to manually register and unregister them. Invocation with either of these options must immediately exit and must not start the program.
10. Any executable component must start automatically if one of the objects it serves is created by a client. Furthermore, it must exit automatically when the last reference to any object it is serving is deleted. Unless there is a good reason to do otherwise, an executable component should start in a minimized window. This is not a hard requirement as a component may benefit from displaying information as it operates. Unless this is the case, though, the component should remain out of sight (minimized) unless manually made visible by the user.

Scripting Interface Requirements

Besides the compatibility requirements described above, ASCOM interfaces must comply with the following. Remember that a major goal of ASCOM is to make programming with scripts straightforward, consistent, and non-intimidating for "ordinary people":

1. Property and method names must be user friendly (e.g., SlewToCurrentObject instead of slw_curob). Use of so-called "Hungarian" notation is specifically discouraged. These interfaces are for the use of ordinary people.
2. Wherever practical, property and method names should be consistent with existing ASCOM standard interfaces. For example, a property that implements equatorial right ascension should be called RightAscension, as used in the Telescope standard interface.
3. Methods must not be used to implement what are really properties. For example, a pair of methods called SetSpeed(newSpeed) and GetSpeed() are really a property Speed.
4. Interfaces for drivers must contain at least a Connected property and a SetupDialog() method. The Connected property establishes or breaks the physical link between the object and the device under control. The SetupDialog() method causes a modal dialog to appear which is used to configure the object for use with the device. Any settings that must persist must be the responsibility of the object itself, clients must not be required to persist object state.
5. The interface must be a scriptable dispatch ("Automation") interface per the Microsoft Component Object Model (COM), and use only automation-compatible data types (see the data type requirements below). It is not permitted to require the use of VTBL binding (via standard abstract interfaces). While a "dual" interface is permitted, ASCOM core functionality demands the use of IDispatch and "loose binding". It must be possible to use the interface from scripting languages which support only dispatch binding, and it must be possible to implement the interface with a Windows Script Component ("scriptlet"), which cannot expose a VTBL.
6. All properties, method parameters, and method return values must be Automation-compatible types such as INT, LONG, DOUBLE, SINGLE, BSTR, DATE, VARIANT_BOOL, and VARIANT/VT_DISPATCH. Any arrays produced and consumed by your product must be SAFEARRAY of VARIANT. In short, all data items must be 100% compatible with Automation, and specifically with ActiveX Scripting engines including both VBScript V5 and JScript V5 (or later) and with Visual Basic for Applications V5 (or later).
7. Method parameters must be passed only by value, as required by some ActiveX Scripting engines (notably JScript). Consider passing object references (VT_DISPATCH) by value as a way to have methods work on arbitrary (non-Automation) data owned by the client.

Client Program Guidelines

1. The product must run (at a minimum) on Microsoft Windows 7 with the latest service packs at the time of driver release. It should work on both 32- and 64-bit systems. Platform 6 does not support Windows 2000.
2. The product must be capable of using scriptable dispatch ("automation") interface(s) via the Microsoft Component Object Model (COM). The client must be able to call via IDispatch.
3. Error exceptions (raised in a component via IErrorInfo) must be caught and handled by the program in a way that gives the user the error message that came from the component. It is not permitted to display a "friendly", "generic", or otherwise mangled version of an error message in the exception.

Logo Usage

If you have a driver or an astronomy product that conforms to the ASCOM Interface requirements, feel free to use the logo on your web site and product packaging, as long as you do the following:

1. If you use the logo on a web site, please link it back to this site https://ascom-standards.org/
2. Post a note to ASCOM-Talk indicating your product, company, and URL. The moderator or other responsible person will add you to the partner's page and link to your web site.

Logo usage is on the honor system, there are no contracts or other covenants required. Please don't undermine this effort by ASCOM-labeling software that doesn't meet the requirements. Make the effort and your software will be better for it!