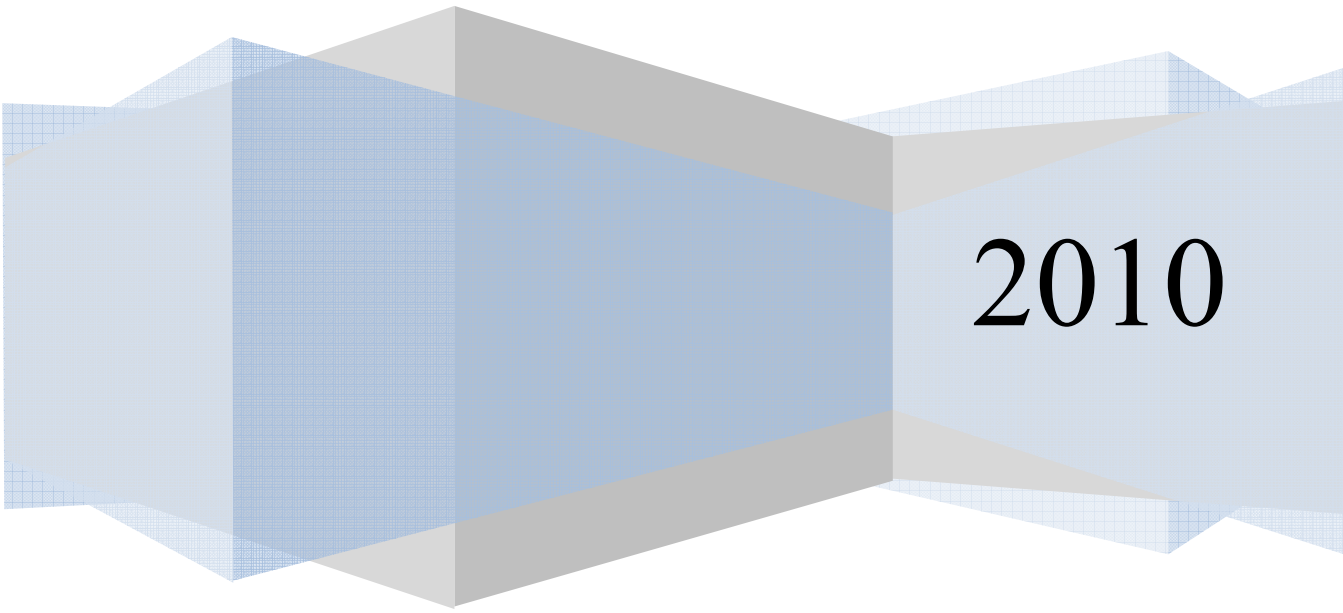
– Standards for Astronomy

Users Guide

Rob Morgan



2010

Table of Contents

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.

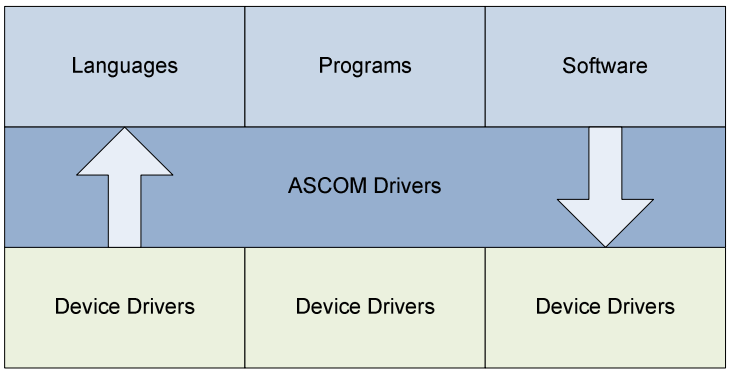


Figure 1

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.

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 required properties and methods that ASCOM compliant software can use to communicate with an ASCOM compliant device. ASCOM also defines a range of optional properties and methods to take advantage of common features that may not be available for every manufacturer's device. By testing various properties an ASCOM client application can determine what features are available for use.

Properties and methods are accessible via scripting interfaces, allowing control of devices by standard scripting applications such as VBScript and JavaScript. In

fact any language that supports access to Microsoft COM objects can interface with ASCOM. Figure 2 shows a breakout of the typical layers used by the Clients, ASCOM, Manufacturer Driver, and the hardware devices.

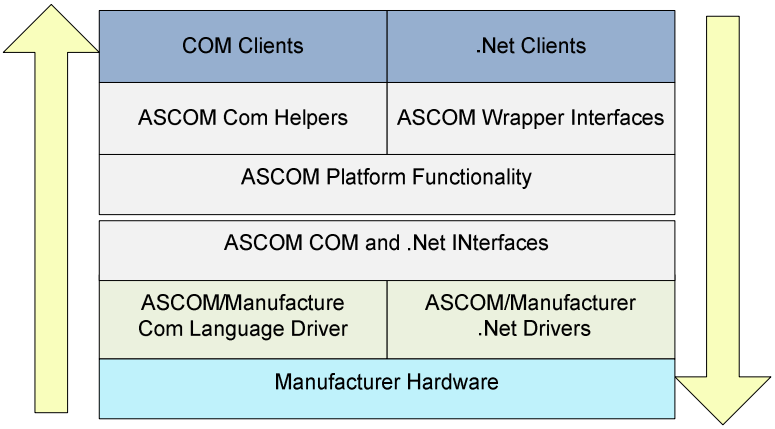


Figure 2

The ASCOM initiate is dedicated in keeping as much compatibility with Microsoft's Component Object Model (COM) as possible. This allows a verity of languages to be compatible with the ASCOM architecture. This book can't possibly show all examples for all the languages so we've select one compiled language, C# and one scripting language, VBScript for the examples.

Why Use COM?

COM is built into Windows. Any language can use COM like it can display on the screen or write to a disk file. COM isn't an I/O service though, it is a Component Object service (hence the name which stands for Component Object Model). Components are a special type of object. 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. They exist apart from the app's code, and are served by the operating system. Once loaded, though, 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 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. Once the component is activated, its

constructor is called. 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 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.

Who uses ASCOM

In general there are three types of actors, or groups of people playing as one role, that use the ASCOM Platform:

1. Client Developers whom create user interfaces for users such as TheSky, Starry Night, SkyMap, and Cartes du ciel.
2. Users that run the client software and load the ASCOM platform.
3. Driver Developers or manufactures that write the drivers for ASCOM and manufacture their hardware devices like Mounts, Telescopes, Focusers, etc...

A use case defines the interactions between external actors and the ASCOM Platform under consideration to accomplish a goal. An actor specifies a role played by a person or thing when interacting with the Platform. The same person using the system may be represented as different actors because they are playing different roles. Figure three depicts the most common of use cases for the ASCOM Platform.

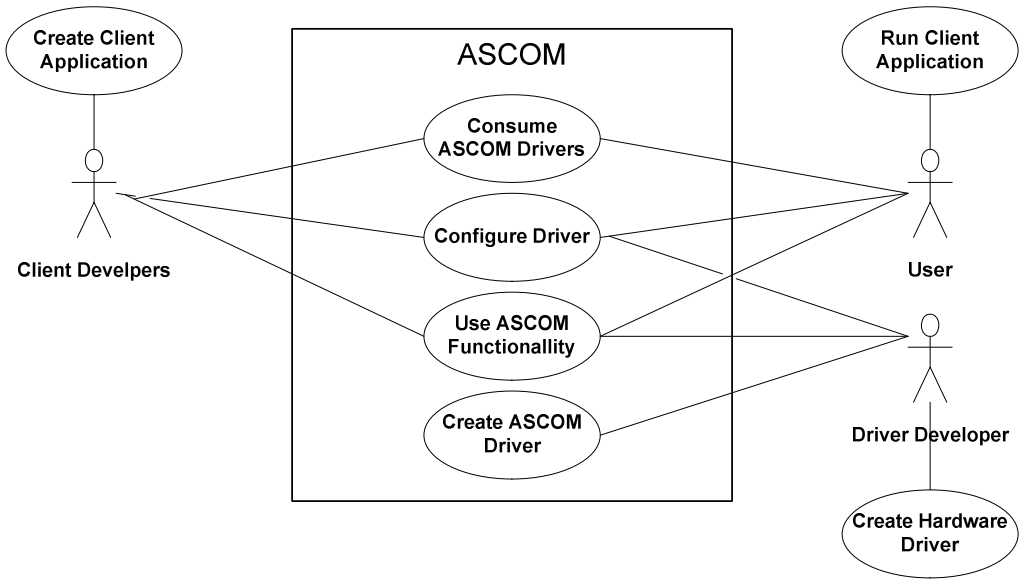


Figure 3

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. The activity diagram in Figure four depicts the flow of a typical use case. The client

application can call the ASCOM Chooser at anytime to select the appropriate device driver for configuration. Don't worry about knowing what the Chooser is right now. Just think of it as a way to select a device or driver. The client may at anytime call the ASCOM interfaces and use the device interface or other ASCOM functionality. This figure also shows that the Client application may optionally use the Manufacturer Hardware Driver at any time without using the ASCOM components. It would be up to the manufacturer to expose any additional functionality not used in the ASCOM standards. It is also optional for the manufacturer to implement their driver before the hardware and after the ASCOM driver.

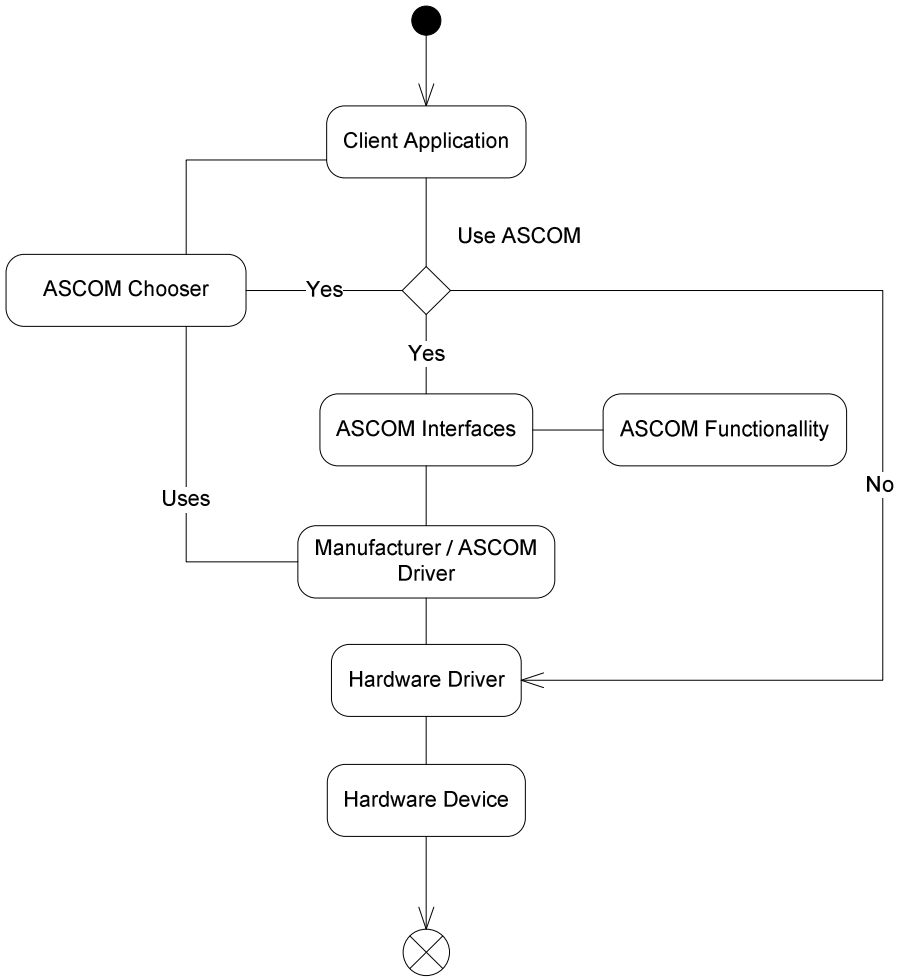


Figure 4

Choosing and Configuring the Driver (one-time setup)

Since all focusers look the same to the application, it first has to give the user a chance to select the MicroGlide as the type of focuser he's using. 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, MicroGlide, from a list. Once chosen, the user then clicks a Properties button. This loads the MicroGlide 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, he closes the config window and the Chooser. At this point, the MicroGlide driver saves the settings entered by the user then disappears from the system. Thus, the user's settings are remembered and need not be entered again unless something changes.

The Standards

The ASCOM Initiative has published a number of standards and guidelines that describe the ways in which the ASCOM Platform would interact with the Client Software and the Drivers the manufacturers write or create.

In order to be called "ASCOM compliant", a driver, component, or application scripting interface must meet all of the applicable guidelines and standards. Only then drivers, interfaces, or a component’s packaging and user interface, carry the ASCOM logo.

Driver Guidelines

1. The driver must install and run 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. Support for Windows 2000 is deprecated and will go away in Platform 6.

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 serial port I/O. for 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.

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 manufactures and be marked for ASCOM compatibility, but if not, you can probably get it from the here http://ascom-standards.org/Downloads/Index.htm.

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.

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. Support for Windows 2000 is deprecated and will go away 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 scripter's 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 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. Support for Windows 2000 is deprecated and will go away in Platform 6.

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. Beginning with Platform 2008, drivers may choose to support early binding by implementing the standard interface. In this case, clients may choose to reference the interface by calling QueryInterface on the instance of the driver.

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 these 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 http://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 above requirements. Make the effort and your software will be better for it!

The Standards Process

This is informal since relatively few people are involved and the astronomy community overall is relatively small. By avoiding the stilted and often political "standards body" approach, standards can be proposed, discussed, implemented, tested, refined, and accepted by vote more quickly and with a higher probability of success. See Philosophical Issues below. Loosely stated, the process is:

1. A single author publishes a draft interface specification. Ideally, this will be derived from an interface already in use and not designed in a vacuum. See Philosophical Issues below.

2. Discuss the proposal on ASCOM-Talk until an interface agreement can be reached. See Philosophical Issues below.

3. Implement a simulator which has all of the properties and methods of the proposed interface (a reference implementation). Ideally this would be done by someone other than the author. Make the simulator available to anyone who wishes to play with it.

4. Refine the specification and simulator as dictated by experience, again reaching an interface agreement brokered by the author. Discussion is closed at this point.

5. The author posts a poll on ASCOM-Talk, giving the community several weeks to vote yes or no. Further suggestions and other feedback will be rejected at this point.

6. If the majority votes yes, the specification is considered "adopted" and the author is responsible for writing the final standard document. If not, go back to step 4 or drop the spec entirely and possibly start over.

The most important goal of the standards process is to avoid the "design and decree" process that has caused so many failures and financial damage in the past. Typically employed by academics, design-and-decree just plain doesn't work. Professional engineers know it's essential to prototype, refine, and plan to throw the first one away or maybe start over.

Another important aspect of the standards process is that the author is responsible for brokering the interface agreement, a difficult task requiring sensitivity and above all the strength to reject "it would be nice if" suggestions which have no clear use-case.

Core Components

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

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

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

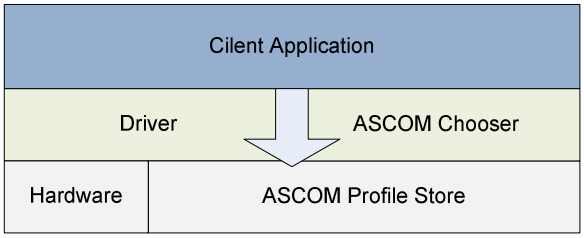


Figure 5

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.
Support includes the following…

- Camera
- Dome

- Filter Wheel
- Focuser
- Rotator
- Switch
- Telescope

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.

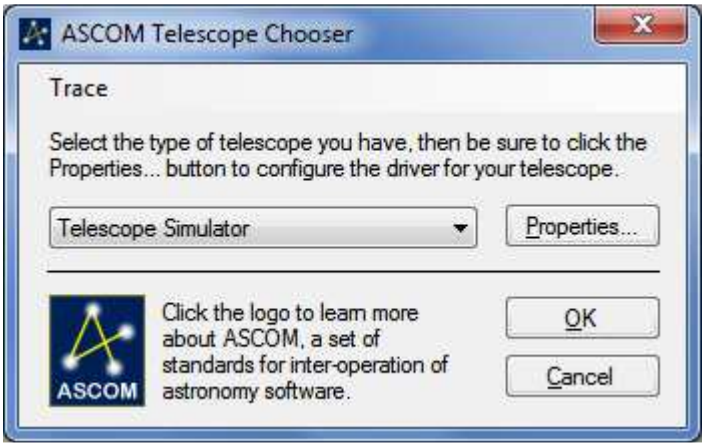
ASCOM.IConform - Driver interface to inform Conform of valid driver commands and returned error codes.

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).

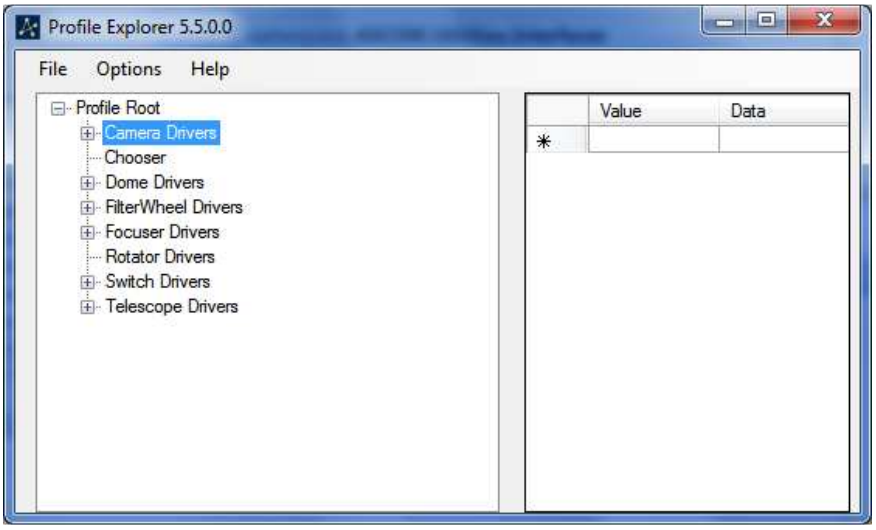
ASCOM.SettingsProvider - SettingsProvider integrates the ASCOM Profile store with the Visual Studio settings designer and the application settings architecture. It is intended for use by driver developers and is incorporated into the VS template projects.

ASCOM.Utilities - Contains things like the Serial, Profile, Chooser and other items like date and time conversions,

> ***Chooser*** - The Chooser object provides a way for users to select a device to work with within an application. The Chooser can be configured to choose any ASCOM device type. The default is "Telescope", but you can change the Chooser.DeviceType property to something else (e.g. "Focuser") and the Chooser will then work for that device type.



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.



Serial – used to assist in com ports and serial communications.

Tools

Driver Templates - Driver templates come in four basic formats for projects in Visual Studio 2005 or later. Two formats are for the Driver projects in both C# and VB and two are in the Local Server formats for both C# and VB. Those individuals or companies wanting to write a driver for their hardware would use

one of the Driver projects for a single hardware interface or the Local Server if their hardware interface supports more than one driver on the same interface.

.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).

History

During 1998, DC-3 Dreams developed its Astronomer's Control Panel (ACP1), a Windows program that gave access to all of the capabilities of the Meade LX200 Classic. During the process, DC-3 Dreams focused on including an open Windows Scripting interface that included a Telescope object. Other programs could use this for a high-level way to control the LX200. As this project evolved, it became clear that ACP1 could be used for observatory automation if it could somehow control the CCD camera. DC-3 Dreams contacted Diffraction Limited and together they conceived of an open Windows Scripting interface to Diffraction's MaxIm DL/CCD program. With this key piece of the puzzle, DC-3 Dreams developed Windows Scripts that controlled both the telescope and CCD camera using ACP and MaxIm. This effort culminated in DC-3 Dreams showing the first off-the-shelf automated observing system at the Riverside Telescope Maker's Conference in the Meade tent on Memorial Day Weekend 1999.

As things progressed during 1999, Diffraction and DC-3 Dreams decided that astronomy software needed a driver/client architecture like the rest of the computing world. At that time, there were only a few astronomy software packages that could control telescopes, cameras, filter wheels, etc. All of them contained their own code for each of the devices they supported. This sort of architecture had long since been abandoned by the computing community in favor of the now-familiar driver-client architecture. Neither DC-3 Dreams nor Diffraction wanted to undertake writing their own built-in control code for the rapidly growing family of astronomy devices out there. Led by Bob Denny of

DC-3 Dreams, they undertook to establish the ASCOM Initiative to promote the driver-client architecture in astronomy software.

The Telescope Interface Standard

The next task was to develop the first driver interface standard, and the Telescope interface was chosen since it has the highest leverage. Since this interface had to be usable in a wide variety of applications (apart from ACP1 and MaxIm), the design was carefully refined over the span of a couple of years, with a heavy emphasis on real-world applications. During this period, Sienna Software (now Imaginova) joined the effort and provided a $10,000 grant to develop an initial set of telescope drivers for use with their Starry Night planetarium program. There were no strings attached to the drivers, so they could be used by anyone writing astronomy software that needed to control telescopes.

Also during this period, the ASCOM web site was launched, several other software vendors jumped on the bandwagon, and Sky & Telescope published an article on ASCOM. As experience was gained in practical applications, and the Telescope interface was nearing final form, a Telescope Driver SDK was developed. This was the genesis of the ASCOM Platform.

Initial Releases and Adoption

In mid-2001, the Telescope interface was finalized and adopted by the group. At the same time, the first ASCOM Platform was released. It contained the final version of the Telescope Interface Spec, the Telescope Driver SDK, a Telescope simulator, some driver support components, and the set of standard telescope drivers underwritten by the Sienna Software grant. During the remainder of the year 2001, two more Platform releases were made, each containing more telescope drivers. By now, even research-grade telescopes were included in those which had drivers. In addition, ACP and MaxIm DL/CCD joined Starry Night in using standard telescope drivers.

The Focuser Interface Standard

In late 2000, Doug George of Diffraction Limited proposed a Focuser Standard. Eventually, in early 2002, the ASCOM Standard Focuser interface was finalized. Drivers for various focusers followed.

FocusMax

Also during late 2000, Larry Weber and Steve Brady developed their FocusMax auto-focus software. FocusMax is perhaps the most significant advance in

automated observing ever. It provides robust automatic focusing for a virtually infinite number of combinations of telescopes, CCD cameras, and focusers. They both asserted that FocusMax would not have been possible without ASCOM telescope drivers and MaxIm DL's Windows Scripting interface. Weber and Brady participated heavily in the discussions and refining of the Focuser Standard.

ASCOM Platform 2.0

Following on the heels of the Focuser interface standard release, the ASCOM Platform 2.0 was released in September, 2002. This Platform contained the new Focuser standard as well as the first set of focuser drivers, developed by various authors. Per the Initiative goals, these drivers were available for anyone to use. During the latter part of 2002 and much of 2003, additional Platform 2.x releases were made, each with more Focuser and Telescope drivers, as well as a new Focuser simulator that serves as the reference implementation of the standard Focuser interface.

The Dome Interface Standard

Also during 2002 and 2003, discussion began toward a dome control driver interface. Robotic control of telescopes enclosed in a dome clearly needed to be tied to the dome rotation, and control of the dome shutter(s) was clearly needed for weather safety during automated operation. A rather large group of people became involved in discussing the dome interface, and the discussion became contentious at times. The goal of eliminating device-specifics from the interface appeared out of reach given the variety of "shutter" configurations in use. An elegant solution was finally reached by including both azimuth and altitude inputs for shutter positioning, leaving the dome controller to simply "make a hole big enough to see through". The standard was finally adopted in August 2003.

It should be mentioned that John Oliver of the University of Florida contributed a vital piece of the puzzle by implementing Chris Lord's dome pointing equations in Visual Basic 6. Calculating the dome's azimuth and altitude needed for that "hole to the sky" for German equatorial and fork mounts is a difficult task. John's code formed the basis of several programs that use dome drivers, and we all owe him a debt of gratitude.

ASCOM Platform 3.0

In October 2003, the ASCOM Platform 3.0 was released. This Platform contained the new Dome standard, a dome simulator and a generic telescope/dome controller and hub. This latter component allowed any astronomy program that used ASCOM telescope drivers to instantly be able to control the combination of the telescope and dome without any changes to the astronomy program itself. ASCOM was clearly coming of age by this time.

The Telescope V2 Interface

During 2003, it was becoming clear that the Telescope Interface Standard needed to grow. A rather large group of application and driver authors participated in discussing additions to the Telescope interface. Additions included control of guiding rates, a new ability to move about the mount's axis, and better pier-side monitoring and control. The interface was adopted in April, 2004.

ASCOM Platform 4.0

In December 2004, the ASCOM Platform 4.0 was released. This Platform contained the new Telescope V2 interface spec, updates for many drivers, and a few new drivers. An update, Platform 4.1 was released about six months later.

Stability and Maturity

The release of Platform 4 marked the beginning of a long period of stability for ASCOM. The Platform 4.1 remained the standard for several years, though individual drivers were updated during this period. by this time, there were nearly a thousand people on the ASCOM Talk group, and over a dozen people had developed drivers and programs that used them. The big issue was the lack of commitment by device manufacturers to provide drivers and support with their devices.

ASCOM Platform 2008 (5.0)

By mid 2006, it was becoming clear that the ASCOM Initiative needed to address several issues:

- Including drivers in the Platform was becoming cumbersome.
- Distribution of drivers not in the Platform (and updates to drivers in the Platform) was haphazard.
- Visual Basic 6, the preferred tool for driver development, was being replaced by the Microsoft .NET family of languages and tools.

- The loose/late binding scheme used for client-driver communication was difficult to use from .NET languages, and imposed an overhead penalty for them.

These issues led to the following changes for Platform 2008:

- Drivers are no longer being included in the Platform. Instead, drivers are to be provided by the device manufacturer or via a central download place on the web.
- Driver authors need to deliver their drivers self-installable. To facilitate this, a new tool was developed that generates driver-specific installer scripts for the free InnoSetup system. This eliminates the need for driver authors to spend money for installer tools and minimizes the learning curve.
- A set of Visual Studio 2005 templates for all driver types are included, making it easy to write robust and complete drivers in C#.NET and VB.NET.
- A set of abstract interfaces are provided for both native COM and via Primary Interop Assemblies, for .NET. These interfaces allow a driver to expose both late and early binding interfaces.
- A Visual Studio 2005 client-side driver interface toolkit is provided to allow client astronomy programs to make use of drivers whether or not they exposed the relevant early-binding interface. With the toolkit, IntelliSense for driver members is available at all times, and early binding is automatically used if available from the driver.

ASCOM Platform 2008 was released in February, 2008. There are over 1,700 members on the ASCOM-Talk list!

Hall of Fame

Many people have participated significantly by writing drivers and tools, providing useful information, and helping with interface agreements, etc. The following people have made particularly significant contributions to the ASCOM Initiative. They are listed in no particular order:

- ***Tom Andersen and Ted Leckie*** - For seeing the logic in ASCOM and the driver/client architecture back in 1998, and providing a grant to develop the first set of drivers and simulators.

- ***Doug George*** - For "getting it" right back in 1999 and adding a non-trivial scripting interface to MaxIm DL immediately thereafter. He also authored the Camera. Focuser, and FilterWheel interfaces, basing them on real-world experience with MaxIm.
- ***Tim Long*** - For all of his work setting up and hosting the open-source environment, pushing forward with .NET, chairing our weekly platform meetings, writing drivers and tools, helping with .NET issues, and being a moderator for the ASCOM Talk group.
- ***Peter Simpson*** - For an enormous amount of work creating the new HelperNET components, much of Platform 5.5, weighing in on strategic Platform issues, the absolutely vital driver conformance checker, and lots of professional-class testing, particularly of Platform 5.
- ***Larry Weber and Steve Brady*** - for their fabulous FocusMax autofocus software, evangelizing ASCOM as being absolutely necessary for FocusMax's success, and for driving the Focuser standard.
- ***Chris Peterson*** - For being a very knowledgeable curmudgeon, keeping us honest, and helping people with concise and correct information. Chris' participation in (and inputs to) our weekly meetings have been invaluable.
- ***Jonathan Fay*** - For the Worldwide Telescope which puts Microsoft behind ASCOM and for the .NET Client Toolkit that provides .NET client software with automatic easy access to both early- and late-bound drivers.
- ***Jeff Dickerman***, president of Optec, Inc. - For having the vision and commitment to provide factory-developed drivers for all of their equipment.
- ***Mark Crossley*** - For the FilterWheel Simulator and his helpful inputs during shakeout of the FilterWheel spec.
- ***David Challis*** - For believing in ASCOM strongly and making Quantum Scientific cameras ship with factory-supplied drivers.
- ***Jon Brewster*** - For developing several drivers and simulators, the current Meade drivers, POTH, Pipe/Hub, and generally being a guiding light. He even went to Meade one time to get the "low down" for the Meade drivers. His crystal-clear headed approach to engineering was a great help on many occasions.
- ***John Oliver*** - For his work on the dome geometry code used in POTH and by several other programs, and for his role in the Dome interface agreement.
- ***Dan Azari*** - For authoring the Rotator interface, and for RCOS shipping factory-supplied drivers for their Focuser and Rotator.
- ***Joe Shuster*** - For quietly doing the lion's share of moderating the ASCOM-Talk forum, and being a clear-headed engineer with a great head for interface negotiation and design.
- ***Jim Moronski*** - For having Finger Lakes Instruments cameras ship with factory-supplied drivers

- ***Ray Gralak*** - For constantly grinding on us to address the Macintosh OS and Linux, and providing useful suggestions in that vein.
- ***Chris Houghton*** - For shipping the Maestro telescope control system with factory-supplied drivers
- ***Chris Rowland*** - For developing the incredibly complex Celestron unified driver, tools for extending interfaces in VB6, rigorous testing, and always providing clear and unwavering advice and counsel.
- ***Ajai Sehgal*** - For putting a ton of work into the high-profile Astro-Physics and Gemini drivers, and supporting them in the face of firmware changes beyond his control.
- ***Stef Cancelli*** - For the graphical and logic design of this new ASCOM web site. It might not seem that way, but it took a lot of his time to do it.
- ***Peter Eschman*** - For the original Driver Checker tool.
- ***Bob Denny*** - For many things...