Introduction to INDI

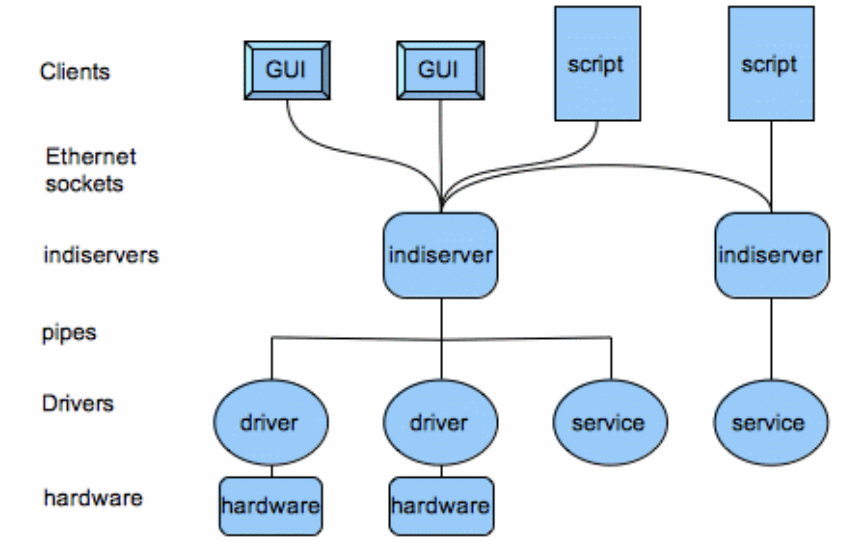
Specification

INDI stands for *Instrument Neutral Distributed Interface*. The definitive specification is available here [http://clearskyinstitute.com/INDI/INDI.pdf]. The core of INDI is the idea that all information is captured in *Properties*. Properties are small typed XML packets that contain anything from telescope azimuth to current wind direction to a camera image.

Information about INDI specifically at LBTI is available here .

Architecture

The diagram on the right shows the basic architecture of an INDI control system.



Start at the bottom to see several individual INDI Drivers. Drivers bridge the details of a device to the world of INDI Properties. Some drivers provide Properties related to a particular hardware device. But Drivers may also offer abstract services, such as scheduling or target prediction, so they are not limited to hardware interfaces. Think of an INDI Driver then as an interface to any kind of resource that is useful to an observatory.

Next skip to the top and notice the boxes representing INDI Clients. An INDI Client is any process that wishes to utilize the Properties provided by INDI Drivers. Clients can be rich GUI applications, interactive command line tools or scripts of coommands. They may be written in any language, from GUI's written in Java or Qt, to scripts written in shell, Python or Perl. The only requirement they share is a need to communicate with INDI Properties to access and control the observatory services they require.

In the middle, between Clients and Drivers, is the INDI Server. Think of this as an intelligent router. It filters and directs INDI messages between Clients and Drivers so only those Properties for which they have an interest are sent to a given Client or Driver, thus eliminating extraneous traffic. The INDI Server does not otherwise interpret the INDI messages it handles and thus it can operate in a very efficient yet generic manner.

The connection between an INDI Server and an INDI Driver is simply the stdin, stdout and stderr channels common to all POSIX processes. This arrangement allows for easy debugging during Driver development. The INDI Server forks each Driver and rearranges file descriptors for all subsequent communications. The INDI Server also serves as a watchdog by automatically restarting any Driver if it should fail for any reason.

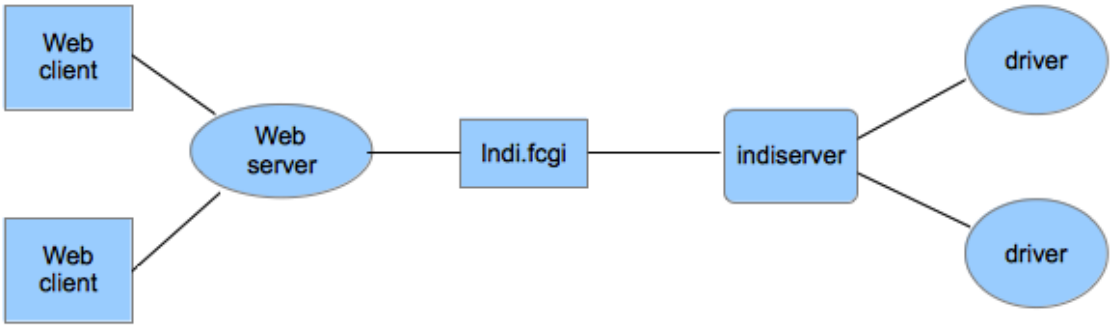
The connection between an INDI Server and an INDI Client is a tcp/ip socket on a well-known port, 7624, which has been officially recorded with IANA [http://www.iana.org]. This allows complete worldwide network interoperability among Clients and Servers. Although INDI itself does not address security, secure connections from Clients to Servers can be accomplished by simply using ssh tunneling of this port. GUI clients often contain this ability internally without need for separate tunneling setup. A VPN can also be used if the participating nodes is relatively stable. Using tcp/ip also provides for a heterogeneous computing environment, for example Clients might be running on Windows and Drivers might be running on Linux; the only requirement is that all message traffic adhere to the INDI protocol.

It is useful to note that it is possible to *chain* together multiple INDI Servers. Since the INDI protocol is nearly symmetric, an INDI Client may in fact be another INDI Server. This can be useful in order to deploy an INDI Driver on a machine optimized for a piece of hardware, such as a camera, and yet provide the functionality of that Driver seamlessly on a wider INDI network. Note also that each INDI Server can be configured as to the INDI Drivers for which it allows communications. This allows, for example, certain highly critical Drivers to be rendered invisible to

outward facing nodes on an INDI network. Embedded INDI servers have also been developed that can be chained into an INDI network seamlessly.

INDI on the WWW

By creating a simple gateway process and using the W3 fcgi [http://www.fastcgi.com] protocol, it is easy to bring the INDI Property model to modern web browsers. The basic topology is described in the following diagram:



We see that any standard collection of indiservers, together with its drivers and any chained indiservers, is bridged through an fcgi helper app named *indi.fcgi*. This is a simple gateway program that just connects to an indiserver as if it were a client, and passes *set* and *get* requests from and to the web page. Javascript [http://www.w3schools.com/js] and jquery-ui [http://jqueryui.com] match very well to this model. The indi.fcgi gateway can also be configured as read-only such that only *get* INDI messages are allowed to flow towards the web browser, any *set* messages from the web browser are discarded and never sent to the indiserver.

Logging and configuration files

The INDI Server produces complete logs of all activities from Clients and Drivers. The level of detail can be configured to four levels, from simple connection requests to full message protocol details. Drivers can also add their own messages to these logs. The logs are stored in simple ASCII format and each entry is automatically time stamped and marked as to the Client or Driver of origin.

Indi provides drivers with a standardized framework for storing configuration parameters. This allows a well-written driver to be used in more than one application with little or no coding changes.

Community

INDI enjoys an active user community based at http://indilib.org [http://indilib.org]. A variety of telescopes, cameras, motion controllers, domes and other devices already have INDI Drivers. New drivers can be developed in a straight forward manner against a simple API. Generic GUI Property browsers are also available or rich custom GUI clients can be created using a standard framework available for Java and Qt that are fully interoperable on Windows, Mac OS X and Linux. There is also a standard suite of command line tools. These *set* or *get* atomic INDI properties and can be utilized by any scripting language to create arbitrarily complex applications written entirely by knowledgeable users on top of the core observatory infrastructure. The tools can be scripted in python, bash, csh, perl or tcl to name a few. There is also a robust and flexible *schedular* application available that can manage multiple disparate collections of Property operations optimally over the course of time based on a wide range of generic and astronomical constraints.