INDI Developers Manual

Jasem Mutlaq
Elwood Downey

Rev 0.9
June 2004

Table of Contents

1. Introduction

The Instrument-Neutral-Distributed-Interface control protocol (INDI) is a new key technology for device automation and control. INDI introduces a common universal standard for developing robust, adaptive, and scalable device drivers under several platforms.

The INDI Library aims to create a solid driver support for a wide range of instruments used both in amateur and professional astronomy. The current INDI repository contains support for several popular telescopes, focusers, CCD cameras, and video capture devices.

INDI has many advantages over similar technologies, including loose coupling between hardware devices and software drivers. Clients that use the device drivers are completely unaware of the device capabilities. In run time, clients discover the device capabilities through introspection. This enables clients to build a completely dynamical GUI based on services provided by the device. Therefore, new device drivers can be written or updated and clients can take full advantage of them without any changes on the client side.

Furthermore, remote control of devices is seamless with INDI's server/client architecture. Distributed devices can be controlled from one centralized environment.

2. Purpose of Manual

The purpose of this manual is to enable programmers to take full advantage of the INDI architecture, by providing a clear and concise guide illustrating the different aspects of the control protocol. The manual covers the architecture of INDI in detail, and provides the latest Application Programming Interface (API) for INDI wire protocol version 1.2.

The manual does not cover the development of INDI clients. Client applications can be very diverse. They can range from simple loggers to complicated automated scripts. Programmers who are interested in client development may want to check out existing clients like KStars and Xephem.

The INDI Library is released under the GNU Library General Public License (LGPL).

3. Intended Audience

The INDI protocol is geared toward experienced programmers planning to develop back-end hardware drivers to run under the INDI architecture. The task of developing hardware drivers requires programmers with sufficient experience on least one high level programming language such as C/C++.

While the INDI Wire protocol is platform-independent, the INDI Library v0.2 is designed to operate on UNIX/Linux platforms. Depending on the nature of a particular device, developers may need to interact with the Linux kernel and its underlying architecture. Developers can port the library and device drivers to different platforms as desired.

4. How to use this manual

This manual is a practical guide to developing device drivers under INDI. A clear understanding of the design and philosophy of the INDI architecture is required before developing any INDI drivers. Elwood Downey, author of the INDI wire protocol, maintains an up to date INDI White paper

<http://www.clearskyinstitute.com/INDI/INDI.pdf>

You should read the white paper as it serves the foundation of this manual. The following sections provide a brief introduction to the INDI architecture, current supported devices in INDI library, and the major components of the INDI library.

The manual makes several references to INDI tutorials, which are packaged with the official INDI Library release. Refer to section 9 for more details.

5. Introduction to the INDI Architecture

INDI is a simple XML-like communications protocol described for interactive and automated remote control of diverse instrumentation. INDI is small, easy to parse, and stateless. In the INDI paradigm each Device poses all command and status functions in terms of settings and getting Properties. Each Property is a vector of one or more names members. Each property has a current value vector; a target value vector; provides information about how it should be sequenced with respect to other Properties to accomplish one coordinated unit of observation; and provides hints as to how it might be displayed for interactive manipulation in a GUI.

Figure 1 illustrates a simple INDI configuration. INDI does not pose any restrictions on the methods clients employ to represent data. Possible clients include simple loggers, GUI clients, and complex automated scripts.

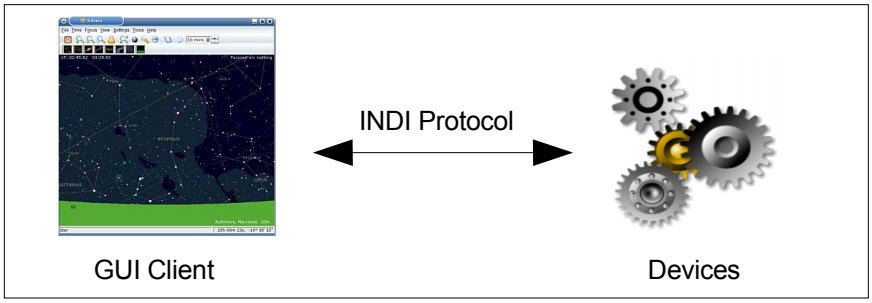

Figure 1. Simple INDI Configuration

Clients learn the Properties of a particular Device at runtime using introspection. This decouples Client and Device implementation histories. Devices have a complete authority over whether to accept commands from Clients. INDI accommodates intermediate servers, broadcasting, and connection topologies ranging from one-to-one on a single system to many-to-many between systems of different genre.

The INDI protocol can be nested within other XML elements such as constraints for automatic scheduling and execution.

6. Supported Devices

The INDI Library currently supports the following devices:

Telescopes:

- LX200 Generic.
- LX200 Autostar.
- LX200 16". LX200 Classic.
- LX200 GPS. Celestron GPS.
- LX200 Compatible Devices (Astro-Physics AP, Astro-Electronic FS-2, Mel Bartels Controllers...etc).

Focusers:

- Meade LX200GPS Microfocuser.
- Meade 1206 Primary Mirror Focuser.
- JMI NGF Series.
- JMI MOTOFOCUS.

CCDs:

- Finger Lakes Instruments CCDs.

Video Capture:

- Any Video4Linux compatible device.
- Phillips Web cam.

7. INDI Library Components

The INDI Library consists of four main components:

1. INDI Core Components: This includes the INDI server and the INDI API required to develop device drivers. The API reference follows INDI wire protocol v1.2

2. The Little XML library reference: A simple library to parse and process XML.

3. FITS library**:** A simple FITS library for reading and writing FITS files.

4. MiniZLO**:** A small and fast lossless compression library used to transfer binary data to INDI clients.

7.1 INDI Server

INDI server is the public network access point where one or more INDI Clients may contact one or more INDI Drivers. Indiserver launches each driver process and arranges for it to receive the INDI protocol from Clients on its stdin (standard in) and expects to find commands destined for Clients on the driver's stdout (standard out).

Anything arriving from a driver process' stderr is copied to indiserver's stderr (standard error). Indiserver only provides convenient port, fork and data steering services. If desired, a Client may run and connect to INDI Drivers directly.

Typical INDI Client / Server / Driver / Device connectivity is illustrated in Figure 2:

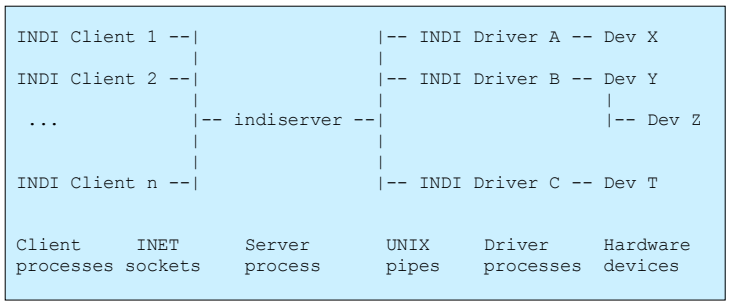

Figure 2. End to End Connectivity

The syntax for INDI server is as following:

$ indiserver [options] [driver ...]

Options:

-p p : alternate IP port, default 7624

-r n : max restart attempts, default 2

-v : more verbose to stderr

For example, if you want to start an INDI server running an LX200 GPS driver and listening to connections on port 8000, you would run the following command:

$ indiserver -p 8000 lx200gps

7.2 Driver Construction

An INDI driver typically consists of one or more .c file (e.g. mydriver.c) which includes indiapi.h and indidevapi.h to access the reference API declarations.
It is compiled then linked with indidrivermain.o, eventloop.o , liblilxml.a, and (optionally)

indicom.a to form an INDI process. These supporting files contain the implementation of the INDI Driver API and need not be changed in any way.

Note that evenloop.[ch] provide a callback facility independent of INDI which may be used in other projects if desired. The driver implementation, again in our example mydriver.c, does not contain a main() but is expected to operate as an event-driven program.

The driver must implement each ISxxx() function but never call them. The IS() functions are called by the reference implementation main() as messages arrive from Clients. Within each IS function the driver performs the desired tasks then may report back to the Client by calling the IDxxx() functions.

The indicom.a library contains a list of common astronomical and parsing routines used across different INDI drivers. The reference API provides IE() functions to allow the driver to add its own callback functions if desired. The driver can arrange for functions to be called when reading a file descriptor that will not block; when a time interval has expired; or when there is no other client traffic in progress. Several utility functions to search and find various INDI vector structs are provided for convenience in the API.

The sample indiserver is a stand-alone process that may be used to run one or more INDI-compliant drivers. It takes the names of each driver process to run in its command line arguments. Once a binary driver is compiled, indiserver can load the driver and handle all data steering services between the driver and any number of clients.

7.3 INDI Standard Properties

While INDI clients can build GUIs to represent all device properties, direct human intervention is required to control the device. To enable complete automation of astronomical devices, drivers and clients alike need to be aware of the INDI Standard Properties (ISP).

The ISP is a list of common astronomical properties that are critical to the operation of many devices. For example, if a client needs to move a telescope to a new position, it can search and modify the EQUATOORIAL_COORDS vector property (J2000 geocentric equatorial coordinates). The imposed semantics insure interoperability across all INDI clients and devices (see Tutorial 1, 2).

The following is a list of reserved INDI Standard Properties as of INDI Library v0.2:

Property Name	***Type***	***Member***	***Description***
EQUATOORIAL_COORD	Number		Equatorial astrometric J2000 coordinate
		RA	J2000 RA, hours
		DEC	J2000 Dec, degrees +N
GEOGRAPHIC_COORD	Number		Earth geodetic coordinate
		LATITUDE	Latitude, degrees +N
		LONGITUDE	Longitude, degrees +E
HORIZONTAL_COORD	Number		Topocentric coordinate
		ALTITUDE	Degrees above horizon
		AZIMUTH	Degrees E or N
ABORT_MOTION	Switch		
		ABORT	Stop rapidly but gracefully
ON_COORD_SET	Switch		Action device takes when sent any *_COORD property.
		SLEW	Slew to a coordinate and stop
		TRACK	Slew to a coordinate and track
		SYNC	Accept current coordinate as correct
CONNECTION	Switch		
		CONNECT	Establish connection to the device
		DISCONNECT	Terminate connection to the device
TIME	Text		
		UTC	UTC Time in ISO 8601 format
DEVICE_PORT	Text		
		PORT	Port name the device is connected to
PARK	Switch		Park the telescope
SDTIME	Text		
		LST	Local sidereal time
DATA_CHANNEL	Number		
		CHANNEL	Port number of the binary data transfer channel
VIDEO_STREAM	Switch		
		ON	Turn on video stream
		OFF	Turn off video stream
IMAGE_SIZE	Number		
		WIDTH	Image width in pixels
		HEIGHT	Image height in pixels

Property Name	*Type*	*Member*	*Description*
MOVEMENT	Switch		Compound directions at the current slew speed
		NORTH	Move to the north
		WEST	Move the the west
		EAST	Move to the east
		SOUTH	Move to the south
EXPOSE_DURATION	Number		
		EXPOSE_S	Expose the CCD chip for EXPOSE_S seconds

7.4 Binary Data Transfer

If a driver needs an additional channel for binary transfer (e.g. to transfer video streams or regular files), it must establish a separate port for such service (see Tutorial 3). The driver must insure that the additional binary channel is operational for any number of potential clients. To utilize any type of binary transfer, clients must be aware of the binary transfer content and any associated semantics. This can be accomplished by defining additional INDI properties to describe the binary data.

Binary data transfer is accomplished using a simple custom protocol:

1. The driver defines an INumberVectorProperty with an ISP name of DATA_CHANNEL. The one INumber member name is CHANNEL. The initial value of CHANNEL is zero.

2. The driver establishes a server and selects a free port to listen to. The CHANNEL numeric value is then set to the port number.

3. The driver sets the status of the DATA_CHANNEL property to IPS_OK, which signals the client that the driver is ready to receive client connections on the data port.

4. Upon a successful connection, the driver can send any arbitrary type of data to the client.

For our discussion, we will define an INDI *frame* as an atomic chunk of data sent by the driver to its clients.

INDI *frames* must be compressed using the minizlo library first. The minozlo is a very small and fast lossless library (see Tutorial 3). It must be used in the driver to compress data and in the client to decompress data.

Three mandatory pieces of information are attached to each *frame* header. The header fields are encapsulated in XML. The fields are as following:

- Data Type: The data type is a string describing the content of the frame. This is usually a type that a client can process (e.g. FITS). If the client cannot process the data type, it may save the data on the disk as a regular file, with the data

type string as the file extension. For example, if the data type was "PNG" and the client does not recognize "PNG", it can save the data to disk using a time-stamped name with PNG as its extension (e.g. File-2004-06-04T12:14:04.PNG). The client is *not* required to save any unrecognizable data, it can silently ignore them. Some clients, like KStars, can process video streams and FITS file using the standard VIDEO and FITS data types (see Tutorial 3).

- Uncompressed frame size: Uncompressed *frame* size in bytes. This enables clients to to allocate memory resources for the decompression process.

- Compressed frame size: Compressed *frame* size in bytes. This is the size of the actual binary data transfered over the data channel to the client.

The header is flexible enough to allow drivers to add more fields as needed. All clients that support binary data transfer must support the three basic obligatory fields at minimum

The DTD of the header is as following:

```
<!ELEMENT Data>
<!ATTLIST Data
  type       %data type;                    #REQUIRED
  size       %uncompressed size in bytes;   #REQUIRED
  compsize   %compressed size in bytes;     #REQUIRED
>
```

Here is an example of a simple header:

```
<Data type='FITS' size='1024' compsize='128' />
```

The data type is FITS; the uncompressed size is 1024 bytes; and the compressed size (the data being transfered) is 128 bytes.

8. INDI API Reference

The INDI Reference is maintained on the INDI Sourceforge site.

<http://indi.sourceforge.net/doc/>

9. INDI Tutorials

The INDI tutorials are included in the official INDI Library v0.2 distribution from sourceforge.

<http://indi.sf.net>

The tutorials will aid you to apply the knowledge gained from this manual to design and develop device drivers that can run under any INDI compatible client.

The tutorials are located in the src/examples directory. Refer to the README file for detailed instructions on building and running the tutorials as device drivers.