

Scheduling, Automation, and Control in Astronomical Instrumentation

Jasem K. Mutlaq

Kuwait National Radio Observatory - Kuwait Science Club

Feb 2007



ABSTRACT

Heavy computational and acquisition demands on modern astronomical infrastructure require the coordination of numerous devices spreading over systems of different genre and widely separated stations. The instrument-neutral-distributed-interface (INDI) is a powerful scalable protocol for distributed control systems (DCS) aimed at reducing costs, accelerating development, and boosting reliability by providing a flexible generic schema suitable for the control of canonical and non-canonical devices. Due to the extensible nature of INDI, scheduling and automation of astronomical instruments integrates seamlessly with any XML-based framework.

Introduction

Traditional control architecture is very often customized for deployment on a specific software and hardware setup, whereas the software and hardware backends are tightly integrated and coupled to deliver the desired applications expected from any DCS system. Any deviation, including future upgrades of either the software or hardware infrastructure is usually prohibitively expensive and time consuming. Furthermore, users are locked into using a tightly coupled and intertwined system incapable of expanding to meet changing demands.

In conventional DCS architectures, when one or more of the core device parameters change, the software needs to be modified to accommodate the change. That is, the software *frontend* and hardware *backend* are tightly **coupled**.

INDI resolves this problem by providing a framework that decouples low level hardware drivers from high level front end clients. Clients that use the device drivers are **unaware** of the device capabilities. In run time, clients communicate with the device drivers and build a completely *dynamical GUI* based on the services provided by the device. In essence, INDI provides the capacity to describe *any device* and its parameters to any INDI-compliant client. Once a client is created, the user can develop any number of devices without any changes on the client side.

Elwood C. Downey of ClearSky Institute developed the Instrument-Neutral-Distributed Interface (INDI) in order to provide a generic framework for interactive and batched control of diverse astronomical instrumentation. Jasem Mutlaq is the lead developer and the current maintainer of *INDI Library*, a POSIX implementation of the control protocol.

Protocol

The INDI protocol defines the control language specifications applicable to INDI drivers and clients.

According to Elwood Downey:

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 setting and getting Properties. Each Property is a vector of one or more members. Each property has a current value vector; a target value vector that provides information about how it should be sequenced with respect to other Properties to accomplish one coordinates unit of observation; and provides hints as to how it might be displayed for interactive manipulation in a GUI.

The main key concept in INDI is that devices have the ability to describe themselves. This is accomplished by using XML to describe a generic hierarchy that can capture the features and functions of virtually any device. All *devices* may contain one or more *properties*. A property in

the INDI paradigm describes a specific function of the driver. Any *property* may contain one or more *elements*.

There are four types of INDI properties:

- **Text property:** Property to transfer simple textual information in ISO-8859-1. If the text includes elements that can break XML syntax, a BLOB property should be used instead.
- **Number property:** Property to transfer numeric information with configurable minimum, maximum, and step values. The supported number formats are decimal and sexagesimal. The property includes a GUI hint element in printf style format to enable clients to properly display numeric properties.
- **Switch property:** Property to hold a group of options or selections (Represented in GUI by buttons and checkboxes).
- **Light property:** Property to hold a group of status indicators (Represented in GUI by colored LEDs).
- **BLOB property:** BLOB is a **B**inary **L**arge **O**bject used to transfer binary data to and from clients.

Figure 1 illustrates a simple INDI configuration. INDI does not pose any policies on how clients represent data, but some properties include hints to clients as to how they should be represented. Possible clients include simple loggers, GUI clients, and complex automated scripts.

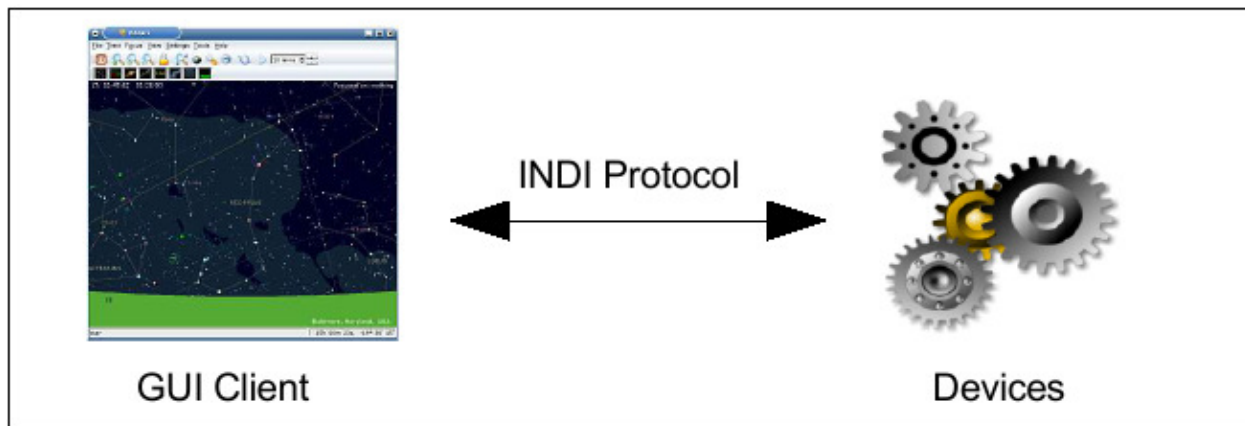


Figure 1: Simple INDI Configuration

Each device driver offers a list of properties that describes the device features and capabilities. Clients connect to the device driver via INDI server and query the device's properties using **getProperty** element.

Drivers respond by sending a list of supported properties in accord with INDI Document Type Definition (DTD). Each property vector may contain one or more elements, and each element type consists of a collection of data structures unique to each property type. In order to define new properties to clients, a driver writes a defTYPE element to the client without waiting for any acknowledgement from the client side. Clients should be designed to accommodate changes in the driver as the driver can define new properties, update existing properties, or attempt to delete properties at any time.

When a client wants to set a new value for a property, it writes a newTYPE element to the driver. When using an interactive real time client, the user can watch the status of the request's progress and condition. On the other hand, scripted clients are expected to watch and process the status and values associated with the request.

Drivers may define properties with respect to client as read-write (RW), read only (RO), or write-only (WO) property. Each property has one of four states:

State	Color	Description
IDLE	●	Device is not performing any action with respect to this property.
OK	●	Last operation performed on this property was successful and active.
BUSY	●	The property is performing an action.
ALERT	●	The property is in critical condition and needs immediate attention.

Table 1. INDI Property States

INDI Drivers

INDI driver is the program that communicates directly to the device. It is responsible for controlling the device parameters and for defining them to clients. Drivers send a list of supported *device properties* to clients where they are parsed and presented to the end users. Furthermore, drives have complete authority on the device operation and parameters, with privileges to accept, ignore, or reject requests from clients. Since INDI is based on a distributed architecture model, drivers should be capable to accommodate any number of clients transparently.

INDI Clients

Clients are the software frontends that communicate with the hardware drivers. They usually communicate with INDI hardware drivers via INDI server, though they can communicate with the drivers directly.

There are many types of clients, most notably:

- Generic GUI clients like KStars or Xephem. Such clients generate a dynamic GUI to offer users a *control panel* to control the device.

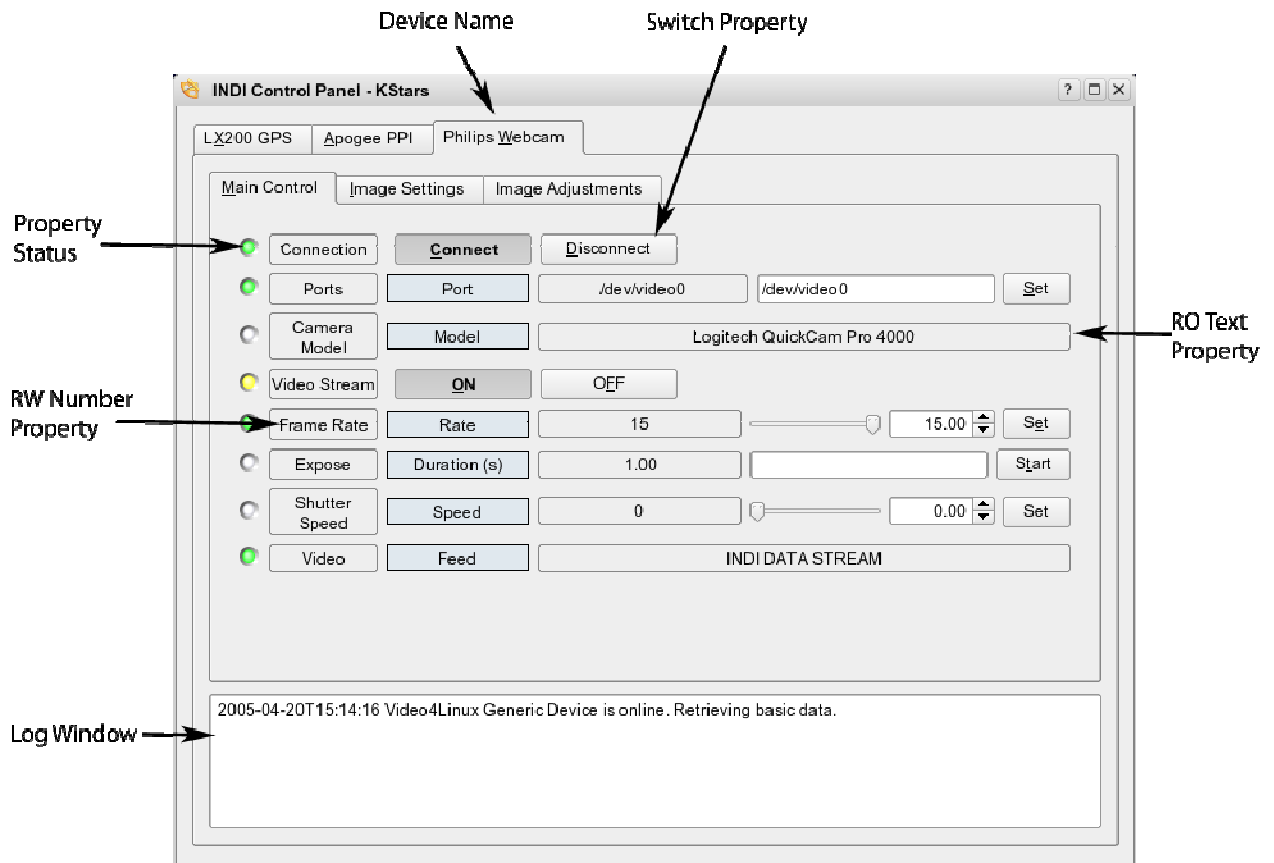


Figure 2. INDI Control Panel in an INDI Compliant Client

- Logger clients to record messages, alarms, and data exchanged between devices and clients.
- Watch dog clients to insure safe and proper operation of devices.
- Automated scripts to carry on complex and coordinates operations on devices.

INDI Server

INDI server is the *hub* that sits between device drivers and respective clients. It reroutes traffic for control and data across distributed networks. Each device or client in the network is a node and may communicate with other nodes whenever desired. The INDI server supports broadcasting, chaining, and marshalling of data.

Typically, each INDI server is responsible for routing traffic for its local devices. In multisite situations, several INDI servers can be *chained* to link local and remote devices. This process is completely transparent to clients and drivers alike.

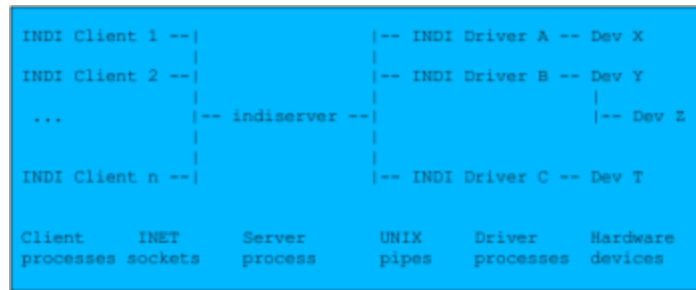


Figure 3. INDI Server Network Topology

Inter-device communication

In today's observatories, automation paves the groundwork for continuous and unattended acquisition, processing, and analysis of scientific data. Observation jobs are often defined as a workflow schema constrained by local weather circumstances, condition and status of equipments, and a multitude of user-defined parameters inherit to the local observatory environment.

Many tasks require the coordination of several devices in the observatory. Let's discuss a classical example in an observatory with the following three devices:

- **Telescope:** Controllable GOTO telescope.
- **Dome:** Controllable dome structure.
- **Rain Collector:** A device to raise an alert when detecting rain.

In the above scenario, the rotation of the dome structure depends on the orientation of the telescope coordinates (RA/DEC) such that the dome remains in sync with telescope's field of view. Furthermore, the dome slit is dependent on the rain collector status. If the rain collector raises a rain alert, the dome driver should be notified in order to initiate a procedure to shutdown the dome slit.

INDI accomplishes inter-device communication by providing a publisher-subscriber mechanism where drivers may subscribe to the properties of other drivers. Depending on the desired configuration, whenever a property is updated in the *publisher* driver, the *subscriber* driver gets notified, even if the two drivers operate at different locations.

By default, all INDI drivers connected to an INDI server become *publishers*. Consequently, drivers only need to subscribe (or unsubscribe) to any property in any device within the INDI server framework. However, the *subscriber* driver should never assume the continued availability of the *publisher* driver all the time, and must plan a procedure in case the *publisher* driver terminates prematurely. In our example, the dome driver might sound an alarm if it loses connection with the rain collector after a specific timeout value, and if no operator is available to mend the problem, then the dome driver automatically shuts down the dome slit as a precaution.

It should be emphasized that drivers should remain as independent as possible from other drivers to keep the overall system modular; utilization of inter-device communication facility in INDI should be limited to crucial cases only.

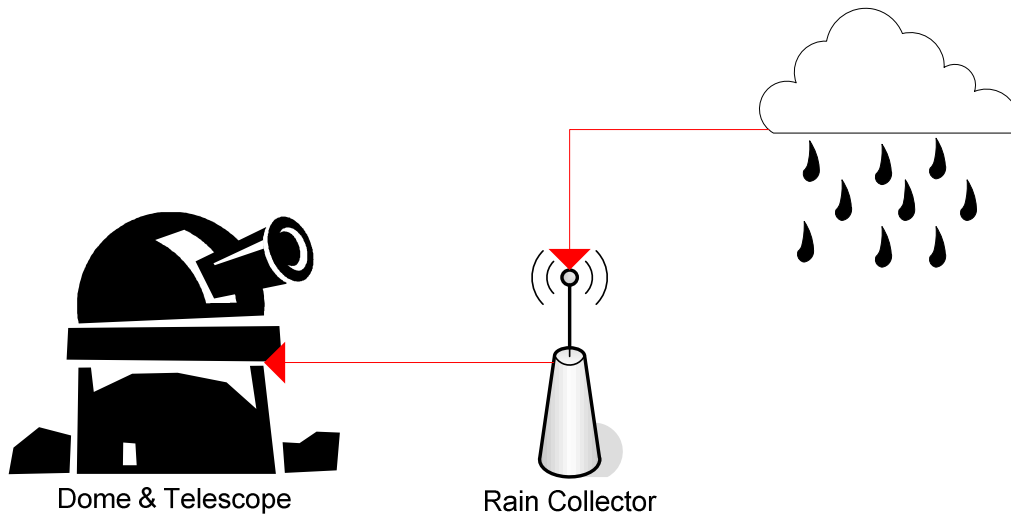


Figure 4. Inter-device communication example

Scheduling & Automation

A key demand in robotic observatories is to maximize utilization efficiency of the observatory's scientific assets to meet the scientific objectives of the project. Scheduling is critical to gain more science with less management & operation overhead. But unless schedulers follow a clearly defined and comprehensive policy, they can introduce more problems than solutions. It should be noted that a common terminology problem is to mix scheduler *policies* with scheduler *tools*. The former defines the philosophy, paradigm, and architecture of a scheduler system, while the latter are the *implementation* of the *policies*, not vice versa. Desired capabilities of a scheduler system is adaptability, interactivity, and connectivity (Johnston 1995)

A typical observation request consists of several distinct stages summarized below:

1. **Proposal:** Define the scientific objective, justification, anticipated outcome, and benefit of the observation request.
2. **Planning:**
 - A. **Asset:** Which devices are involved?
 - B. **Constraints:** What conditions must be met?
 - C. **Workflow:**
 - a. Define unique simple tasks with optional generic IO interfaces.
 - b. Define escalation procedures.

- c. Establish relations among tasks, parallel and sequential execution paths, queues, and dependencies.

3. **Scheduling:**

- A. **Time:** When to perform the observation request?
- B. **Priority:** What is the priority of this request? The system automatically calculates the priority of a scheduler job, and adds the priority specified by the user into account.
- C. **Cycle:** Does the observation job need to run at specific intervals?

- 4. **Execution:** The scheduler prioritizes observation jobs and run the best candidate given the local circumstances.

For scheduling purposes, INDI defines two new elements, *at* and *by*:

Element	Description	Example
by	by time <i>t</i> from the beginning of the observation, all enclosed actions must achieve an OK status.	<pre><by t="20"> <new SwitchVector device="Camera" Name="Shutter_CTL"> <oneSwitch name="Shutter">On</oneSwitch> </newSwitchVector> </by></pre>
at	at time <i>t</i> , perform the enclosed actions.	<pre><at t="60"> <newNumberVector device="Telescope" Name="COORDS"> <oneNumber name="ALT">20</oneNumber> <oneNumber name="AZ">50</oneNumber> </newNumberVector> </at></pre>

INDI Library does not provide a scheduling daemon as of the writing of this paper, but due to the flexibility of the INDI protocol, its DCS functionality can be nested in any scheduler framework that supports XML such as RTML.

Conclusion

INDI is an open-source Distributed Control System (DCS) based on XML technology and distributed architecture. It features a truly instrument-neutral behavior between frontend clients and backend drivers that permits parallel independent developments of its high level components without affecting their operation or performance. Furthermore, remote control of devices is seamless with INDI's server/client architecture. Distributed devices can be controlled from one centralized environment.

Finally, INDI drivers are scriptable using INDI scripting tools. These command line tools can be used to orchestrate fairly complicated observation jobs. Additionally, developers can exploit the tools to provide scheduling and automation frameworks for their devices.

References

Downey, Elwood. 2003, "Instrument Neutral Distributed Interface"
<http://www.clearskyinstitute.com/INDI/INDI.pdf>

Mutlaq, J. and Downey, E. 2005 "INDI Developers Manual" <http://indi.sf.net/manual>

Johnston, M. 1995 "Scheduling Tools for Astronomical Observations" in New observing modes for the next century. Astronomical Society of the Pacific Conference Series, Volume 87, Proceedings of a workshop held in Hilo, Hawaii, 6-8 July 1995