# Introduction & FAQ


## First of all… why another standard?

Yes, there are other initiatives for astronomy software development. Probably the most important is ASCOM. Well established, accepted by the industry, nevertheless Windows only, locked to .Net technologies and without any clear strategy toward IoT and distributed computing.

Another one is INDI with opposite approach. Easy to port to Unix based systems, fully distributed, but not enough standardised in many aspects, quite inefficient for large amount of data and published under license avoiding real commercial use in many situations and thus wider industry acceptance.

That's why we begun development of INDIGO framework with the goal to take the best from the both worlds – community driven, open to everybody, portable, distributed, efficient, easy to use for developers and invisible for end-users.

Why version 2.0 at the very beginning? It can use legacy INDI wire protocol version 1.7 as a fall down option and thus INDIGO protocol version started on 2.0. And unlike in INDI, INDIGO wire protocol version is coupled with the rest of framework.

Oh you asked just why the name INDIGO? It is colour #4B0082. Any similarity to other framework, living or dead, is purely coincidental ☺

## What are design requirements and limitations?

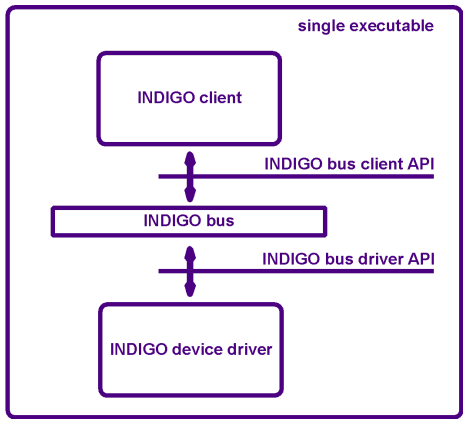
This is current list of requirements taken into consideration:

1. No GPL/LGPL code used to allow commercial use under application stores licenses.
2. ANSI C for portability and to allow simple wrapping into .Net, Java, GO, Objective-C or Swift.
3. Layered architecture to allow both direct linking of the drivers into applications or distributed use.
4. Scalability from mobile devices and embedded computers to desktops.
5. Atomic approach to device drivers. E.g. if camera has imaging and guiding chip, driver should expose two independent simple devices instead of one complex. It is much easier and transparent for client.
6. Hot-plug support for USB devices. If device is connected/disconnected while driver is running, its properties should appear/disappear on the bus.

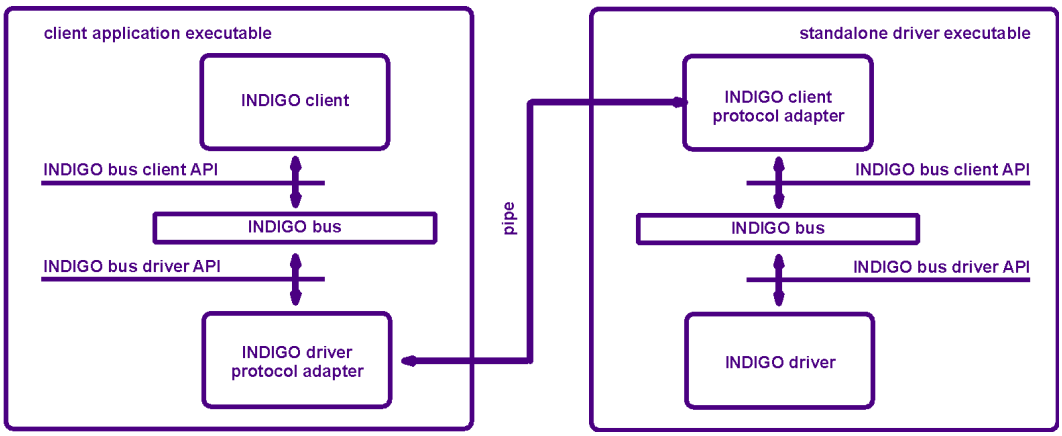
## What is the high-level architecture of INDIGO?

INDIGO is actually a kind of software bus. There are processes in “device” role connected from “device” side and processes in “client” role connected from “client” side. Processes exchange messages modelled according INDI standard (not just because of backward compatibility, but also because it is proven good practise) over the bus and process them.

In the simplest case, on “device” side of INDIGO bus is a single device driver and on “client” side an application using this driver, all linked into single executable and operating on raw speed (only pointer to raw data is transmitted over INDIGO bus instead of base64 encoding/decoding and wire protocol transfer in INDI or native data marshalling in ASCOM).



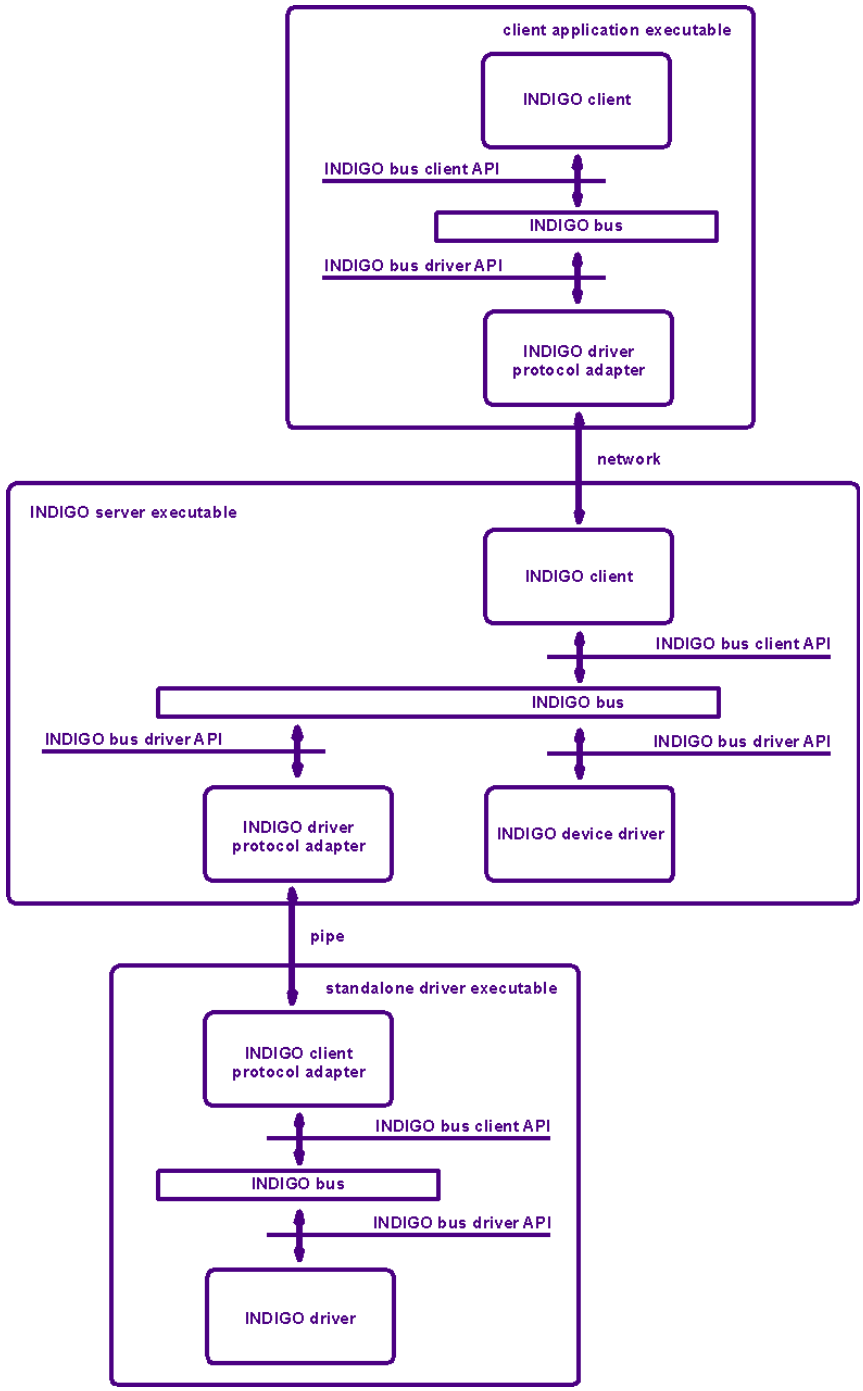

In the more complicated case, application can be divided into separate client and driver executables.




In the client executable, INDIGO wire protocol adapter is in the role of “device” and on “device” side and actual application code again on “client” side of INDIGO bus. Client executable can spawn the driver executable and communicate with it over pipes. In this case driver can be selected on run-time. Also vendor provided driver executable or legacy INDI driver can be used.

In the driver executable, INDIGO wire protocol adapter is in the role of “client” on “client” side and the device driver is again on “device” side of INDIGO bus.

In the most generic case one or more drivers, linked or standalone executables and INDIGO or INDI, can be used by specialised application called INDIGO server (similar to INDI server with the exception of much more efficient linked driver option).




Client application will discover one or more INDIGO servers on the network, connect over wire protocol adapters on client and server side and use remote devices the same way as if they are linked.

There are infinite possibilities how to combine real or virtual device drivers, client or server side protocol adapters and real or virtual clients sharing common design principles and messaging API.

## Can INDIGO client or application use INDI driver?

Yes, INDIGO wire protocol adapter has the ability to fall down to legacy INDI wire protocol version 1.7 to maintain backward compatibility with INDI drivers. In this mode also property and item names are mapped to INDI standard corresponding to version 1.2.

If you want use linked driver option or to use different wire protocol, you have to use INDIGO driver.

## Can INDI client or application use INDIGO driver?

Yes, if client or application will initiate communication over INDI wire protocol, INDIGO driver will simulate behaviour of INDI driver with common base set of properties.

## Can INDIGO client or application use ASCOM driver?

Partially, INDIGO client or application can use INDI Server for Windows as ASCOM-to-INDI bridge out-of-the-box, but probably much more efficient ASCOM driver wrapper for direct linking will come soon.

## Can ASCOM client or application use INDIGO driver?

Not yet, but it should be quite easy to write ASCOM driver wrapping linked INDIGO driver or generic ASCOM driver speaking INDIGO wire protocol.