

Serial Relay Service

This chapter contains the following sections:

- Serial Relay Service Overview, on page 1
- Data Paths, on page 1
- Configuration Commands, on page 3

Serial Relay Service Overview

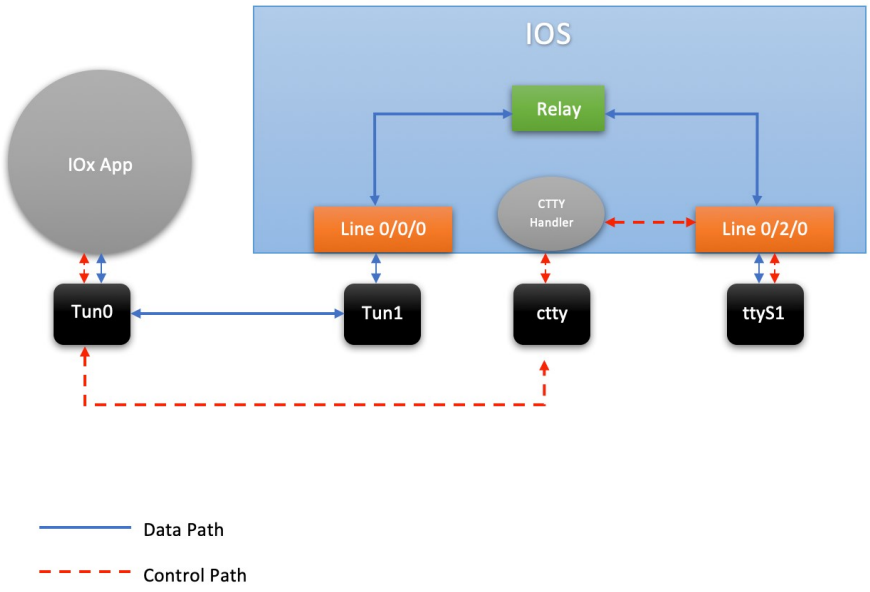
Serial Relay service on the IR1101 enables IOx apps to communicate with the Async Serial port (/dev/ttyS1 under IOS-XE). The configuration of Serial Relay service is similar to that of the IR800.

Data Paths

On the IR1101, IOS-XE has complete control over the data path and control path of the Async Serial port. This aspect is essential to other encapsulations supported on the Aysnc port such as PPP, raw-socket, SCADA, etc. The IOx app is never allowed to exercise full control over the device. All data and configurations are passed through IOS-XE before going to the device. Instead of exposing the actual Serial port to IOx apps, the Serial relay service creates a software emulated serial tty device enumerated as /dev/ttyTun0 (shown below).

The pair of devices /dev/ttyTun0 and /dev/ttyTun1 represent a data tunnel whose primary function is to act as a pass-through gateway during any data transfer. /dev/ttyTun1 is open by IOS-XE and all the ingress/egress data from IOS to the app uses this device during data transfer. Line 0/0/0 is used to communicated with /dev/ttyTun1. Serial relay service should be configured beforehand to allow the connection between two lines.

Figure 1: Data Paths



Data Path:

1. When the IOx app sends a character to /dev/ttyTun0, the tunnel driver automatically pushes the data to /dev/ttyTun1.
2. IOS reads the data which it then passes to the Serial relay service.
3. The Serial relay service retrieves information about the other end of the relay service (Line 0/2/0 in this case) and forwards the data to the Line's buffer.
4. The line driver actively pushes the data into the actual serial device (/dev/ttyS1) based on buffer availability.
5. The reverse path functions the same with the roles of /dev/ttyS1 and /dev/tun0 reversed.

Control Path:

1. When the IOx app performs TCGETS ioctl call on /dev/ttyTun0, the tunnel driver uses /dev/cttyTun to send request to the CTTY handler service running in IOS.
2. CTTY handler service and the kernel driver use a client-server architecture to communicate configuration objects.
3. Upon receiving the request about TCGETS from /dev/cttyTun, the CTTY handler examines the request and requests Line driver to populate the required data into control data structures.
4. Upon receiving the control data structures, CTTY handler sends out a response to /dev/cttyTun which eventually goes back to /dev/ttyTun0.
5. /dev/ttyTun0 passes the control data to IOx app as requested.

6. Similar path can be extrapolated for TCSETS where the CTTY handler requests the Line driver to update the settings of the underneath /dev/ttyS1 driver.
7. Line driver of Line 0/2/0 and driver config on /dev/ttyTun0 are always in sync with each other. Any configuration changes such as baud rate modification is transparently propagated to the Line driver without any additional configuration overhead. This emulates the propagation feature of Serial relay on the IR800 series where the virtual serial port can configure the parameters of the real serial port.

Configuration Commands

```
IR1101#configure terminal
IR1101(config)#interface async 0/2/0
IR1101(config-if)#encapsulation relay-line
IR1101(config-if)#exit
IR1101(config)#relay line 0/2/0 0/0/0
IR1101(config)#exit
IR1101#
```