# Serial TTY overview

*Stable: 15.10.2019 - 10:44 / Revision: 15.10.2019 - 10:44*

**SUMMARY**
This article gives information about the Linux® TTY framework. It explains how to activate the **UART** interface, and how to access it from user and kernel spaces.

**Contents**



## 1 Framework Purpose

The **TTY** subsytem controls the communication between UART devices and the programs using these devices.
The TTY subsystem is responsible for:

- controlling the physical flow of data on asynchronous lines (including the transmission speed, character size, and line availability)
- interpreting the data by recognizing special characters and adapting to national languages
- controlling jobs and terminal access by using the concept of controlling terminal

**The synchronous mode of STM32 USART peripheral is not supported by the TTY subsystem**.
A controlling terminal manages the input and output operations of a group of processes. The TTY special file (ttyX filesystem entry) supports the controlling terminal interface.
To perform its tasks, the TTY subsystem is composed of modules, also called disciplines. A module is a set of processing rules that govern the interface for communication between the computer and an asynchronous device. Modules can be added and removed dynamically for each TTY.

The TTY subsystem supports three main types of modules:

- TTY drivers: TTY drivers, or hardware disciplines, directly control the hardware (TTY devices) or pseudo-hardware (PTY devices). They perform the actual input and output to the adapter by providing services to the modules above it. The services are flow control and special semantics when a port is being opened.
- Line disciplines: Line disciplines provide editing, job control, and special character interpretation. They perform all transformations that occur on the inbound and outbound data streams. Line disciplines also perform most of the error handling and status monitoring for the TTY drivers.
- Converter modules: Converter modules, or mapping disciplines, translate, or map, input and output characters.

Since kernel 4.12 version, the serial device bus (also called Serdev) has been introduced in TTY framework to improve the interface offered to devices attached to a serial port (ex: Bluetooth, NFC, FM Radio and GPS devices), as the line disciplines "drivers" have some known limitations:

- devices are encoded in user space rather than in firmware (Device Tree of ACPI)
- "driver" are not kernel drivers but user space daemons
- Associated ressources (GPIOs and interrupts, regulators, clocks, audio interface) are not described in kernel space, wich impacts power management
- "drivers" are registered when a port is opened

Serdev allows to attach a device on UART without known line disciplines limitations:

- New bus type: serial
- Serdev controllers
- Serdev devices (clients or slaves)
- Serdev TTY-port controller
  - Only in-kernel controller implementation
  - Registered by TTY driver when client is defined
  - clients are described by firmware (Device Tree or ACPI)

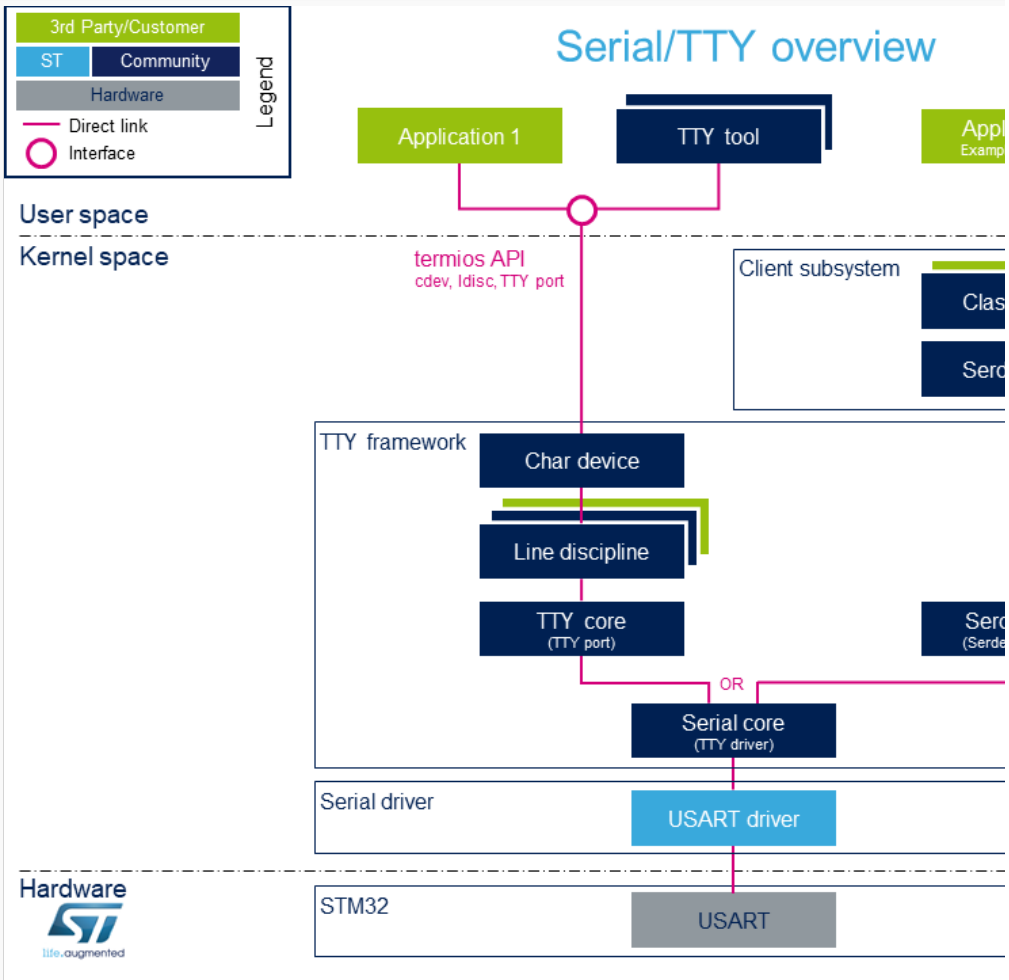
The USART low-level driver provided by STMicroelectronics, (**drivers/tty/serial/stm32-usart.c**) supports RS-232 standard (for serial communication transmission of data), and RS-485 standard (for modbus protocol applications as example).

The Synchronous mode of USART is not supported by Linux® low-level driver .

TTY framework is used to access the serial device in following use cases:

- **tty virtual console** during Linux boot sequence
- **pts pseudo-terminal** to access over a terminal
- **user space application**

## 2 System overview




Serial TTY architecture overview

Template:WarningImageMapOverlay

Note: during boot, while a serial device is probed, serial framework instantiates an associated tty terminal as a virtual device. Then the system sees this tty virtual device as a child of the associated serial device.

## 2.1 Components description

*From client application to hardware*

- Application: customer application to read/write data from the peripheral connected on the serial port.
- TTY tools: tools provided by Linux community, such as **stty**, **ldattach**, **inputattach**, **tty**, **ttys**, **agetty**, **mingetty**, **kermit** and **minicom**.
- Termios: API which offers an interface to develop an application using serial drivers.
- Client subsystem: kernel subsystem client of **serdev** core (Example: Bluetooth_overview )
- TTY framework: high-level TTY structures management, including **tty character device driver** , **TTY core functions** , **line discipline** and **Serdev core functions** management.
- Serial framework: low-level serial driver management, including the **serial core** functions .
- USART driver: stm32-usart **low-level serial driver** for all stm32 family devices.
- STM32 USART: **STM32 frontend IP** connected to the external devices through a serial port

## 2.2 APIs description

The TTY provides only **character device interface** (named /dev/ttyX) to user space. The main API for user space TTY client applications is provided by the portable POSIX terminal interface termios, which relies on /dev/ttyX interface for TTY link configuration.

The **termios API** [1] is a user land API, and its functions describe a general terminal interface that is provided to control asynchronous communications ports.

The POSIX termios API abstracts the low-level details of the hardware, and provides a simple yet complete programming interface that can be used for advanced projects. It is a wrapper on **character device API** [2] ioctl operations.

Note: If a serial interface is needed at kernel level (to control an external device through U(S)ART by a kernel driver for example), customer can use a **line discipline** or a **Serdev** client.

- The **line discipline** will be responsible for:
  - creating this new kernel API
  - routing data flow between serial core and new kernel API
- The **Serdev** provides an interface to kernel drivers.
  - This interface resembles line-discipline operations: open and close, terminal settings, write, modem control, read (callback), and write wakeup (callback)

# 3 Configuration

This section describes how to configure a device on a serial port.

## 3.1 Kernel Configuration

Serial driver, serial framework, and TTY framework are activated by default in ST deliveries. Nevertheless, if a specific configuration is needed, this section indicates how IIO can be activated/deactivated in the kernel.

Activate the device TTY in kernel configuration with Linux Menuconfig tool.

For TTY, select:

```
Device Drivers  --->
   Character devices  --->
     [*]   Enable TTY

Allows to remove TTY support which can save space, and blocks features that require TTY fr
TTY is required for any text terminals or serial port communication. Most users should lea
```

For STM32 serial driver, select:

```
Device Drivers  --->
   Character devices  --->
      Serial drivers  --->
     <*> STMicroelectronics STM32 serial port support
     [*]   Support for console on STM32

This driver is for the on-chip serial controller on STMicroelectronics STM32 MCUs.
USART supports Rx & Tx functionality. It supports all industry standard baud rates.
```

## 3.2 Device tree configuration

UART configuration thanks to device tree is described in the dedicated article serial/TTY device tree configuration.

# 4 How to use TTY

This section describes how to use TTY from user land (from a terminal and a an application) and from kernel space, based on the two following use cases:

- How to configure the serial port
- How to send/receive data

Tips to use TTY:

- **How to use TTY from user terminal**: TTY usage from a user terminal is described in a dedicated article, How to use TTY from a user terminal
- **How to use TTY from an application**: TTY usage from an application is described in a dedicated article, How to use TTY from an application
- **TTY line discipline**: TTY line discipline is described in a dedicated article, Serial/TTY line discipline

# 5 How to trace and debug the framework

## 5.1 How to monitor

As Debugfs doesn't propose any information about serial or TTY frameworks, the way to monitor Serial and TTY frameworks is to use the linux kernel log method (based on printk) described in Dmesg_and_Linux_kernel_log article.

## 5.2 How to trace

- The following extract of **kernel boot log** shows a serial driver properly probed:

```
[    0.793340] STM32 USART driver initialized
[    0.798779] 4000f000.serial: ttySTM1 at MMIO 0x4000f000 (irq = 25, base_baud = 4000000)
[    0.808875] stm32-usart 4000f000.serial: interrupt mode used for rx (no dma)
[    0.816106] stm32-usart 4000f000.serial: interrupt mode used for tx (no dma)
[    0.824253] 40010000.serial: ttySTM0 at MMIO 0x40010000 (irq = 27, base_baud = 4000000)
[    0.833796] console [ttySTM0] enabled
[    0.833796] console [ttySTM0] enabled
[    0.840862] bootconsole [earlycon0] disabled
[    0.840862] bootconsole [earlycon0] disabled
[    0.850132] stm32-usart 40010000.serial: interrupt mode used for rx (no dma)
[    0.855755] stm32-usart 40010000.serial: interrupt mode used for tx (no dma)
```

- dmesg output information

The system log shows that the UART devices and associated TTY terminals registered during the probe.

```
Board $> dmesg | grep ttySTM*
[    0.000000] Kernel command line: root=/dev/mmcblk0p5 rootwait rw earlyprintk console=tt
# ttySTM1 terminal is associated with usart3 (4000f000.serial) #
[    0.798779] 4000f000.serial: ttySTM1 at MMIO 0x4000f000 (irq = 25, base_baud = 4000000)
# ttySTM0 terminal is associated with uart4 (40010000.serial) for console#
[    0.824253] 40010000.serial: ttySTM0 at MMIO 0x40010000 (irq = 27, base_baud = 4000000)
# ttySTM0 terminal is activated by default for console #
[    0.833796] console [ttySTM0] enabled
```

## 5.3 How to debug

While a TTY serial port is instantiated, TTY core exports different files through devfs, sysfs and procfs.

### 5.3.1 devfs

- The repository **/dev** contains all the probed TTY serial devices.

```
Board $> ls /dev/ttySTM*
# ttySTM1 and ttySTM0 terminals are probed #
/dev/ttySTM1  /dev/ttySTM0
```

### 5.3.2 sysfs

- **/sys/class/tty/** lists all TTY devices which ttySx correspond to serial port devices.

```
Board $> ls /sys/class/tty/*/device/driver
/sys/class/tty/ttySTM1/device/driver -> ../../../../bus/platform/drivers/stm32-usart
/sys/class/tty/ttySTM0/device/driver -> ../../../../bus/platform/drivers/stm32-usart
```

- **/sys/devices/platform/soc/** lists all the usart devices probed

```
Board $> ls -d /sys/devices/platform/soc/*.serial
# Serial devices 4000f000.serial (usart3) and 40010000.serial (uart4) are probed #
/sys/devices/platform/soc/4000f000.serial  /sys/devices/platform/soc/40010000.serial
```

- **/sys/devices/platform/soc/*device*.serial/tty** lists the TTY terminal associated to a serial device

```
Board $> ls /sys/devices/platform/soc/4000f000.serial/tty/
# ttySTM1 is associated to serial device 4000f000.serial (usart3)  #
ttySTM1
```

### 5.3.3 procfs

- The repository **/proc/device-tree** lists all the usart devices declared in the device-tree, including the disabled ones.

```
Board $> ls -d /proc/device-tree/soc/serial@*
/proc/device-tree/soc/serial@4000e000  /proc/device-tree/soc/serial@40010000  /proc/device
tree/soc/serial@40018000  /proc/device-tree/soc/serial@44003000
/proc/device-tree/soc/serial@4000f000  /proc/device-tree/soc/serial@40011000  /proc/device
tree/soc/serial@40019000  /proc/device-tree/soc/serial@5c000000
```

- Then for each device listed, device-tree properties are available.

```
Board $> ls /proc/device-tree/soc/serial@40010000/
clock-names  compatible       interrupts-extended  name     pinctrl-0  pinctrl-names  reg
clocks       interrupt-names  linux,phandle        phandle  pinctrl-1  power-domains  stat
```

As an example, the status entry provide the status of the device in the device tree node.

```
Board $> cat /proc/device-tree/soc/serial@40010000/status
# status of device serial@40010000 (uart4) is set to "okay" in the device tree #
okay
```

- **/proc/interrupts** lists all active interrups. To be listed, the serial interrupts need to be activated by opening a port on the UART instances.

```
Board $> cat /proc/interrupts | grep serial
26:          0          0     GIC-0  71 Level     4000f000.serial
27:          0          0  stm32-exti-h  28 Edge      4000f000.serial
28:      13509          0     GIC-0  84 Level     40010000.serial
29:          0          0  stm32-exti-h  30 Edge      40010000.serial
```

## 6 How to go further

The Linux community provides many detailed documentation about Linux serial/TTY. Please find below a selection of the most relevant ones:

- Linux Serial-HOWTO [3] describes how to set up serial ports from both hardware and software perspectives.
- Serial Programming Guide for POSIX Compliant Operating Systems [4] , by Michael Sweet.

More information can be found in the following web articles in order to get a good understanding of Linux TTY framework:

- TTY Subsystem [5], by IBM
- The TTY demystified [6], by Linus Akesson
- Serial drivers training [7], by Bootlin
- Linux Serial drivers [8], by Alessandro Rubini
- Serial Device Bus [9], by Johan Hovold

## 7 References


1. ↑ termios API, Linux Programmer's Manual termios API Documentation (user land API with serial devices)
2. ↑ Character device API overview, *Accessing hardware from userspace* training, Bootlin documentation
3. ↑ Linux Serial-HOWTO, tdlp.org training document, describes how to set up serial ports from both hardware and software perspectives
4. ↑ Serial Programming Guide for POSIX Compliant Operating Systems, by Michael Sweet, training document
5. ↑ TTY Subsystem, by IBM
6. ↑ The TTY demystified TTY subsystem presentation article, by Linus Akesson
7. ↑ Linux serial drivers training Linux Serial Drivers training, by Bootlin
8. ↑ Linux Serial Drivers Serial drivers article describing data flows, by Alessandro Rubini
9. ↑ The Serial Device Bus Serdev framework presentation, by Johan Hovold


TeleTYpewriter

Universal Asynchronous Receiver/Transmitter

Universal Synchronous/Asynchronous Receiver/Transmitter

Serial device bus

Application programming interface

Portable Operating System Interface based on uniX (https://en.wikipedia.org/wiki/POSIX_terminal_interface for more details)

terminal input output structure

Android Runtime (see https://source.android.com/devices/tech/dalvik)

Industrial I/O Linux subsystem

Device File System (See https://en.wikipedia.org/wiki/Device_file#DEVFS for more details)

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Process File System (See https://en.wikipedia.org/wiki/Procfs for more details)

Generic Interrupt Controller