Category:Serial TTY

Contents

Stable: 17.06.2020 - 15:27 / Revision: 16.01.2020 - 09:28

Category:Serial TTY

Stable: 17.06.2020 - 10:27 | Revision: 16.01.2020 - 09:23

A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles related to the Linux® **serial/TTY** software framework.

It is recommended to first read the Serial TTY overview article.

TeleTYpewriter

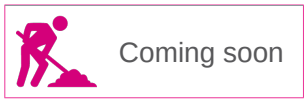
Pages in category "Serial TTY"

The following 6 pages are in this category, out of 6 total.

- Serial TTY overview
- Serial TTY device tree configuration
- How to use TTY with User Terminal
- How to use TTY from an application
- TTY tools
- Serial TTY line discipline

Stable: 04.02.2020 - 08:03 / Revision: 04.02.2020 - 07:55

A quality version of this page, approved on *4 February 2020*, was based off this revision.



Stable: 17.03.2020 - 14:36 / Revision: 25.02.2020 - 15:13

A quality version of this page, approved on *17 March 2020*, was based off this revision.

Contents

1 Purpose

This article describes how to use TTY with a user terminal. The TTY overview is described in Serial TTY overview article.

The use case of the following examples is a data transfer between a STM32 MPU board and PC, over a USB to a RS232 adapter cable.
The setup of this use case is described in details in the How to get Terminal article.

For the following examples:

- uart4 is activated by default (for the Linux console)
- usart3 is enabled by device tree
- The usart3 pins are connected to a RS232 card
- The RS232 card is connected to the PC over the USB to RS232 adapter cable.

Note: Some TTY tools are used in this article. A list of TTY tools is defined a dedicated article [TTY Tools].

2 Print the file name of the terminal connected to standard input (with tty tool)

```
Board $> tty
# The console is connected to uart4 (aka ttySTM0) #
/dev/ttySTM0
```

3 Change serial port configuration (with stty tool)

Many serial port properties can be displayed and changed with the stty tool. The full feature list is available in stty user manual pages[1].

```
Board $> stty --help
```

- Display the current configuration:

  The termios default configuration is specific to each Linux distribution. Before starting a serial communication between two devices, it is recommended to check that termios configurations are compatible on both devices. The termios configurations need to be aligned first.

```
Board $> stty -a -F /dev/ttySTM1
# Display the configuration of uart3 (aka ttySTM1) #
speed 115200 baud; rows 45; columns 169; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;
swtch = <undef>; start = ^Q;
stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
discard = ^O; min = 1; time = 0;
-parenb -parodd -cmspar cs8 hupcl -cstopb cread clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc -
ixany -imaxbel -iutf8
opost -olcuc ocrnl -onlcr -onocr onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig -icanon iexten -echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl
echoke -flusho -extproc
```

- Display only the current baud rate:

```
Board $> stty -F /dev/ttySTM1 speed
# uart3 (aka ttySTM1) baud rate is set to 115200 bps #
115200
```

- Change the baud rate:

  stty -F /dev/ttySTMx *EXPECTED_BAUDRATE*

  Example: change the baud rate to 19200

```
# Change uart3 (aka ttySTM1) baud rate to 19200 bps #
Board $> stty -F /dev/ttySTM1 19200
```

The stty tool proposes many arguments allowing many operations on a tty terminal, such as:

- special settings (various arguments such as speed, line discipline, minimum number of characters for a completed read, size, timeout, etc...)
- control settings
- input settings
- output settings
- local settings

- combination settings

Note: If you want to go further, an interesting tutorial describes termios and stty [2].

4 Send / Receive data (with stty, minicom, echo and cat tools)

Serial counters can be very useful to debug the following use cases.

4.1 Default configuration (8 data bits frame, no parity errors detection, no framing errors detection)

Canonical mode, input modes and output modes termios settings have a major influence on data processing. The following settings can be deactivated for testing.
In case of unexpected behavior, all canonical mode, input modes and output modes settings must be checked. mkssoftware proposes an enriched version of termios manual [3], where the following definitions are provided.

- echo: Enable echo. If ECHO is set input characters are echoed back to the terminal.
- icanon: Canonical input (erase and kill processing). If ICANON is set canonical processing is enabled. In canonical mode input processing is processed in units of lines. A line is delimited by a '\n' character or and end-of-file (EOF) character. A read request does not return until an entire line is read from the port or a signal is received.
- onlcr: Map NL to CR-NL on output. If ONLCR is set the NL character is transmitted as the CR-NL character pair.

Sending data can be simply done by opening the device as a file and writing data to it.

- Configure a port on ttySTM1 (aka usart3). echo, icanon and onlcr properties are deactivated to handle raw data.

```
Board $> stty -F /dev/ttySTM1 115200 -echo -icanon -onlcr
```

- Display the current configuration on ttySTM1 (usart3):

```
# display the configuration of uart3 (aka ttySTM1) #
Board $> stty -a -F /dev/ttySTM1
speed 115200 baud; rows 45; columns 169; line = 0;
```

- Open a port on ttySTM1 (usart3) to receive data

```
Board $> cat /dev/ttySTM1 &
```

- On the remote PC, identify the tty terminal associated to the RS232 card connected on STM32MPU USART3 pins

```
# Command to execute from host terminal #
PC $> ls /dev/ttyUSB*
/dev/ttyUSB0
```

- Open a minicom in a second terminal on the remote device connected on USART3 pins

```
PC $> minicom -D /dev/ttyUSB0
```

- Display the current configuration on ttyUSB0 (remote device):

```
# Display the configuration of host uart (aka ttyUSB0) #
PC $> stty -a -F /dev/ttyUSB0
speed 115200 baud; rows 45; columns 169; line = 0;
```

- Send data from remote PC to STM32MPU over USART3 with default termios configuration (8 frames length, no parity)

```
# Execute this command from host terminal #
PC $> echo "HELLO" > /dev/ttyUSB0
```

- Send data from STM32MPU to remote PC over USART3 with default termios configuration (8 frames length, no parity)

```
# Execute this command from STM32 terminal #
Board $> echo "HELLO" > /dev/ttySTM1
```

4.2 Parity errors detection

Some additional termios functions allow to enable parity errors detection:

- parenb: Parity enable
- parodd: Odd parity else even
- inpck: Enable input parity or framing check
- ignpar: Ignore characters with parity or framing errors

Exemples:

- Configure a port on ttySTM1 (usart3) with even parity enabling

```
# STM32 parity enabling #
Board $> stty -F /dev/ttySTM1 115200 -echo -icanon -onlcr parenb -parodd inpck ignpar
```

- Open a port on ttySTM1 (usart3) to receive data

```
Board $> cat /dev/ttySTM1 &
```

```
Open a minicom in a second terminal on the remote device connected on USART3 pins
PC $> minicom -D /dev/ttyUSB0
```

- Configure a port on ttyUSB0 (remote device) with even parity enabling:

```
# Remote device parity enabling #
PC $> stty -F /dev/ttyUSB0 115200 -echo -icanon -onlcr parenb -parodd inpck ignpar
```

- Send data from remote PC to STM32MPU over USART3

```
# Execute this command from host terminal #
PC $> echo "HELLO" > /dev/ttyUSB0
```

- Send data from STM32MPU to remote PC over USART3

```
#  Execute this command from STM32 terminal #
Board $> echo "HELLO" > /dev/ttySTM1
```

4.3 Framing errors detection

Some additional termios functions allow to enable framing errors detection:

- csize: Number of bits per byte (character size and parity bit configurations)
- inpck: Enable input framing check
- ignpar: Ignore characters with parity or framing errors

Exemples:

- Configure a port on ttySTM1 (usart3) with framing check enabling and 7 data bits length frames

```
# STM32 framing enabling #
Board $> stty -F /dev/ttySTM1 115200 -echo -icanon -onlcr cs7 inpck ignpar
```

- Open a port on ttySTM1 (usart3) to receive data

```
Board $> cat /dev/ttySTM1 &
```

```
Open a minicom in a second terminal on the remote device connected on USART3 pins
PC $> minicom -D /dev/ttyUSB0
```

- Configure a port on ttyUSB0 (remote device) with framing check enabling and 7 data bits length frames

```
# Remote device parity enabling #
PC $> stty -a -F /dev/ttyUSB0 115200 -echo -icanon -onlcr cs7 inpck ignpar
speed 115200 baud; rows 45; columns 169; line = 0;
```

- Send data from remote PC to STM32MPU over USART3

```
# Execute this command from host terminal #
PC $> echo "HELLO" > /dev/ttyUSB0
```

- Send data from STM32MPU to remote PC over USART3

```
# Execute this command from STM32 terminal #
Board $> echo "HELLO" > /dev/ttySTM1
```

5 Identify processes using a tty serial device (with fuser tool)

```
Board $> fuser /dev/ttySTM0
# The process numbered 395, 691 and 3872 are using a tty serial device #
395 691 3872
```

6 Link a tty serial device with a line discipline (with ldattach tool)

Attach ttySTM1 with line discipline number *n* :

```
Board $> ldattach n /dev/ttySTM1
```

7 File transfer over serial console

Please see the dedicated article How to transfer a file over serial console.

8 References

- stty manual page
- A Brief Introduction to termios: termios(3) and stty stty tutorial
- struct_termios man page

TeleTYpewriter

Microprocessor Unit

Linux® is a registered trademark of Linus Torvalds.

also known as

terminal input output structure

Stable: 02.11.2020 - 15:54 / Revision: 03.09.2020 - 13:39

A quality version of this page, approved on *2 November 2020*, was based off this revision.

Contents

1 Article Purpose

This article explains how to configure the USART when it is assigned to the Linux®OS. In that case, it is controlled by the Serial and TTY frameworks.

The configuration is performed using the device tree mechanism that provides a hardware description of the USART peripheral, used by the stm32-usart Linux driver.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The USART is a multifunction device.

Each function is represented by a separate binding document:

- Generic UART bindings[1] used by UART framework.
- STM32 USART driver bindings[2] used by stm32-usart driver. This bindings documentation explains how to write device tree files for STM32 USARTs.

3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the Device tree for an explanation of the device tree file split.

STM32CubeMX can be used to generate the board device tree. Refer to How to configure the DT using STM32CubeMX for more details.

3.1 DT configuration (STM32 level)

All STM32 USART nodes (excepted USART1, secure instance under ETZPC control) are described in microprocessor device tree (ex: stm32mp151.dtsi [3]) with default parameters and disabled status.
The required and optional properties are fully described in the bindings files.

Warning

This device tree configuration related to the STM32 should be kept as is, without being modified by the customer.

3.2 DT configuration (Board level)

Part of the device tree is used to describe the USART hardware used on a given board:

- Which USART instances are enabled (by setting status to "okay")
- Which features are used (such as DMA transfer or direct transfer, transfer speed or parity)
- Which pins are configured via pinctrl.
- Which serial aliases are linked to UART instances. Please check the alias already used in other device tree files to avoid alias conflicts. The alias defines the index of the ttySTMx instance linked the UART.

Note:

- As the pin configuration can be different for each board, several DT configurations can be defined for each UART instance.
- The pin configuration is described in board datasheet. Each new pin configuration described in boards datasheet needs to be defined in device tree.

Three device tree configurations can be defined for each pin muxing configuration:

- Default: for standard usage (mandatory)
- "sleep": for Sleep mode, when the UART instance is not a wake up source (mandatory)
- "idle": for Sleep mode, when the UART instance is a wake up source (optional)

3.3 DT configuration examples

3.3.1 Activation of a USART or UART instance

Information

Some UART pins are available on GPIO expansion and Arduino connectors (depending on the connectors available on the board).

- STM32MP157x-EV1 Evaluation board GPIO expansion connector
- STM32MP157x-DKx Discovery kit GPIO expansion connector

To communicate with a UART instance, an RS232 card must be plugged on the UART pins.

The example below shows how to configure and enable a UART instance at board level, based on STM32MP157C-EV1 board USART3 example.
Note: For STM32 boards, the configuration is already defined in the device tree. Only the device activation is needed.

To activate a UART instance, please follow steps below:

- Define the instance pin configuration (ex: stm32mp15-pinctrl.dtsi [4]).

```
usart3_pins_a: usart3-0 {
        pins1 {
                /* USART3 TX and RTS pins activation for default mode */
                pinmux = <STM32_PINMUX('B', 10, AF7)>, /* USART3_TX */
                         <STM32_PINMUX('G', 8, AF8)>; /* USART3_RTS */
                bias-disable;
                drive-push-pull;
                slew-rate = <0>;
        };
        pins2 {
                /* USART3 RX and CTS_NSS pins activation for default mode */
                pinmux = <STM32_PINMUX('B', 12, AF8)>, /* USART3_RX */
                         <STM32_PINMUX('I', 10, AF8)>; /* USART3_CTS_NSS */
                bias-disable;
        };
};

usart3_idle_pins_a: usart3-idle-0 {
        pins1 {
                /* USART3 TX, RTS, and CTS_NSS pins deactivation for sleep mode */
                pinmux = <STM32_PINMUX('B', 10, ANALOG)>, /* USART3_TX */
                         <STM32_PINMUX('G', 8, ANALOG)>, /* USART3_RTS */
                         <STM32_PINMUX('I', 10, ANALOG)>; /* USART3_CTS_NSS */
        };
        pins2 {
                /* USART3_RX pin still active for wake up */
                pinmux = <STM32_PINMUX('B', 12, AF8)>; /* USART3_RX */
                bias-disable;
        };
};

usart3_sleep_pins_a: usart3-sleep-0 {
        pins {
                /* USART3_TX, RTS, CTS_NSS, and RX pins deactivation for sleep mode */
                pinmux = <STM32_PINMUX('B', 10, ANALOG)>, /* USART3_TX */
                         <STM32_PINMUX('G', 8, ANALOG)>, /* USART3_RTS */
                         <STM32_PINMUX('I', 10, ANALOG)>, /* USART3_CTS_NSS */
                         <STM32_PINMUX('B', 12, ANALOG)>; /* USART3_RX */
        };
};
```

- Define the serial alias for this instance at board level (ex: stm32mp157c-ev1.dts [5]).

```
aliases {
        /* Serial1 alias (ie ttySTM1) assigned to usart3 */
        serial1 = &usart3;
        ethernet0 = &ethernet0;
};
```

- Configure and activate the instance at board level (ex: stm32mp157c-ev1.dts [5]).

```
&usart3 {
        pinctrl-names = "default", "sleep", "idle";        /* pin configurations
definition */
        pinctrl-0 = <&usart3_pins_a>;                        /* default pin configuration
selection */
        pinctrl-1 = <&usart3_sleep_pins_a>;                /* sleep pin configuration
selection */
        pinctrl-2 = <&usart3_idle_pins_a>;                /* idle pin configuration
selection */
        status = "okay";                                        /* device activation */
};
```

Note: The pin configuration selected has to be aligned with the pin configuration described in the board datasheet.

4 How to configure the DT using STM32CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

- Documentation/devicetree/bindings/serial/serial.txt , UART generic device tree bindings
- Documentation/devicetree/bindings/serial/st,stm32-usart.txt , STM32 USART device tree bindings
- arch/arm/boot/dts/stm32mp151.dtsi , STM32MP151 device tree file
- arch/arm/boot/dts/stm32mp15-pinctrl.dtsi , STM32MP15 pinctrl device tree file
- 5.05.1 arch/arm/boot/dts/stm32mp157c-ev1.dts , STM32MP157c ev1 board device tree file

Linux® is a registered trademark of Linus Torvalds.

Operating System

Universal Synchronous/Asynchronous Receiver/Transmitter

Device Tree

Universal Asynchronous Receiver/Transmitter

Direct Memory Access

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Transmit

Receive

Compatibility Test Suite (Android specific) or Clear to send (in UART context)

Stable: 19.02.2019 - 13:32 / Revision: 29.01.2019 - 11:08

A quality version of this page, approved on *19 February 2019*, was based off this revision.

Template:ArticleMainWriter Template:ArticleApprovedVersion

Stable: 17.03.2020 - 14:44 / Revision: 25.02.2020 - 14:49

A quality version of this page, approved on *17 March 2020*, was based off this revision.

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 the 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: the line disciplines provide editing, job control, and special character interpretation. They perform all the transformations that occur on the inbound and outbound data streams. The line disciplines also perform most of the error handling and status monitoring for the TTY drivers.
- Converter modules: the 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 the 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:

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

The Serdev allows a device to be attached on UART without known the line disciplines limitations:

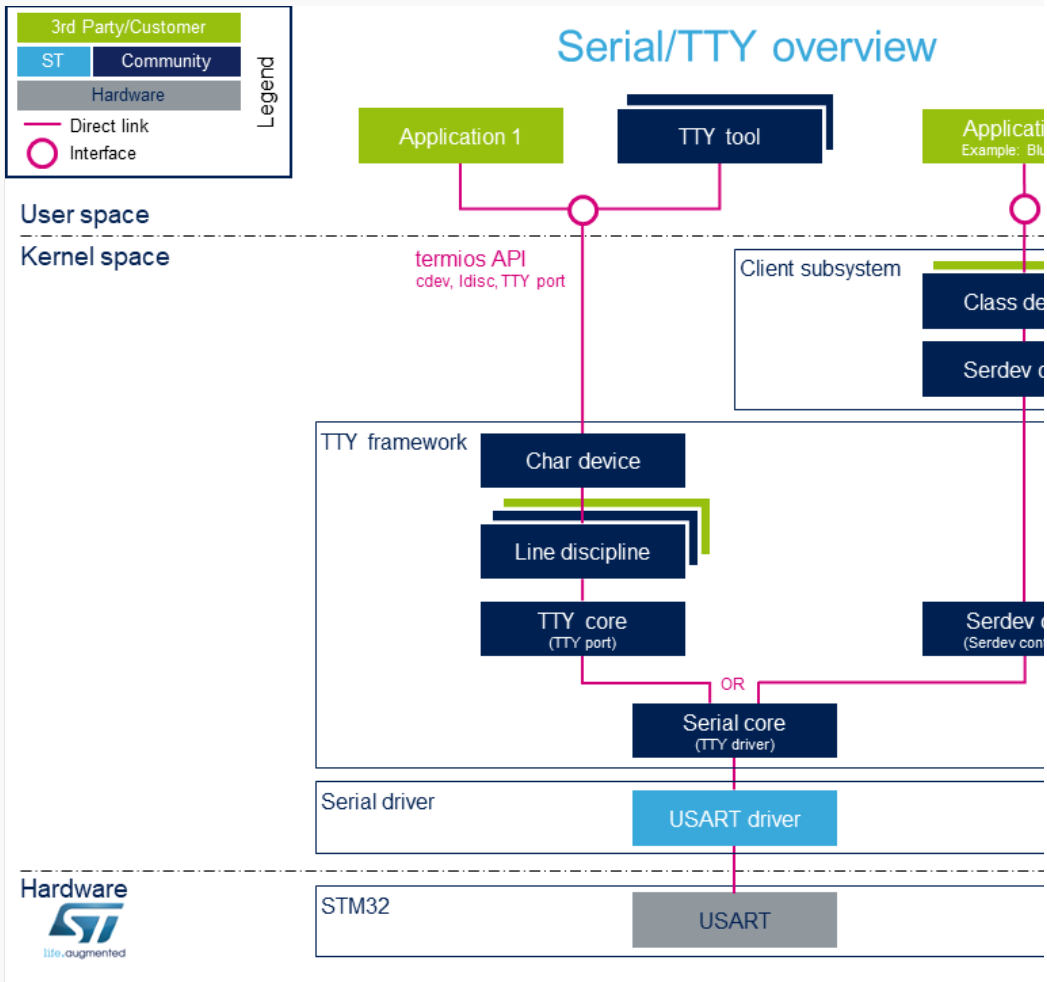
- New bus type: serial
- Serdev controllers
- Serdev devices (clients or slaves)
- Serdev TTY-port controller
  - Only in-kernel controller implementation
  - Registered by TTY driver when client is defined
  - clients are described by firmware (Device Tree or ACPI)

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 .

The TTY framework is used to access the serial device in the 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

Note: during boot, while a serial device is probed, the 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 the 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), the 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 the serial core and the 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

The 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 the TTY support which can save space, and blocks features that require TT
Y from inclusion in the kernel.
The TTY is required for any text terminals or serial port communication. Most users
should leave this enabled.
```

For the 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.
The USART supports Rx and Tx functionality. It supports all industry standard baud rates.
```

3.2 Device tree configuration

The UART configuration thanks to the 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 the user land (from a terminal or an application) and from the kernel space, based on the two following use cases:

- How to configure the serial port by using the termios structure
- How to send/receive data

The termios structure allows to configure communication ports with many settings, such as :

- Baud rate
- Character size mask
- Parity bit enabling
- Parity and framing errors detection settings
- Start/stop input and output control
- RTS/CTS (hardware) flow control
- ...

As the USART internal peripheral supports 7, 8 and 9 word length data, the following termios character size and parity bit configurations are supported:

- CS6 with parity bit
- CS7 with or without parity bit
- CS8 with or without parity bit

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 does not 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

5.2.1 Kernel boot log

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) is a stm32-usart
[    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) is a stm32-usart
[    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)
```

5.2.2 dmesg output information

The system log shows 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=ttySTM1,115200
# ttySTM1 terminal is associated with usart3 (4000f000.serial) #
[    0.798779] 4000f000.serial: ttySTM1 at MMIO 0x4000f000 (irq = 25, base_baud =
4000000) is a stm32-usart
# ttySTM0 terminal is associated with uart4 (40010000.serial) for console#
[    0.824253] 40010000.serial: ttySTM0 at MMIO 0x40010000 (irq = 27, base_baud =
4000000) is a stm32-usart
# ttySTM0 terminal is activated by default for console #
[    0.833796] console [ttySTM0] enabled
```

5.2.3 Dynamic trace

A detailed dynamic trace is available in How to use the kernel dynamic debug

```
 Board $> echo  "file drivers/tty/* +p" > /sys/kernel/debug/dynamic_debug/control
```

This command enables all the traces related to the TTY core and drivers at runtime.
A finer selection can be made by choosing only the files to trace.

Information

Reminder: *loglevel* needs to be increased to 8 by using either boot arguments or the *dmesg -n 8* command through the console

5.3 How to debug

While a TTY serial port is instantiated, the 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     wakeup-source
clocks       interrupt-names  linux,phandle        phandle  pinctrl-1  power-domains
status
```

As an example, the status entry provides 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
```

- The file **/proc/interrupts** lists the interrupts for active serial ports.

```
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
```

- The file **/proc/tty/driver/stm32-usart** lists serial core counters and modem information for each serial instance.

  Driver information:
  - serial driver name
  - serial device start address
  - irq number

  Counters:
  - tx: Number of bytes sent
  - rx: Number of bytes received
  - fe: Number of framing errors received
  - pe: Number of parity errors received
  - brk: Number of break signals received
  - oe: Number of overrun errors received
  - bo: Number of framework buffer overrun errors received

  Modem information:
  - RTS: Request To Send
  - CTS: Clear To Send
  - DTR: Data Terminal Ready
  - DSR: Data Set Ready
  - CD: Carrier Detect
  - RI: Ring Indicator

```
Board $> cat /proc/tty/driver/stm32-usart
serinfo:1.0 driver revision:
0: uart:stm32-usart mmio:0x40010000 irq:29 tx:22722 rx:2276 RTS|CTS|DTR|DSR|CD
1: uart:stm32-usart mmio:0x4000F000 irq:27 tx:0 rx:1149 fe:121 oe:2 pe:296 brk:3 RTS|CTS|D
TR|DSR|CD
3: uart:stm32-usart mmio:0x4000E000 irq:25 tx:0 rx:0 CTS|DSR|CD
```

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 the 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

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

TeleTYpewriter

Universal Asynchronous Receiver/Transmitter

Universal Synchronous/Asynchronous Receiver/Transmitter

Serial device bus

Near Field Communication (is a short-range wireless standard for communication over distances up to around 10cm that will enable enhanced services for users of NFC-enabled smart phones. These could include receiving coupons from retailers upon entering a store, or sharing contacts or photos, in addition to making mobile payments and collecting data from medical monitors, smart meters or other equipment containing ST’s dual-interface EEPROM. We produce a wireless memory that can transmit and receive information from the heart of an application to a smart phone containing NFC technology. NFC is expected to become a widely used system for making payments by smart phone in the United States and it is estimated that by 2015, 30.5% (iSupply) of all handsets shipped will contain NFC technology.)

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

Compatibility Test Suite (Android specific) or Clear to send (in UART context)

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

Stable: 21.02.2020 - 15:07 / Revision: 21.02.2020 - 10:43

A quality version of this page, approved on *21 February 2020*, was based off this revision.

Contents

1 Article Purpose

This article aims at giving some first information useful to start with the Linux®TTY tools. These tools are useful for interacting with TTY terminals.

2 Introduction

These tools use TTY sysfs and character device directly (See TTY user space interface for further details).

3 Tools list

Please find below a list of useful TTY tools provided by Linux® community:

- **fuser** [1] - to identify processes using files or sockets.
- **inputattach** [2] (based on termios API) - to attach a serial line to an input-layer device.

Inputattach attaches a serial line to an input-layer device via a line discipline. Exactly one of the available modes must be specified on the command line.

- **kermit** [3] - transport and platform independent interactive and scriptable communications software.

C-Kermit is a modem program, a Telnet client, an Rlogin client, an FTP client, an HTTP client, and on selected platforms, also an X.25 client. It can make its own secure internet connections using IETF-approved security methods including Kerberos IV, Kerberos V, SSL/TLS, and SRP and it can also make SSH connections through an external SSH client application. It can be the far-end file-transfer or client/server partner of a desktop Kermit client. It can also accept incoming dialed and network connections. It can even be installed as an internet service on its own standard TCP socket, 1649 [RFC2839, RFC2840].

- **ldattach** [4] (based on termios API) - to attach a line discipline to a serial line.

The ldattach daemon opens the specified device file (which should refer to a serial device) and attaches the line discipline ldisc to it for processing of the sent and/or received data. It then goes into the background keeping the device open so that the line discipline stays loaded. The line discipline ldisc may be specified either by name or by number. In order to detach the line discipline, kill the ldattach process. With no arguments, ldattach prints usage information.

- **minicom** [5] - friendly serial communication program.

Minicom is a communication program which somewhat resembles the shareware program TELIX but is free with source code and runs under most Unices. Features include dialing directory with auto-redial, support for UUCP-style lock files on serial devices, a separate script language interpreter, capture to file, multiple users with individual configurations, and more.

- **setserial** [6] - To get/set Linux serial port information.

Setserial is a program designed to set and/or report the configuration information associated with a serial port. This information includes the I/O port and the IRQ used by a particular serial port, and whether or not the break key should be interpreted as the Secure Attention Key, and so on.

- **stty** [7] (based on termios API) - to change and print terminal line settings.
- **tty** [8] - to print the file name of the terminal connected to standard input

Information

The descriptions above are provided by the manual pages of the tools.

4 Getting started

Examples of TTY tools usage are handled in the following articles :

- How to use TTY with User Terminal
- How to transfer a file over serial console

5 Installation on your target

Some of the TTY tools aren't embedded by default in OpenSTLinux distribution. They can be compiled independently and then installed on the target (see Adding Linux user space applications).

6 References

- fuser man page
- inputattach man page
- kermit man page
- ldattach man page
- minicom man page
- setserial man page
- stty man page
- tty man page

Linux® is a registered trademark of Linus Torvalds.

TeleTYpewriter

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

terminal input output structure

Application programming interface