

# How Type-C Port Role Swaps

## UM70105/D

### INTRODUCTION

In Android devices, the ability to dynamically switch between different **power roles** (source or sink) and **data roles** (host or device) is crucial for efficient power management and versatile connectivity. This document explains how users can change these roles through the Android GUI, how the system handles these operations, and the underlying mechanisms involved.

### USER INTERACTION IN ANDROID GUI

#### Accessing USB Settings

1. Users navigate to **Settings** on their Android device.
2. Under **Connected devices** or a similar section, they select **USB preferences** or **USB settings**.

#### Changing Roles

1. Users see options related to USB connections
  - Charging only
  - File transfer (MTP)
  - USB tethering
  - No data transfer
  - Other modes (e.g., MIDI, PTP)
2. By selecting a specific mode, users change the data role (host or device).

#### Power Role

1. The power role (source or sink) is implicitly determined by the connected device:
  - **Source:** The Android device acts as a power source (e.g., when charging another device).
  - **Sink:** The Android device consumes power (e.g., when charging itself).

### INFORMATION EXCHANGE BETWEEN USER SPACE AND KERNEL

#### USB Type-C Connector Class

The `typec` class is meant for describing the USB Type-C ports in a system to the user space in unified fashion. The class is designed to provide nothing else except the user space interface implementation in the hope that it can be utilized on as many platforms as possible.

The platforms are expected to register every USB Type-C port they have with the class. In a normal case the

registration will be done by a USB Type-C or PD PHY driver, but it may be a driver for firmware interface such as UCSI, driver for USB PD controller or even driver for Thunderbolt3 controller. This document considers the component registering the USB Type-C ports with the class as “port driver”.

On top of showing the capabilities, the class also offers user space control over the roles and alternate modes of ports, partners and cable plugs when the port driver is capable of supporting those features.

#### User Space Interface

Every port will be presented as its own device under `/sys/class/typec/`. The first port will be named “port0”, the second “port1” and so on.

When connected, the partner will be presented also as its own device under `/sys/class/typec/`. The parent of the partner device will always be the port it is attached to. The partner attached to port “port0” will be named “port0-partner”. The full path to the device would be `/sys/class/typec/port0/port0-partner/`.

The cable and the two plugs on it may also be optionally presented as their own devices under `/sys/class/typec/`. The cable attached to the port “port0” port will be named “port0-cable” and the plug on the SOP Prime end (see USB Power Delivery Specification ch. 2.4) will be named “port0-plug0” and on the SOP Double Prime end “port0-plug1”. The parent of a cable will always be the port, and the parent of the cable plugs will always be the cable.

In the port, partner, or cable plug supports Alternate Modes, every supported Alternate Mode SVID will have their own device describing them. Note that the Alternate Mode devices will not be attached to the `typec` class. The parent of an alternate mode will be the device that supports it, so for example an alternate mode of port0-partner will be presented under `/sys/class/typec/port0-partner/`. Every mode that is supported will have its own group under the Alternate Mode device named “mode<index>”, for example `/sys/class/typec/port0/<alternate mode>/mode1/`. The requests for entering/exiting a mode can be done with “active” attribute file in that group.

## HOW FUSB302B DRIVER TO IMPLEMENT THE PORT DRIVER INTERFACE

a) Declare Type-C class related staff.

Add Type-C class related staff to FUSB302B driver. The driver exposes information to the system through them and executes requests from the user space.

```
//kernel 5.4
struct typec_capability typec_caps;
struct typec_port *typec_port;
struct typec_partner *partner;
struct typec_partner_desc partner_desc;
struct usb_pd_identity partner_identity;
//
```

These are added to the struct fusb30x\_chip in fusb30x\_global.h. typec\_caps and typec\_port is used to describe local Type-C while partner and partner\_desc are used to describe the attached device. Partner\_identity is used to describe USB power delivery identity data. As described above, the information of the Type-C port and partner is written to sys files.

- `/sys/class/typec/` : The root directory for USB Type-C® connector class.
- `/sys/class/typec/portX/` : Represents a specific USB Type-C® port (e.g., `/sys/class/typec/port0/`).
- `/sys/class/typec/portX/portX-partner/` : Contains information about the connected partner device.
- `/sys/class/typec/portX/portX-cable/` : Provides details about the attached cable.

b) Initialize

In initialization, FUSB302B driver will register the port it represents. The function typec\_register\_port provided by Type C class is used to complete this work.

Here FUSB302B driver fills the port capabilities to type\_caps, including the operations can be done to the port, such as switching the power role and data role of the port with PD, or reconfigure the port type.

```
static struct typec_operations tcpc_ops = {
    .dr_set = usbpd_typec_dr_set,
    .pr_set = usbpd_typec_pr_set,
    .port_type_set = usbpd_typec_port_type_set,
};

static void fusb_register_port(void)
{
    struct fusb30x_chip* chip = fusb30x_GetChip();
    chip->typec_caps.type = TYPEC_PORT_DRP;
    chip->typec_caps.data = TYPEC_PORT_DRD;
    chip->typec_caps.revision = 0x0130;
    chip->typec_caps.pd_revision = 0x0300;
    chip->partner_desc.identity = &chip->partner_identity;
    chip->typec_caps.ops = &tcpc_ops;
    chip->typec_port = typec_register_port(&chip->client->dev, &chip->typec_caps);
}
```



c) when a device attached

When a device is attached, FUSB302B driver will update the port information with a set of functions below:

```

typec_set_data_role
typec_set_pwr_role
typec_set_vconn_role
typec_pwr_opemode
typec_set_orientation

```

and register the attached device as a partner with `typec_register_partner`.

```

void fusb_register_partner(struct fusb30x_chip* chip)
{
    if (!chip->partner) {
        memset(&chip->partner_identity, 0, sizeof(chip->partner_identity));
        chip->partner_desc.usb_pd = false;
        chip->partner = typec_register_partner(chip->typec_port, &chip->partner_desc);
    }
}

if (chip->port.ConnState == AudioAccessory) {
    chip->partner_desc.accessory = TYPEC_ACCESSORY_AUDIO;
} else {
    chip->partner_desc.accessory = TYPEC_ACCESSORY_NONE;
    if (chip->port.sourceOrSink == SINK) {
        cancel_delayed_work_sync(&chip->apsd_recheck_work);
        schedule_delayed_work(&chip->apsd_recheck_work, 200);

        typec_set_data_role(chip->typec_port, TYPEC_DEVICE);
        typec_set_pwr_role(chip->typec_port, TYPEC_SINK);
        typec_set_pwr_opemode(chip->typec_port, chip->port.SinkCurrent - utccDefault);
        typec_set_vconn_role(chip->typec_port, TYPEC_SINK);
        typec_set_orientation(chip->typec_port,
            (chip->port.CCPin==CC1)?TYPEC_ORIENTATION_NORMAL:TYPEC_ORIENTATION_REVERSE);
        fusb_register_partner(chip);
    } else if (chip->port.sourceOrSink == SOURCE) {
        start_usb_host(chip, true);
        typec_set_data_role(chip->typec_port, TYPEC_HOST);
        typec_set_pwr_role(chip->typec_port, TYPEC_SOURCE);
        typec_set_pwr_opemode(chip->typec_port, chip->port.SourceCurrent - utccDefault);
        typec_set_vconn_role(chip->typec_port, TYPEC_SOURCE);
        typec_set_orientation(chip->typec_port,
            (chip->port.CCPin==CC1)?TYPEC_ORIENTATION_NORMAL:TYPEC_ORIENTATION_REVERSE);
        fusb_register_partner(chip);
        chip->usb_state = 2;
        pr_info("FUSB %s start_usb_host\n", __func__);
    }
}

```

d) when the device detached

When the device is detached, FUSB302B driver will unregister it with `type_unregister_partner`. The partner node under port will be removed.

```

if (chip->partner) {
    typec_unregister_partner(chip->partner);
    chip->partner = NULL;
}

```

## e) How to swap the Power and Data Role

When a phone user swaps the roles in the GUI, FUSB302B driver needs to do the underlying work.

The request comes from `/sys/class/typec/portx`, where the user space and kernel exchange information. Registered callback `dr_set` or `pr_set` will be called to do the data role swap or power role swap respectively.

FUSB302B driver has defined the following functions:

```
static int usbpd_typec_dr_set(struct typec_port *port,
                             enum typec_data_role role)

static int usbpd_typec_pr_set(struct typec_port *port,
                             enum typec_role role)
```

Here it leverages the PD commands Data Role Swap and Power Role Swap to negotiate with the partner.

When the negotiation is completed, the FUSB302B driver PD engine will notify DATA\_ROLE and POWER\_ROLE event. The power changes in the procedure of the Power

Role Swap. But the data role is different. It's a simple message exchange, the real work is in the event handle. It needs to request USB driver to switch from Host to Device or vice versa.

FUSB302B driver does this like below.

```
case DATA_ROLE:
    pr_debug("FUSB %s:DATA_ROLE=0x%x\n", __func__, event);

    if (chip->port.PolicyIsDFP == FALSE) {
        typec_set_data_role(chip->typec_port, TYPEC_DEVICE);
        if (chip->usb_state == 2)
            stop_usb_host(chip);
        start_usb_peripheral(chip);
        chip->usb_state = 1;
    } else if (chip->port.PolicyIsDFP == TRUE) {
        typec_set_data_role(chip->typec_port, TYPEC_HOST);
        if (chip->usb_state == 1)
            stop_usb_peripheral(chip);
        start_usb_host(chip, true);
        chip->usb_state = 2;
    }
}
```

All brand names and product names appearing in this document are registered trademarks or trademarks of their respective holders.

**onsemi**, **ONSEMI**, and other names, marks, and brands are registered and/or common law trademarks of Semiconductor Components Industries, LLC dba "onsemi" or its affiliates and/or subsidiaries in the United States and/or other countries. onsemi owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of onsemi's product/patent coverage may be accessed at [www.onsemi.com/site/pdf/Patent-Marking.pdf](http://www.onsemi.com/site/pdf/Patent-Marking.pdf). onsemi reserves the right to make changes at any time to any products or information herein, without notice. The information herein is provided "as-is" and onsemi makes no warranty, representation or guarantee regarding the accuracy of the information, product features, availability, functionality, or suitability of its products for any particular purpose, nor does onsemi assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using onsemi products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by onsemi. "Typical" parameters which may be provided in onsemi data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. onsemi does not convey any license under any of its intellectual property rights nor the rights of others. onsemi products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use onsemi products for any such unintended or unauthorized application, Buyer shall indemnify and hold onsemi and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that onsemi was negligent regarding the design or manufacture of the part. onsemi is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

## ADDITIONAL INFORMATION

### TECHNICAL PUBLICATIONS:

Technical Library: [www.onsemi.com/design/resources/technical-documentation](http://www.onsemi.com/design/resources/technical-documentation)  
onsemi Website: [www.onsemi.com](http://www.onsemi.com)

### ONLINE SUPPORT: [www.onsemi.com/support](http://www.onsemi.com/support)

For additional information, please contact your local Sales Representative at [www.onsemi.com/support/sales](http://www.onsemi.com/support/sales)