Zephyr® Project

Developer Summit

Zephyr® Project
Developer Summit

USB Type-C and You

Turning your product concept into implementation necessities

Sam Hurst, *Google*
Github ID: sambhurst

Diana Zigterman, *Google*
Github ID: DianaZig

Contents

- Port Role Recap
- Specification Requirements
- Creating a New Sink Device

Zephyr® Project
Developer Summit

Port Role Recap

Port Role Recap

Power Role	Data Role	Vconn Role
Source • Supplies power	DFP • Acts as USB Host or Hub Downstream Port • Controls alternate mode entry	Source • Supplies Vconn to the Electronically Marked Cable • Communicates with the Electronically Marked Cable
Sink • Consumes power	UFP • Acts as USB Device	Off • May not communicate with the cable

For more details, see last year's USB-C Power Delivery Overview

Zephyr® Project
Developer Summit

Specification Requirements

Accessing the Specifications

- USB Type-C Specification
  - https://usb.org/document-library/usb-type-cr-cable-and-connector-specification-release-22
  - Goto spec for valid voltage ranges, connect and disconnect timings, TBT alternate mode specifics
- USB Power Delivery Specification
  - https://usb.org/document-library/usb-power-delivery
  - Goto spec for minimum PD requirements and features to explore
- USB Compliance specifications also available on usb.org

Roles: Which of these USB Type-C states apply to me?

- What do you want to connect to?
  - Think through all the things your customer might think they can plug in
- Accessory modes?
  - Debug accessory
    - Generally only useful if your product has a specific debug accessory you use with it
  - Audio accessory
    - Rarely used in industry
- Try states?
  - Primarily useful if you or your partner doesn’t support Power Delivery

Roles: Am I “Dual Role Data”?

- Definite “yes” if your port can function as both a host and device
- Also required if you will be using PD **DR_Swap** messages
  - Because you’d like to use a non-default role combination (sink/DFP)
  - Or because you’re dual-role power with a preferred data role
- Note for DR swaps: the partner may reply with a **Reject**

Roles: Do I have to source Vconn?

- Power source ports can reference the USB Type-C Spec
  - Not required if device sources <= 3A and have no USB 3 support
  - Required Vconn power level ranges from 100 mW to 1.5 W
- Sink ports primarily require Vconn sourcing if they need to probe the cable
- Can I turn Vconn off?
  - Yes, if:
    - Cable e-marker reports it doesn't need Vconn
    - No Ra is detected
    - Cable e-marker doesn't reply to **Discover Identity VDMs**
  - Note: your port is still considered "Vconn source" for purposes of Vconn swaps even if Vconn is turned off

Do I need Power Delivery support?

- If you're content with default USB Type-C roles and have no need for more than 15 W sourcing or sinking, then possibly not
- But you do want PD if you need:
  - More power
  - Non-default role combinations
  - Alternate modes
  - Power button transmission

Which PD messages do I have to support?

- Reference the “Message Applicability” section of the PD Specification

Message Type	Source	Sink	Dual-Role Power	Cable Plug SOP’	Cable Plug SOP’’	VPD [6]
Get_Country_Info	CN[5]/O	CN[5]/O		NA	NA	NA
BIST	N[1]	N[1]		NA	NA	NA
Sink_Capabilities	NA	N	N	NA	NA	NA
Battery_Status	CN[2]	CN[2]		NA	NA	NA

Note 1: For details of which BIST Modes and Messages ***Shall*** be supported see Section 5.9 and Section 6.4.3.
Note 2: ***Shall*** be supported by products that contain batteries.
Note 3: ***Shall*** be supported by products that support the *Get_Battery_Status* Message.
Note 4: ***Shall*** be supported by products that support the *Get_Sink_Cap* Message.
Note 5: ***Shall*** be supported when required by a country authority.

Zephyr® Project
Developer Summit

USB Type-C Sink Device Application

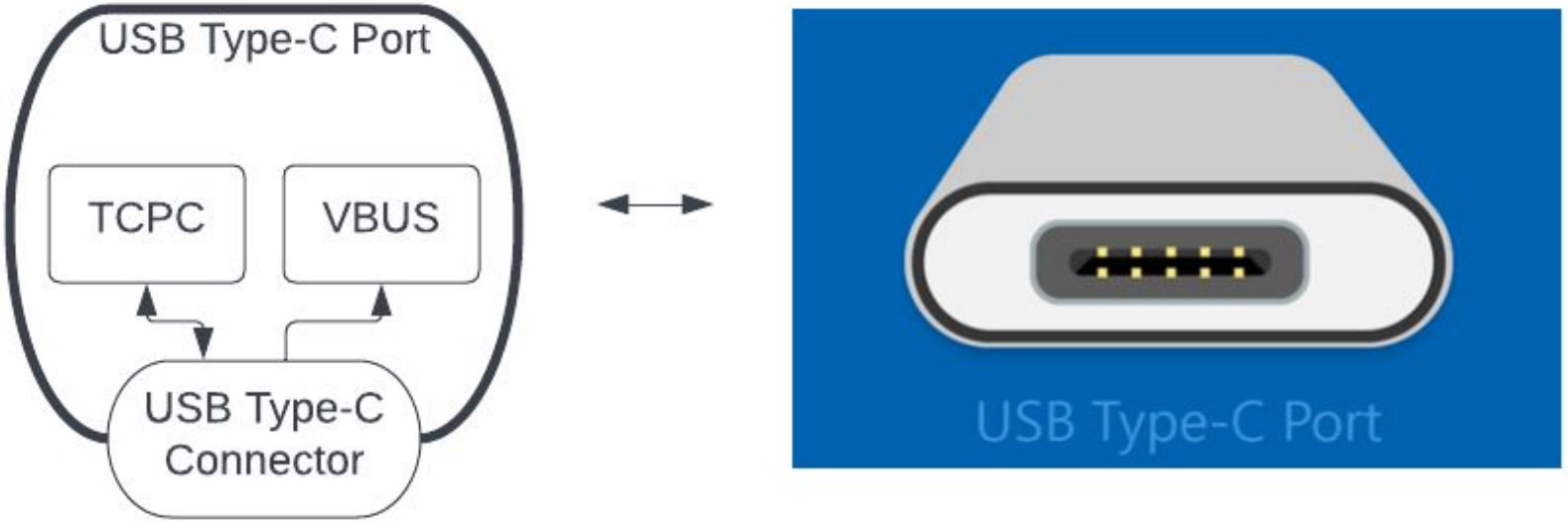
Three Steps to create a USB Type-C Device

1. Create a Devicetree usb-c-connector node
2. Create an application specific data structure
3. Define a few policy callbacks

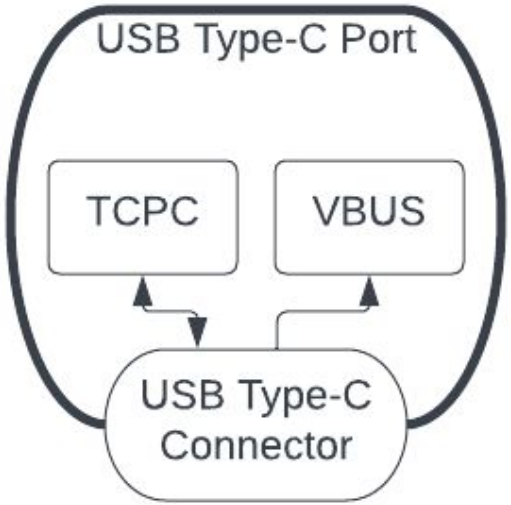
Zephyr® Project
Developer Summit

USB Type-C Devicetree node

Zephyr's USB Type-C Port perspective

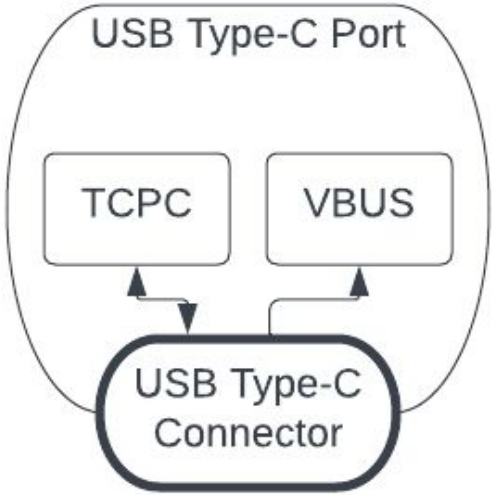


Devicetree USB Type-C Sink Port description



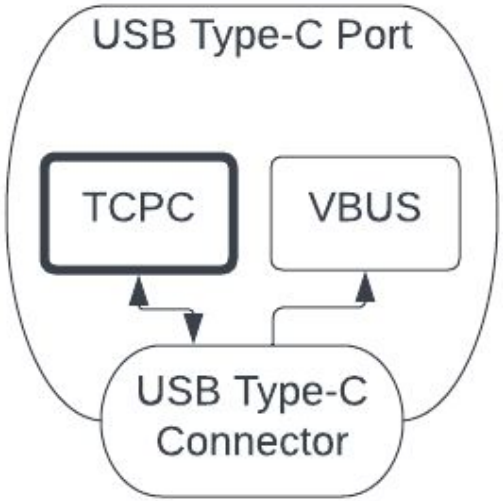

```
port1: usbc-port@1 {
        compatible = "usb-c-connector";
        tcpc = <&ucpd1>;
        vbus = <&vbus1>;
        power-role = "sink";
        sink-pdos = <PDO_FIXED(5000, 100, 0)>;
    };
```

Devicetree USB Type-C Sink Port description



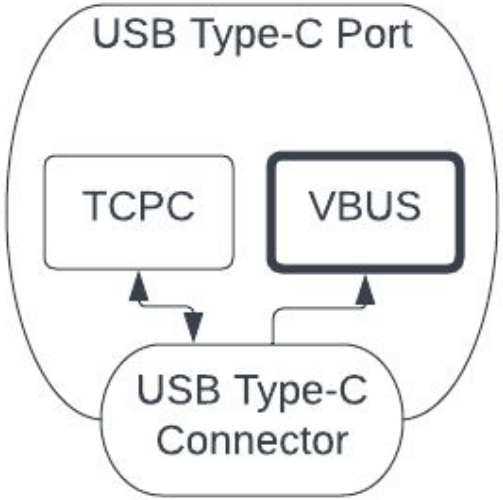

```
port1: usbc-port@1 {
        compatible = "usb-c-connector";
        tcpc = <&ucpd1>;
        vbus = <&vbus1>;
        power-role = "sink";
        sink-pdos = <PDO_FIXED(5000, 100, 0)>;
    };
```

Type-C Port Controller (TCPC) Driver




```
port1: usbc-port@1 {
        compatible = "usb-c-connector";
        tcpc = <&ucpd1>;
        vbus = <&vbus1>;
        power-role = "sink";
        sink-pdos = <PDO_FIXED(5000, 100, 0)>;
    };
```

VBUS Driver




```
port1: usbc-port@1 {
        compatible = "usb-c-connector";
        tcpc = <&ucpd1>;
        vbus = <&vbus1>;
        power-role = "sink";
        sink-pdos = <PDO_FIXED(5000, 100, 0)>;
    };
```

Zephyr® Project
Developer Summit

Application specific data structure

Check if Power Role is supported

```
#define PORT1_NODE          DT_NODELABEL(port1)
#define PORT1_POWER_ROLE    DT_ENUM_IDX(DT_NODELABEL(port1), power_role)

#if (PORT1_POWER_ROLE != TC_ROLE_SINK)
#error "Unsupported board: Only Sink device supported"
#endif
```

```
port1: usbc-port@1 {
       compatible = "usb-c-connector";
       reg = <1>;
       tcpc = <&ucpd1>;
       vbus = <&vbus1>;
       power-role = "sink";
       sink-pdos = <PDO_FIXED(5000, 100, 0)>;
};
```

Application Data Structure (Extract Sink PDOs)

```
static struct port1_data_t {
   uint32_t snk_caps[DT_PROP_LEN(DT_NODELABEL(port1), sink_pdos)];
   int snk_cap_cnt;
} port1_data = {
   .snk_caps = {DT_FOREACH_PROP_ELEM(DT_NODELABEL(port1),
                sink_pdos, SINK_PDO)},
   .snk_cap_cnt = DT_PROP_LEN(DT_NODELABEL(port1), sink_pdos),
};
```

```
port1: usbc-port@1 {
       compatible = "usb-c-connector";
       reg = <1>;
       tcpc = <&ucpd1>;
       vbus = <&vbus1>;
       power-role = "sink";
       sink-pdos = <PDO_FIXED(5000, 100, 0)>;
};
```

Application Data Structure (Additional members)

```
static struct port1_data_t {
      ...
   uint32_t src_caps[PDO_MAX_DATA_OBJECTS];

   int src_cap_cnt;

   atomic_t ps_ready;

} port1_data = {
      ...
   .src_caps = {0},

   .src_cap_cnt = 0,

   .ps_ready = 0
};
```

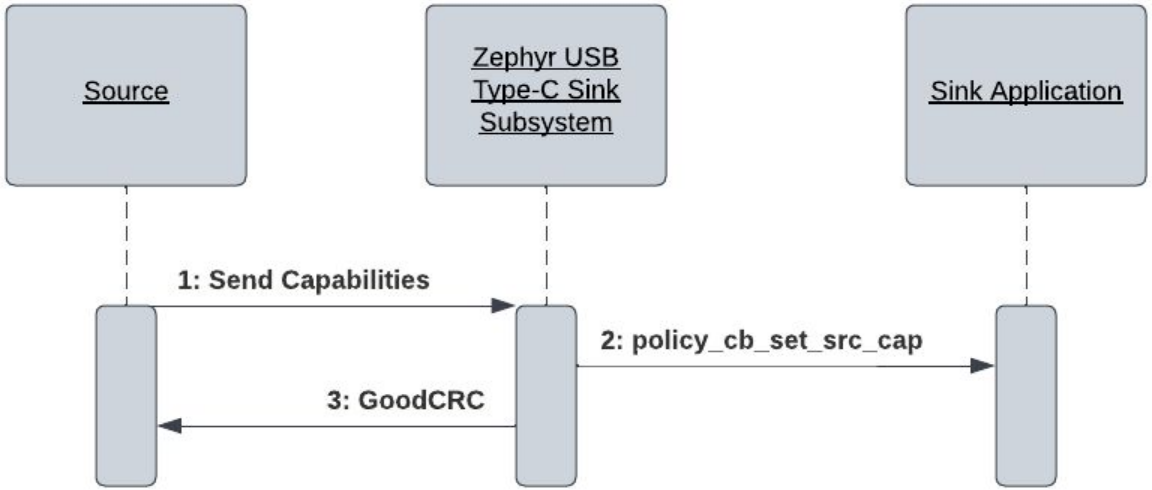
Zephyr® Project
Developer Summit

Policy Callbacks

Minimum Sink Policy Callbacks

- int (***policy_cb_get_snk_cap_t***)(const struct device *dev, uint32_t **pdos, int *num_pdos);
- void (***policy_cb_set_src_cap_t***)(const struct device *dev, const uint32_t *pdos, const int num_pdos);
- uint32_t (***policy_cb_get_rdo_t***)(const struct device *dev);
- bool (***policy_cb_check_t)***(const struct device *dev, const enum usbc_policy_check_t policy_check);
- void (***policy_cb_notify_t***)(const struct device *dev, const enum usbc_policy_notify_t policy_notify);

Power Delivery Negotiation (Step 1)



policy_cb_set_src_cap_t

The Zephyr USB Type-C PD Subsystem uses this callback to send the received Source capabilities to the application.

For this application, the following would be set:

- **port1_data.src_caps**
- **port1_data.src_cap_cnt**

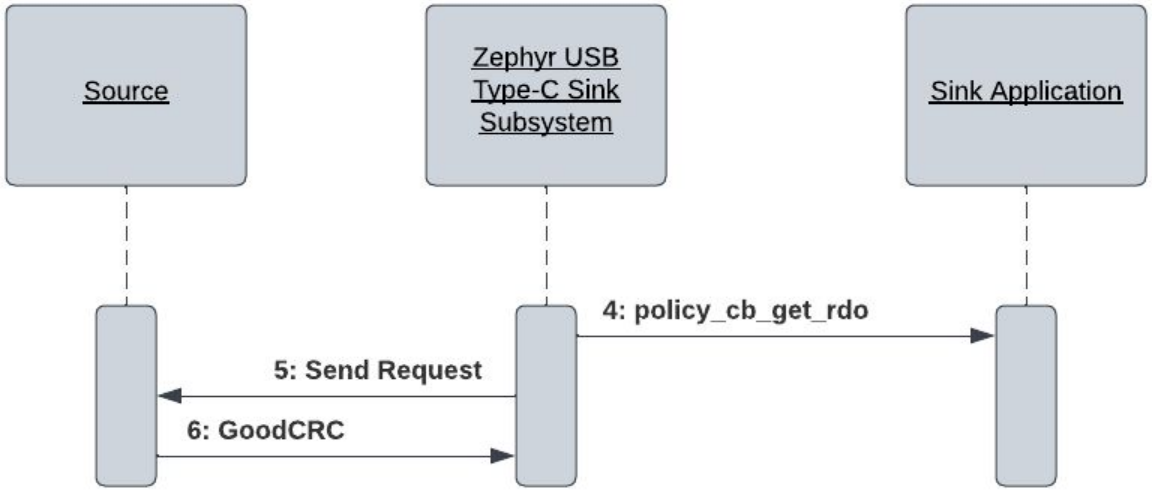
```
static struct port1_data_t {
      ...
   uint32_t src_caps[PDO_MAX_DATA_OBJECTS];
   int src_cap_cnt;
   atomic_t ps_ready;

} port1_data = {
      ...
   .src_caps = {0},
   .src_cap_cnt = 0,
   .ps_ready = 0
};
```

Sample Source Capabilities Could include:

- PDO1: 5V (ALWAYS 5V)
- PDO2: 12
- PDO3: 20

Power Delivery Negotiation (Step 2)



policy_cb_get_rdo_t

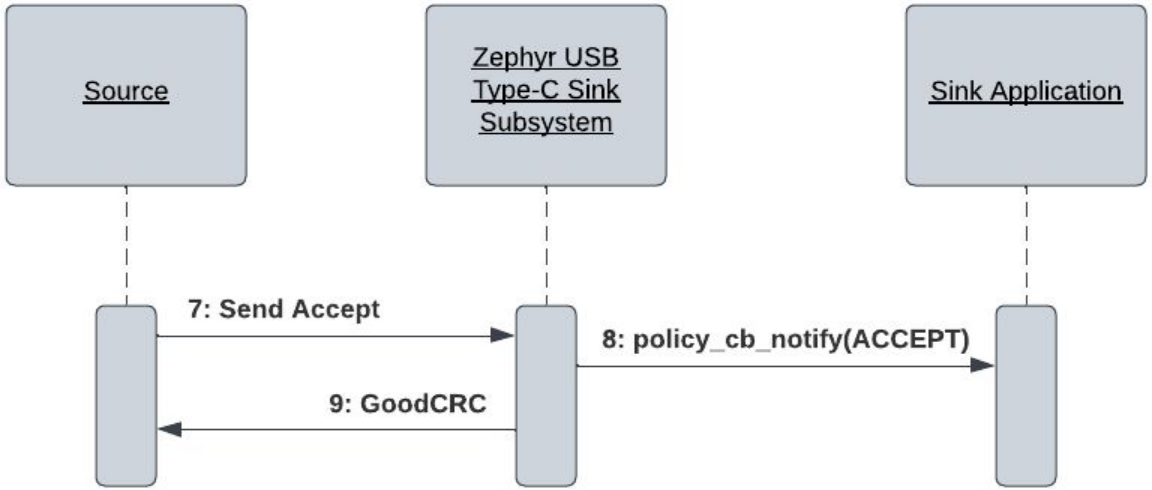
After the Source PDOs are stored, the application must select one of them and build an RDO to return to the stack. Zephyr uses this callback to get that RDO.

```
union pd_rdo rdo;

/* Maximum operating current 100mA */
rdo.fixed.min_or_max_operating_current = PD_CONVERT_MA_TO_FIXED_PDO_CURRENT(100);
/* Operating current 100mA */
rdo.fixed.operating_current = PD_CONVERT_MA_TO_FIXED_PDO_CURRENT(100);
/* Object position 1 (5V PDO) */
rdo.fixed.object_pos = 1;

return rdo.raw_value;
```

Power Delivery Negotiation (Step 3)

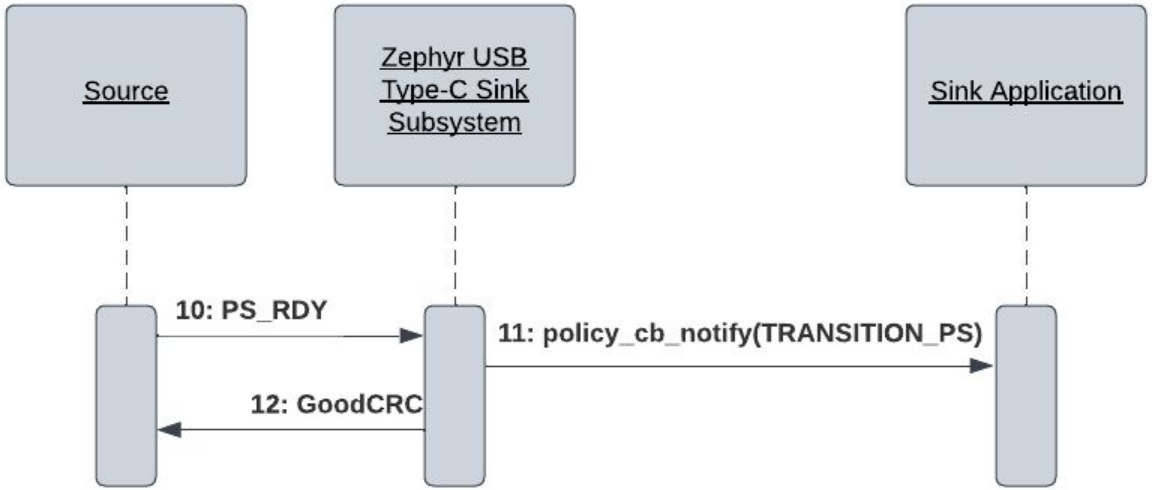


policy_cb_notify_t

As the Zephyr USB Type-C subsystem executes, it notifies the application certain changes occur. And that's what this callback is used for. In the following, the application is notified that an Accept message was received.

```
case MSG_ACCEPT_RECEIVED:
      LOG_INF("Source accepted the request");
      break;
case TRANSITION_PS:
      atomic_set_bit(&dpm_data->ps_ready, 0);
      break;
case NOT_PD_CONNECTED:
      break;
case POWER_CHANGE_0A0:
      LOG_INF("PWR 0A");
      break;
case POWER_CHANGE_DEF:
      LOG_INF("PWR DEF");
      break;
```

Power Delivery Negotiation (Step 4)

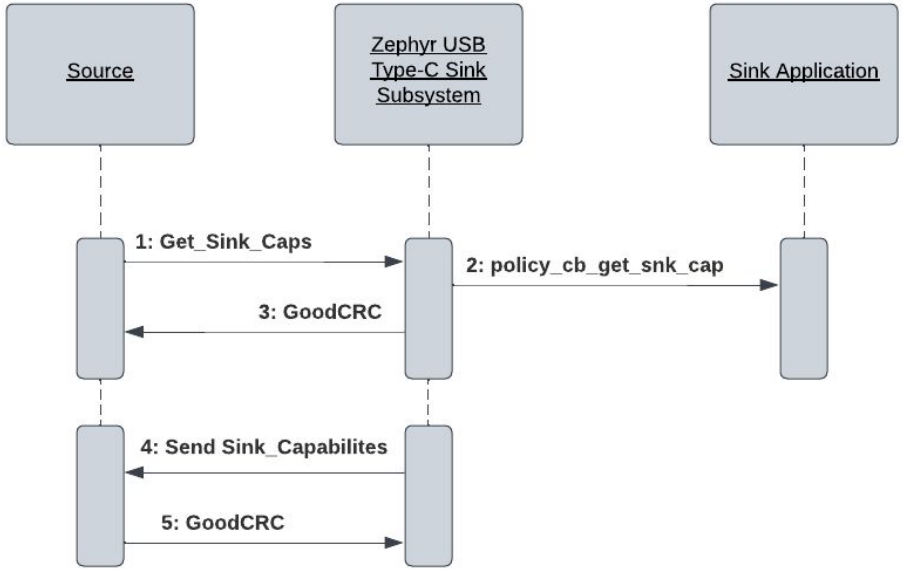


policy_cb_notify_t

As the Zephyr USB Type-C subsytem executes, it notifies the application certain changes occur. And that's what this callback is used for. In the following, the application is notified that an PS_RDY message was received.

```
case MSG_ACCEPT_RECEIVED:
      LOG_INF("Source accepted the request");
      break;
case TRANSITION_PS:
      atomic_set_bit(&dpm_data->ps_ready, 0);
      break;
case NOT_PD_CONNECTED:
      break;
case POWER_CHANGE_0A0:
      LOG_INF("PWR 0A");
      break;
case POWER_CHANGE_DEF:
      LOG_INF("PWR DEF");
      break;
```

Get Sink Capabilities



policy_cb_get_snk_cap_t

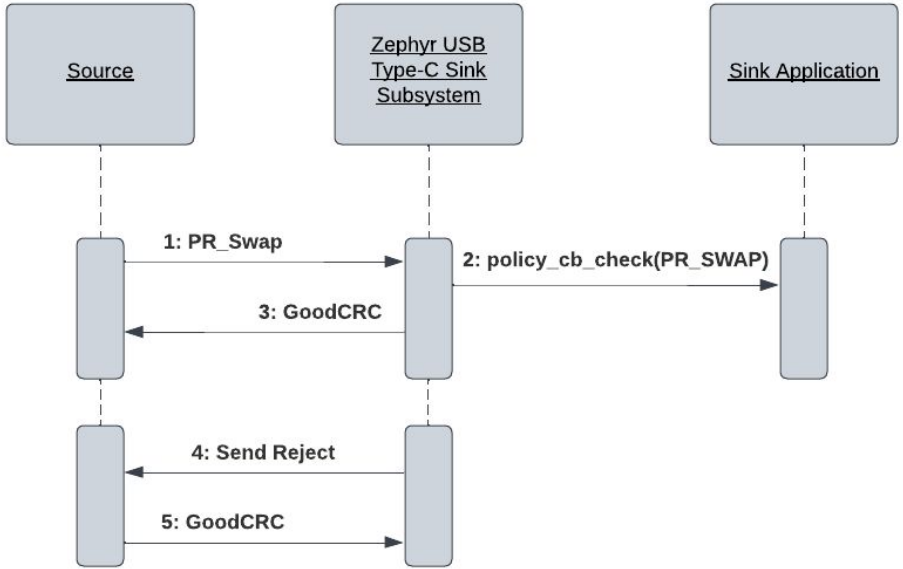
The Zephyr USB Type-C PD Stack uses this callback to get the Sink PDOs and their count.

For this application, the following is returned:

- **port1_data.snk_caps**
- **port1_data.snk_cap_cnt**

```
static struct port1_data_t {
   uint32_t snk_caps[DT_PROP_LEN(DT_NODELABEL(port1), sink_pdos)];
   int snk_cap_cnt;
} port1_data = {
   .snk_caps = {DT_FOREACH_PROP_ELEM(DT_NODELABEL(port1),
                sink_pdos, SINK_PDO)},
   .snk_cap_cnt = DT_PROP_LEN(DT_NODELABEL(port1), sink_pdos),
};
```

Power Role Swap to Source



policy_cb_check_t

As the Zephyr USB Type-C stack executes, it needs to check with application before taken certain actions. And that's what this callback is used for.

```
switch (policy_check) {
case CHECK_POWER_ROLE_SWAP:
	/* Reject power role swaps */
	return false;
case CHECK_DATA_ROLE_SWAP_TO_DFP:
	/* Reject data role swap to DFP */
	return false;
case CHECK_DATA_ROLE_SWAP_TO_UFP:
	/* Accept data role swap to UFP */
	return true;
case CHECK_SNK_AT_DEFAULT_LEVEL:
	/* The device is at the default power level */
	return true;
default:
	/* Reject all other policy checks */
	return false;
}
```

Register the callbacks with the following API

- `usbc_set_policy_cb_set_src_cap`
- `usbc_set_policy_cb_get_rdo`
- `usbc_set_policy_cb_get_snk_cap`
- `usbc_set_policy_cb_notify`
- `usbc_set_policy_cb_check`

Register the Application's data structure

- `usbc_set_dpm_data`

After registration, the Application's data structure can be retrieved with the following:

- usbc_get_dpm_data

Start the Zephyr USB Type-C stack

- `usbc_start`

Zephyr USB Type-C Samples

- `Sample Sink: zephyr/samples/subsys/sink`
- `Sample Source: zephyr/samples/subsys/source`

Zephyr USB Type-C Boards

B-G747E-DPOW11 (Sink ONLY)

STM32G081B-EVAL (Source and Sink)

See Zephyr Supported Boards For details: https://docs.zephyrproject.org/latest/boards/index.html

Thank you!