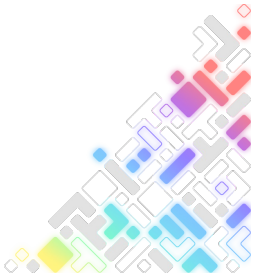
Zephyr® Project
Developer Summit

Charging a Battery with Zephyr

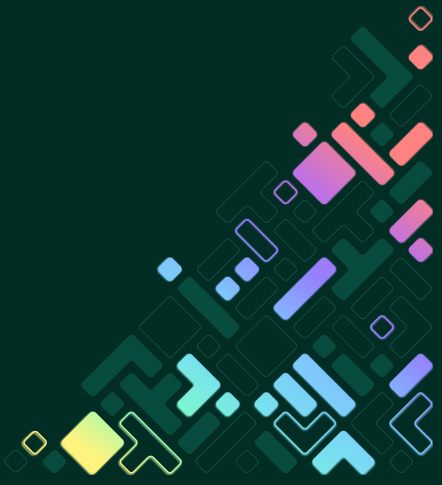
A primer on charging portable embedded systems running Zephyr and an introduction of the charging API

Agenda

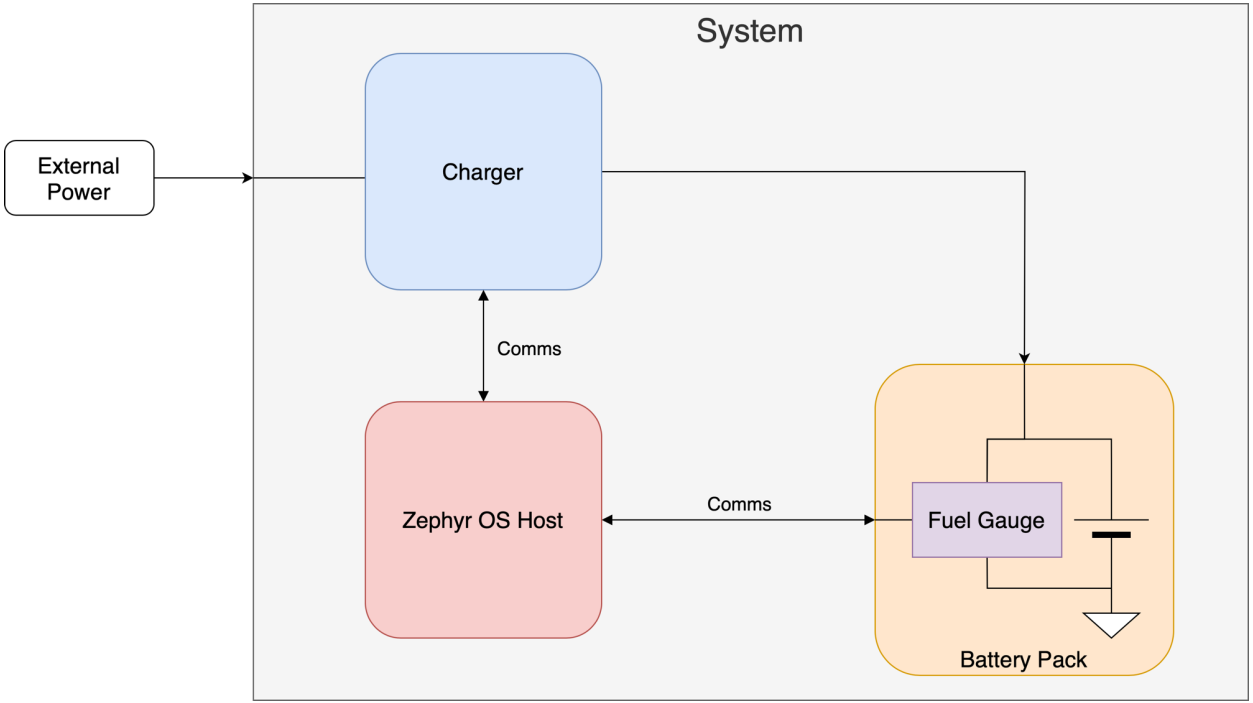
- Charging Fundamentals
  - What does a system charging tree look like?
  - What is a charging peripheral?
  - What is a charge profile?
- Where does Zephyr come in?
  - What was the status quo in Zephyr?
  - Charging Device API
  - Battery DeviceTree
- Charger sample application walkthrough



Charging Fundamentals



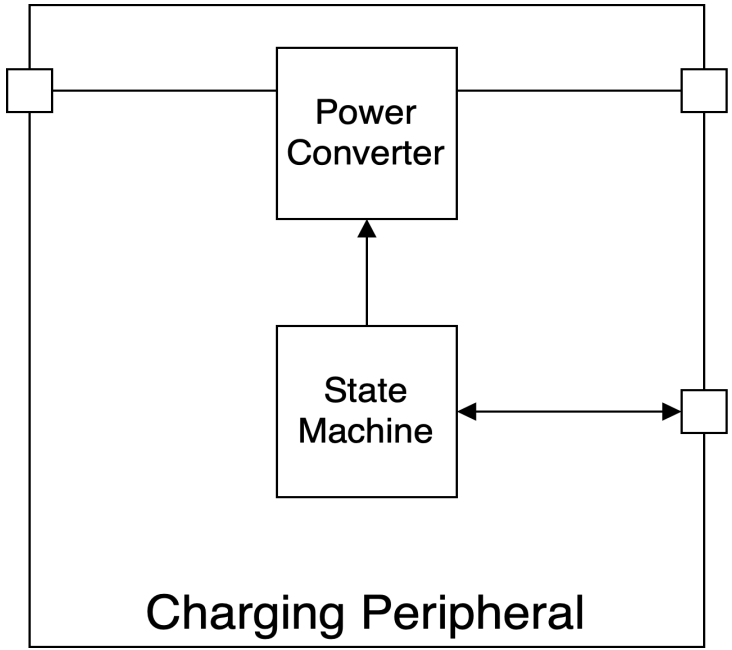
System Charging Tree



Charging Peripherals

External input power provided to the system

Ex.
- Barrel Jack
- USB
- Solar Panel



Power output sent downstream

Ex.
- Battery Pack
- System Rail

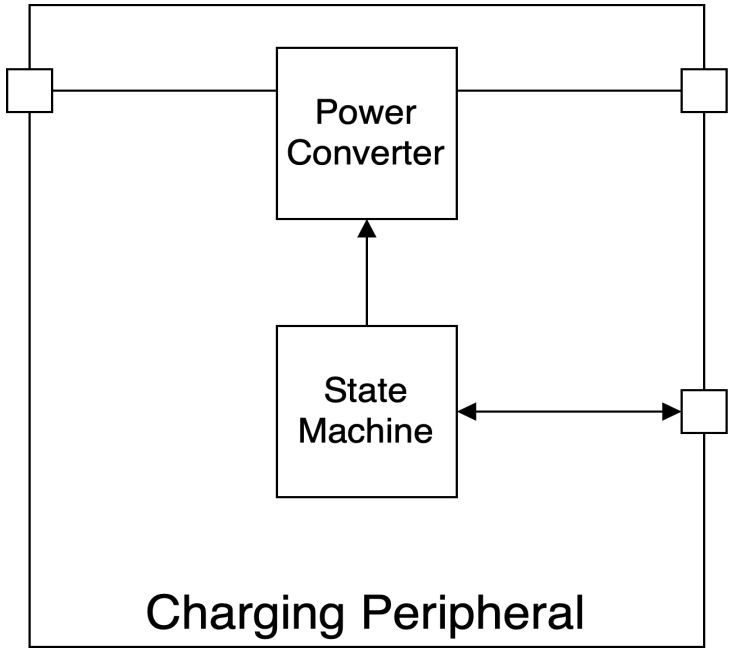
Control port to Zephyr Host

Ex.
- Set charge profile
- Set protections
- Get status info

Charging Peripherals

External input power provided to the system

Ex.
- Barrel Jack
- USB
- Solar Panel



Power output sent downstream

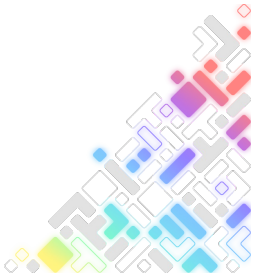
Ex.
- Battery Pack
- System Rail

Control port to Zephyr Host

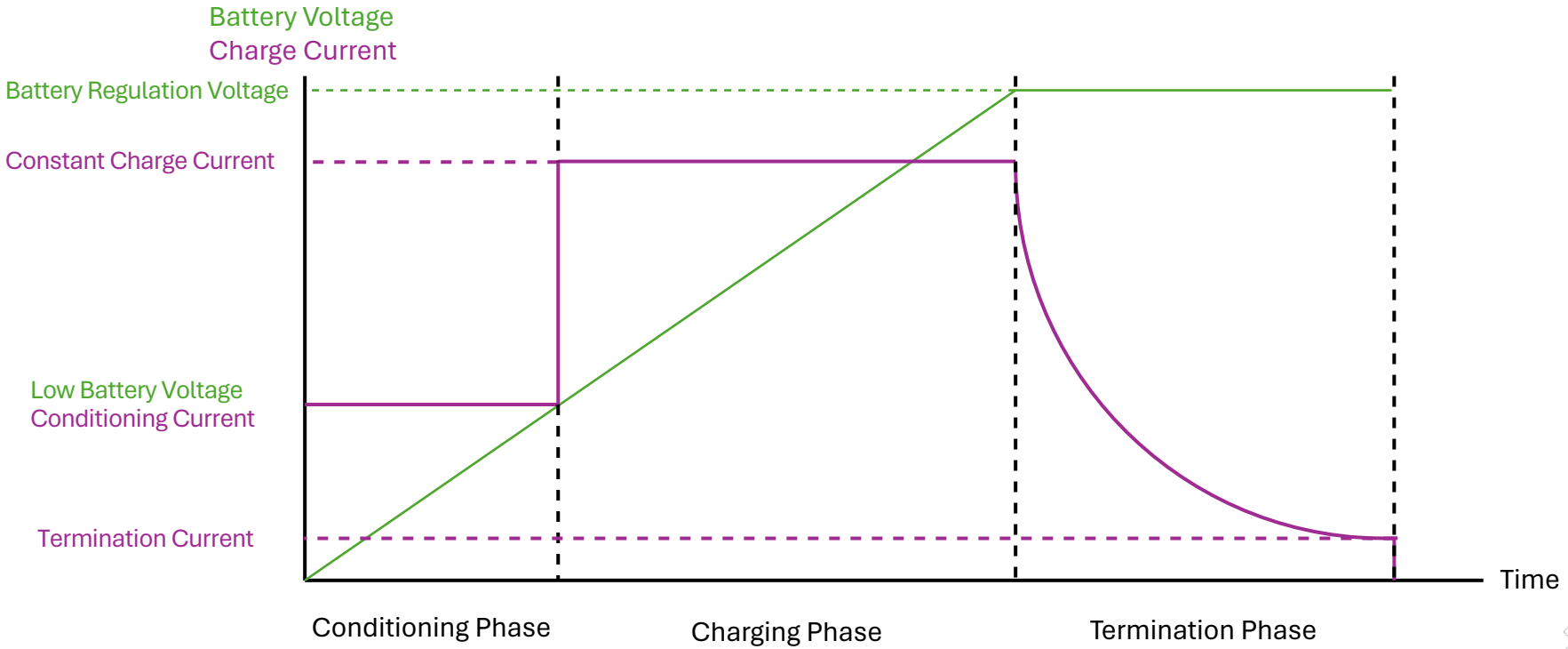
Ex.
- **Set charge profile**
- Set protections
- Get status info

What is charge profile?

- A charge profile is a set of instructions from the battery pack's specification describing how to charge the battery safely and effectively
- The charging peripheral is configured according to the given charge profile to produce the desired output while charging the battery
- Under ideal conditions the charge profile is static, but real life is not ideal
  - Temperature
  - Time
  - Supply impedance



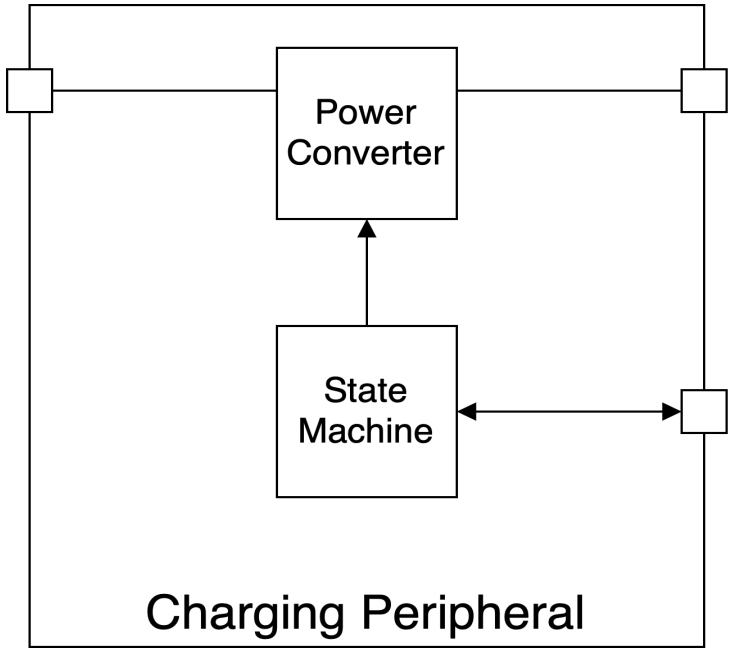
A Typical Charge Profile



Charging Peripherals

External input power provided to the system

Ex.
- Barrel Jack
- USB
- Solar Panel



Power output sent downstream

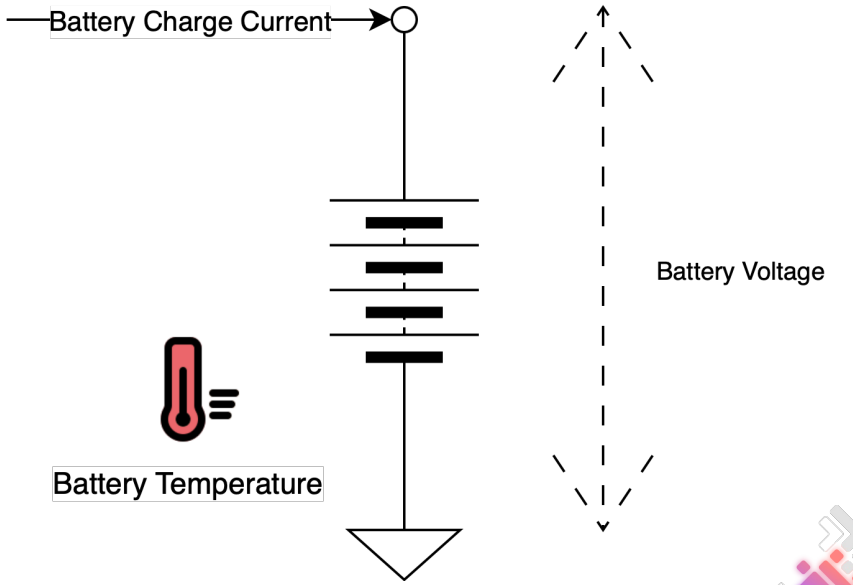
Ex.
- Battery Pack
- System Rail

Control port to Zephyr Host

Ex.
- Set charge profile
- **Set protections**
- Get status info

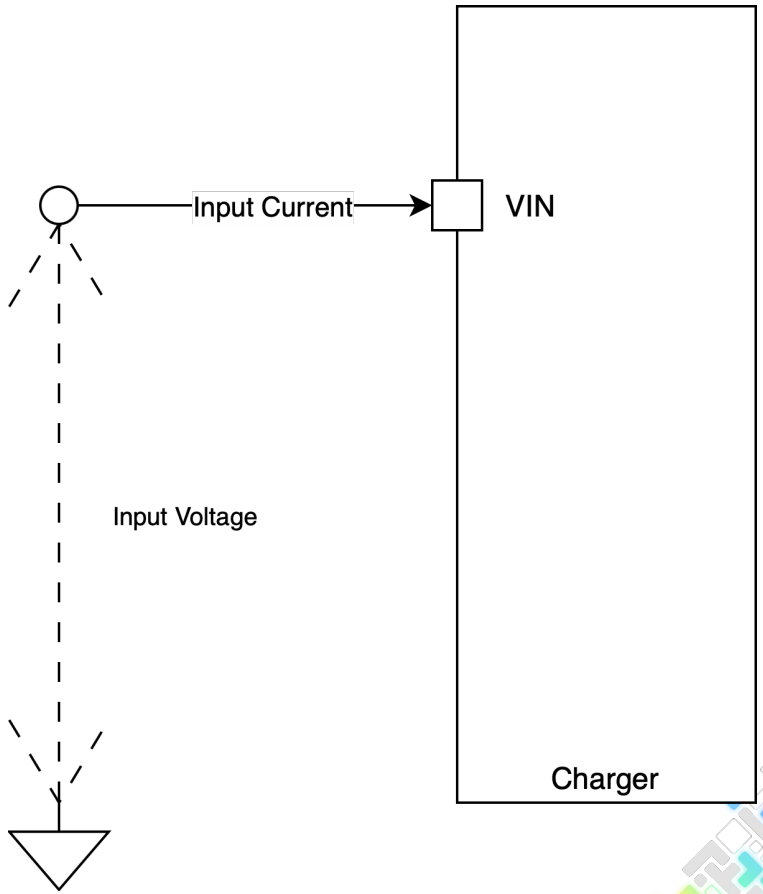
Charger Protections

- The Zephyr host sets battery protections
  - Charge overcurrent limiting
  - Battery overvoltage limiting
  - Responding to changes in temperature

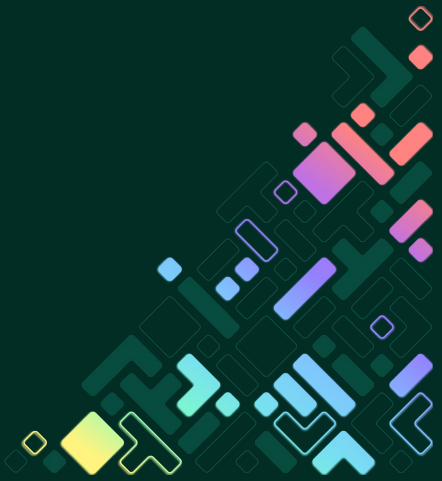


Charger Protections

- The Zephyr host sets charger device protections
  - Input overcurrent limiting and regulation
  - Input overvoltage limiting
  - Input undervoltage regulation (line loading)



Charger Driver API



Zephyr Status Quo and Linux Power Supply Class

- Prior to implementing the *charger_driver_api* there was no dedicated interface for facilitating a battery charge cycle
- A proposed charging API to pair with the existing *fuel_gauge_driver_api* had already been discussed in the *fuel_gauge_driver_api* RFC
- The Linux kernel offers the *power_supply_class* for runtime access and battery.yaml for boot time configuration of both fuel gauges and chargers
- Both the *charger_driver_api* and *fuel_gauge_api* were inspired by the *power_supply_class* with some additional improvements specific to Zephyr OS and its applications

Charging Device API

- *charger.h* describes the *charger_driver_api* and its runtime handlers
  - *get_property* – grabs a property value from a client driver
  - *set_property* – sets a property value from a client driver
  - *charge_enable* – enables and disables the charge cycle

```
/**
 * @brief Charging device API
 *
 * Caching is entirely on the onus of the client
 */
__subsystem struct charger_driver_api {
        charger_get_property_t get_property;
        charger_set_property_t set_property;
        charger_charge_enable_t charge_enable;
};
```

Charging Runtime Properties

- *charger.h* contains an enum of runtime charger properties
- These properties are largely inherited from the Linux power supply class
- The enumeration of properties is extensible allowing for custom vendor specific properties

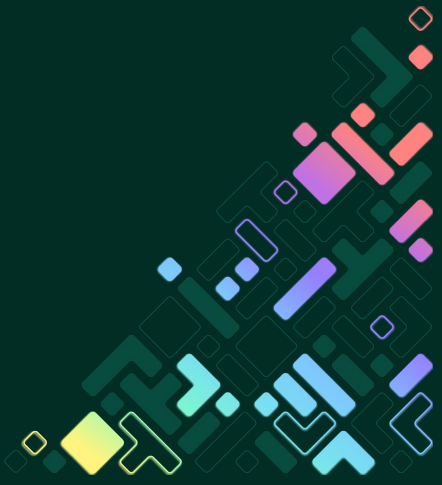
```
CHARGER_PROP_ONLINE
CHARGER_PROP_PRESENT
CHARGER_PROP_STATUS
CHARGER_PROP_CHARGE_TYPE
CHARGER_PROP_HEALTH
CHARGER_PROP_CONSTANT_CHARGE_CURRENT_UA
CHARGER_PROP_PRECHARGE_CURRENT_UA
CHARGER_PROP_CHARGE_TERM_CURRENT_UA
CHARGER_PROP_CONSTANT_CHARGE_VOLTAGE_UV
CHARGER_PROP_INPUT_REGULATION_CURRENT_UA
CHARGER_PROP_INPUT_REGULATION_VOLTAGE_UV
CHARGER_PROP_INPUT_CURRENT_NOTIFICATION
CHARGER_PROP_DISCHARGE_CURRENT_NOTIFICATION
CHARGER_PROP_SYSTEM_VOLTAGE_NOTIFICATION_UV
CHARGER_PROP_STATUS_NOTIFICATION
CHARGER_PROP_ONLINE_NOTIFICATION
CHARGER_PROP_COMMON_COUNT
```

Battery DeviceTree Properties

- *battery.yaml* describes the static battery characteristics to be applied at initialization
- Largely inherited from *battery.yaml* in Linux
- This dt-binding is shared between clients of the both the *charger_driver_api* and the *fuel_gauge_driver_api*

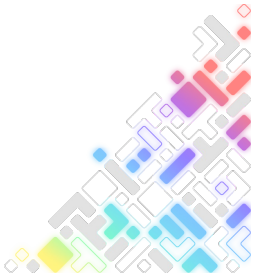
```
properties:
  precharge-current-microamp:
    type: int
    description: current for pre-charge phase

  charge-term-current-microamp:
    type: int
    description: current for charge termination phase

  constant-charge-current-max-microamp:
    type: int
    description: maximum constant input current

  constant-charge-voltage-max-microvolt:
    type: int
    description: maximum constant input voltage
```

Charger Sample Application

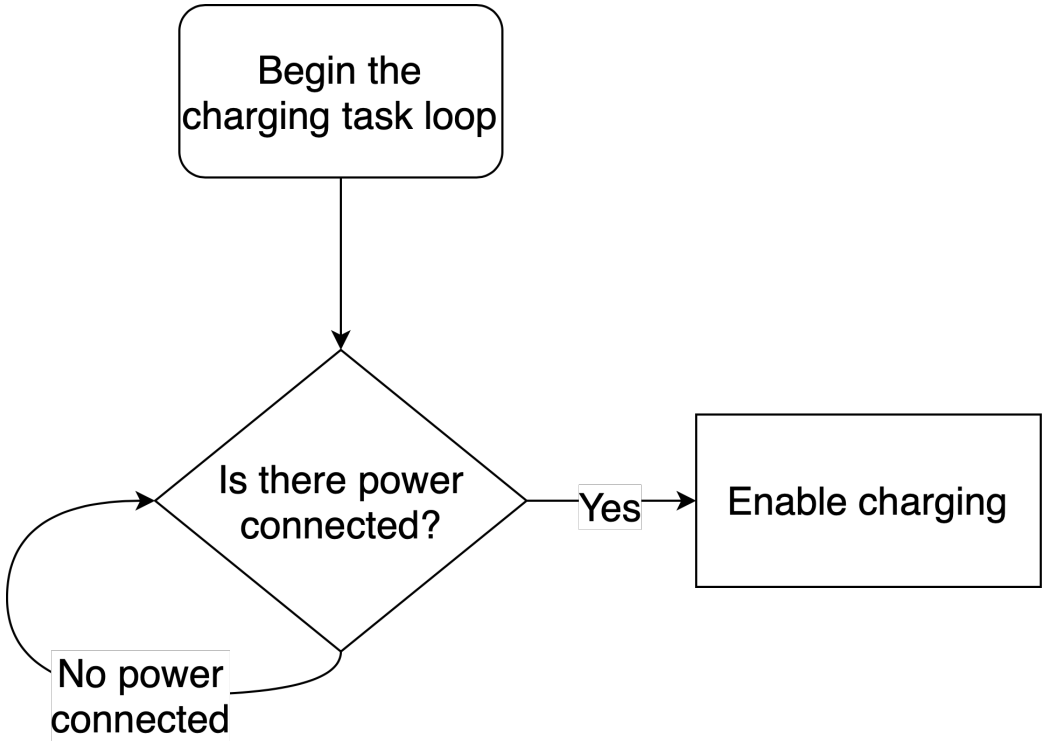


Charger Sample Application

- A charging sample application was upstreamed along with the *charger_driver_api*
- The sample application is a simple charging task loop intended to show developers how to leverage the *charger_driver_api* for their own applications
- The charging task is an infinite loop that is broken out of only when charging is complete
- Inside of the charging task loop there is a do-while loop polling for external power and a switch-case for handling the charging peripheral's status



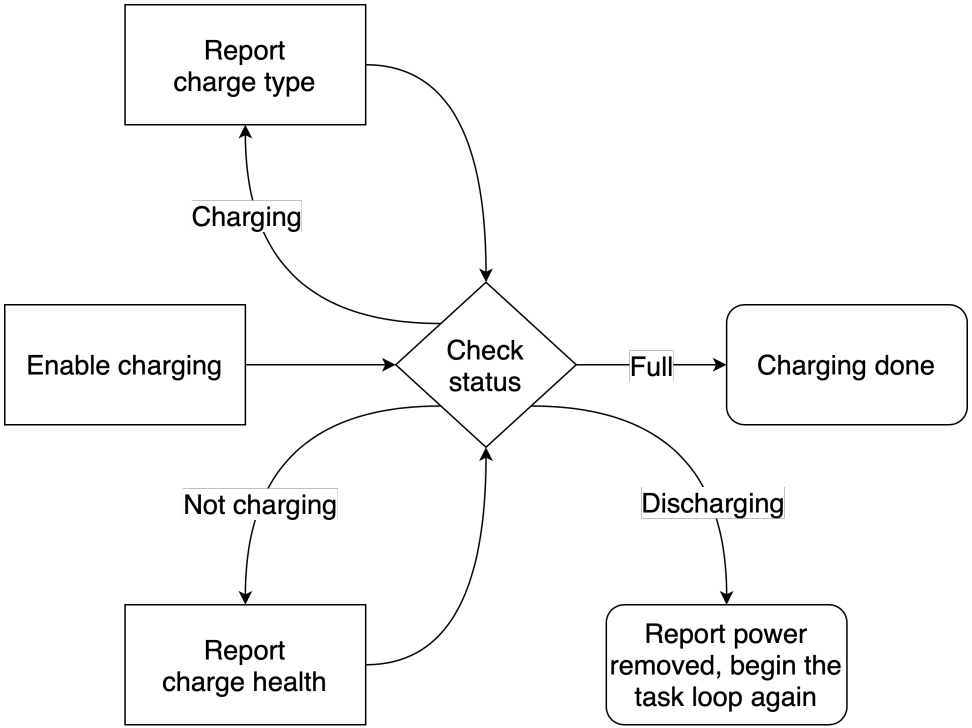
Charger Sample Application – Polling for power



Charger Sample Application – Polling for power

```
<...>

/* Poll until external power is presented to the charger */
do {
            ret = charger_get_prop(chgdev, CHARGER_PROP_ONLINE, &val);
            if (ret < 0) {
                        return ret;
            }

            k_msleep(100);
} while (val.online == CHARGER_ONLINE_OFFLINE);

ret = charger_charge_enable(chgdev, true);
if (ret == -ENOTSUP) {
            printk("Enabling charge not supported, assuming auto charge enable\n");
            continue;
} else if (ret < 0) {
            return ret;
}

<...>
```

Charging Sample App – Attempt charging

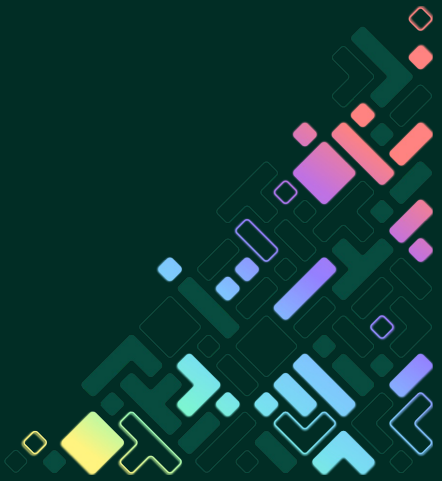


Charging Sample App – Attempt charging

```
switch (val.status) {
case CHARGER_STATUS_CHARGING:
    printk("Charging in progress...\n");

    ret = charger_get_prop(chgdev, CHARGER_PROP_CHARGE_TYPE, &val);
    if (ret < 0) {
        return ret;
    }

    printk("Device \"%s\" charge type is %d\n",
           chgdev->name, val.charge_type);
    break;
case CHARGER_STATUS_NOT_CHARGING:
    printk("Charging halted...\n");

    ret = charger_get_prop(chgdev, CHARGER_PROP_HEALTH, &val);
    if (ret < 0) {
        return ret;
    }

    printk("Device \"%s\" health is %d\n",
           chgdev->name, val.health);
    break;
```

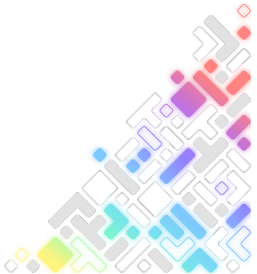
```
case CHARGER_STATUS_FULL:
    printk("Charging complete!");
    return 0;
case CHARGER_STATUS_DISCHARGING:
    printk("External power removed, discharging\n");

    ret = charger_get_prop(chgdev,
                           CHARGER_PROP_ONLINE,
                           &val);
    if (ret < 0) {
        return ret;
    }
    break;
default:
    return -EIO;
}
```

Closing

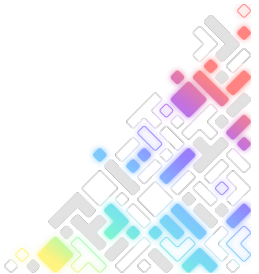


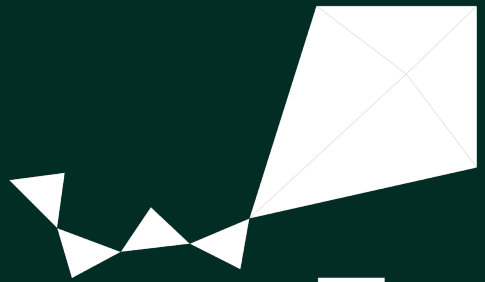
Future Subsystem Improvements

- Generic GPIO charger driver – Add a driver for a GPIO controlled charger peripheral as exists in the Linux *power_supply* class
- Charger sample app improvements – tidy up the sample application and add in more of the properties
- Adds support for charger telemetry – chargers often integrate ADCs with useful telemetry in real units



Q&A?





Zephyr® Project

Developer Summit