componentManager

Release 1.0

CTRL

Jun 09, 2022

CONTENTS

Contents:

CHAPTER

ONE

COMPONENT MANAGER

The Component Manager has been designed to configure multiple devices at the same time in a generic fashion. A set of devices is hereafter referred to as *component*. The list of handled devices is defined by the operator in the property **deviceNames**. The main principles of the component manager device are:

```
- The Component Manager uses the ``ConfigurationManager`` device to store and retrieve␣
↪configurations
- The Component Manager coordinates the save of multiple runtime device configurations,␣
↪the so-called component configuration
- The Component Manager can apply a given configuration set by a unique `name` to the␣
↪component.
```

- The for each device is retrieved and each configuration is filtered such that only the values which are allowed in the current device state are applied.
- **Instantiate the component from a given configuration set by a unique *name***
  - A class schema is retrieved from the server and the configuration is filtered for reconfigurable and init only parameters

Note: A generic component manager does not have the knowledge in which device states configurations can be applied (safe state), etc.. It is detached from any specific device knowledge. Any extra knowledge and actions have to be implemeneted in more specific component managers (e.g. move after apply of configuration).

1.1 Generic Slots

Every generic slot listed here (key - displayedName) is acting over the device listed in the **deviceNames** property, e.g. all devices controlled by the component manager.

- **applyConfigurations** ("Apply configurations") Applies configured configuration to the component. Only validates the state dependent reconfiguration and the attribute *options*.
- **startInstantiate** ("Instantiate") Starts instantiating the component from a known configuration by *name*. Only devices that are not already online are instantiated. Alternatively, the **Force Shutdown** property can be checked to force an initial shutdown of the whole component, thus allowing the loading of the desired configuration in all the selected devices.
- **stopInstantiate** ("Stop Instantiate") Stops a running instantiate action
- **saveConfigurations** ("Save configurations") Manages the savings of all device configurations under a snapshot in the `Configuration Manager` device.

- **listConfigurations** ("List configurations") Lists all configurations which were saved (snapshot) for the set of devices which the component manager controls. The status is displayed in the table element.

1.2 Technical: Functions available for subclassing

- **async def get_reconfigurable_config(devices: ist[str], config_name: str) -> dict[str, Hash]**
  - Return the configurations for all devices in a *dict {deviceId: config}*
- **async def check_config_name_taken(devices: list[str], config_name: str) -> Tuple[success bool, status str]**
  - Check if any of the configuration names has been taken for the *devices*. If the configuration is overwritable and allowed to write, will return False. The status string will provide detailed information.
- **async def list_configuration_sets(devices) -> Tuple[success bool, items list]**
  - Will retrieve the list of configurations for the devices on success. Retrieves an empty list if no success or no configurations are there.
- **async def check_devices_alive() -> bool**
  - Pings all devices of the component manager and checks if they are alive. Returns *True* if the devices replied, otherwise *False*

1.3 Technical Design Choices

- The Component Manager does not create *proxies* for the devices it manages. This would unnecessarily create more messages (plain duplication) on the broker. Instead a **slotPing** to every device is executed before every action to validate.
- The Component Manager makes heavy use of the function **allCompleted**, which will cancel pending tasks after a given timeout (mostly 4 seconds).