The [source](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) folder contains the core implementation of the sink service. Each file has a specific purpose, and together they implement the functionality of the service. Here's an explanation of each file and how they work together:

### **1.**[**config.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**and**[**config.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Purpose: These files implement the configuration module of the sink service.

Details:

[config.h](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) declares the interface for initializing and closing the configuration module.

[config.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) implements the configuration logic, including:

Reading and writing configuration properties (e.g., node role, network channel, sink cost).

Handling D-Bus properties and methods for configuration.

Sending D-Bus signals when the stack starts or stops.

Managing static configuration values (e.g., stack profile, MTU, channel range).

It uses helper macros from [config\_macros.h](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) to simplify the implementation of D-Bus property handlers.

Role in the system: This module provides an interface for managing and querying the configuration of the sink.

#include <stdint.h>

#include <stdio.h>

// Define the sink\_config\_t structure

typedef struct

{

uint16\_t stack\_profile;

uint16\_t hw\_magic;

uint16\_t ac\_range\_min;

uint16\_t ac\_range\_max;

uint16\_t app\_config\_max\_size;

uint16\_t version[4];

uint8\_t max\_mtu;

uint8\_t ch\_range\_min;

uint8\_t ch\_range\_max;

uint8\_t pdu\_buffer\_size;

} sink\_config\_t;

int main()

{

// Create an instance of sink\_config\_t

sink\_config\_t config;

// Assign demo values to the fields

config.stack\_profile = 1; // Example stack profile ID

config.hw\_magic = 0xABCD; // Example hardware magic number

config.ac\_range\_min = 10; // Minimum application configuration range

config.ac\_range\_max = 100; // Maximum application configuration range

config.app\_config\_max\_size = 256; // Maximum application configuration size

config.version[0] = 1; // Major version

config.version[1] = 0; // Minor version

config.version[2] = 3; // Patch version

config.version[3] = 42; // Build number

config.max\_mtu = 128; // Maximum Transmission Unit

config.ch\_range\_min = 11; // Minimum channel range

config.ch\_range\_max = 26; // Maximum channel range

config.pdu\_buffer\_size = 64; // Protocol Data Unit buffer size

// Print the values to verify

printf("Stack Profile: %u\n", config.stack\_profile);

printf("HW Magic: 0x%X\n", config.hw\_magic);

printf("AC Range Min: %u\n", config.ac\_range\_min);

printf("AC Range Max: %u\n", config.ac\_range\_max);

printf("App Config Max Size: %u\n", config.app\_config\_max\_size);

printf("Version: %u.%u.%u Build %u\n", config.version[0], config.version[1], config.version[2], config.version[3]);

printf("Max MTU: %u\n", config.max\_mtu);

printf("Channel Range Min: %u\n", config.ch\_range\_min);

printf("Channel Range Max: %u\n", config.ch\_range\_max);

printf("PDU Buffer Size: %u\n", config.pdu\_buffer\_size);

return 0;

}

### **2.**[**data.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**and**[**data.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Purpose: These files implement the data module of the sink service.

Details:

[data.h](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) declares the interface for initializing and closing the data module.

[data.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) implements the logic for:

Sending messages to the sink.

Handling incoming messages from the sink and broadcasting them as D-Bus signals.

Managing message queuing and downlink limits.

It uses the Wirepas C-Mesh API (wpc.h) to interact with the sink.

Role in the system: This module handles data communication between the sink and the D-Bus interface.

### **3.**[**otap.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**and**[**otap.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Purpose: These files implement the OTAP (Over-The-Air Programming) module of the sink service.

Details:

[otap.h](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) declares the interface for initializing and closing the OTAP module.

[otap.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) implements the logic for:

Uploading, clearing, and processing scratchpads (firmware images).

Managing scratchpad-related properties (e.g., stored length, CRC, sequence number).

Handling D-Bus methods for OTAP operations.

It uses the Wirepas C-Mesh API (wpc.h) to interact with the sink for OTAP operations.

Role in the system: This module provides an interface for managing firmware updates on the sink.

### **4.**[**config\_macros.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Purpose: This file defines helper macros to simplify the implementation of D-Bus property handlers.

Details:

Provides macros for generating read and write handlers for D-Bus properties.

Includes a macro for setting error messages in D-Bus responses.

Role in the system: This file reduces boilerplate code in the [config.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) file and ensures consistency in D-Bus property handling.

### **5.**[**main.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Purpose: This is the entry point of the sink service.

Details:

Initializes the sink service by:

Parsing environment variables and command-line arguments for configuration.

Establishing a connection with the sink via UART.

Setting up the D-Bus connection and registering the config, data, and otap modules.

Implements the main event loop to process D-Bus requests and signals.

Handles graceful shutdown on receiving termination signals (e.g., SIGINT, SIGTERM).

Role in the system: This file orchestrates the initialization, operation, and shutdown of the sink service.

### **6.**logger.h**(Referenced but not provided)**

Purpose: Provides logging functionality for the sink service.

Details:

Each module uses logging macros (e.g., LOGE, LOGI, LOGD) to log errors, informational messages, and debug messages.

Role in the system: Helps in debugging and monitoring the service.

### **7.**wpc.h**(Referenced but not provided)**

Purpose: This is the Wirepas C-Mesh API, which provides functions to interact with the sink hardware.

Details:

Used by all modules (config, data, otap) to communicate with the sink.

Provides functions for reading and writing configuration, sending and receiving data, and managing OTAP operations.

Role in the system: Acts as the bridge between the sink service and the sink hardware.

### **How It Works Overall**

Initialization:

The [main.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) file initializes the service by:

Parsing configuration from environment variables and command-line arguments.

Establishing a UART connection with the sink.

Setting up the D-Bus connection.

Initializing the config, data, and otap modules.

D-Bus Interface:

Each module (config, data, otap) registers its own D-Bus interface and methods.

The service listens for D-Bus method calls and signals and processes them in the main event loop.

Communication with the Sink:

The service uses the Wirepas C-Mesh API (wpc.h) to communicate with the sink hardware.

The data module handles data communication, while the config and otap modules manage configuration and firmware updates.

Event Loop:

The main event loop in [main.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) processes D-Bus requests and signals.

It also handles termination signals to gracefully shut down the service.

Shutdown:

On shutdown, the [main.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) file:

Closes the config, data, and otap modules.

Closes the D-Bus connection.

Closes the UART connection with the sink.7

### **Summary**

The [source](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) folder implements a modular sink service that interacts with a physical device (sink) via UART and exposes its functionality through a D-Bus interface.

The config, data, and otap modules handle specific aspects of the service, while [main.c](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) orchestrates the overall operation.

The service is designed to be robust, modular, and extensible, with clear separation of concerns between modules.

**ANALOGY**

Alright, let’s break this down with a funny analogy! Imagine the sink service is like a restaurant kitchen that communicates with its waiters (D-Bus) and customers (the physical device connected via UART). Each file in the [source](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) folder plays a specific role in this kitchen. Let’s meet the team:

### **1.**[**config.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**and**[**config.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**– The Head Chef**

The Head Chef (config module) is in charge of setting up the kitchen. They decide:

What ingredients (configuration properties like node role, network channel) are available.

How the kitchen operates (stack profile, MTU, etc.).

When the kitchen opens or closes (stack start/stop signals).

They use recipe cards (macros from [config\_macros.h](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)) to make sure every dish (D-Bus property) is prepared consistently.

If a waiter (D-Bus) asks, "What’s the special today?" the Head Chef knows exactly how to respond.

### **2.**[**data.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**and**[**data.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**– The Sous Chef**

The Sous Chef (data module) handles the actual food orders (data messages).

They take orders from the waiters (send messages to the sink).

They shout out when food is ready (broadcast incoming messages as D-Bus signals).

They make sure the kitchen doesn’t get overwhelmed (manage message queues and downlink limits).

If a waiter says, "Table 5 wants spaghetti," the Sous Chef ensures the spaghetti is cooked and delivered on time.

### **3.**[**otap.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**and**[**otap.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**– The Pastry Chef**

The Pastry Chef (OTAP module) is in charge of desserts (firmware updates).

They bake scratchpads (upload firmware images).

They clear out old desserts (clear scratchpads).

They make sure the desserts are perfect (check CRC, sequence numbers).

If a waiter says, "The customer wants a chocolate cake," the Pastry Chef ensures it’s baked, frosted, and delivered without a hitch.

### **4.**[**config\_macros.h**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**– The Recipe Book**

The Recipe Book (config\_macros.h) is the secret sauce of the kitchen.

It contains shortcuts (macros) for the Head Chef to quickly prepare dishes (D-Bus property handlers).

For example, instead of writing "boil water, add pasta, cook for 10 minutes" every time, the Recipe Book says, "Use the COOK\_PASTA macro."

### **5.**[**main.c**](vscode-file://vscode-app/snap/code/187/usr/share/code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**– The Restaurant Manager**

The Restaurant Manager (main.c) runs the whole show.

They open the restaurant (initialize the service).

They hire the chefs (initialize config, data, and otap modules).

They make sure the waiters (D-Bus) are ready to take orders.

They handle customer complaints (termination signals like SIGINT).

If the manager doesn’t show up, the restaurant doesn’t open!

### **6.**logger.h**– The Gossiping Waiter**

The Gossiping Waiter (logger) tells everyone what’s happening in the kitchen.

If the Head Chef burns the soup, the waiter shouts, "Error: Soup is ruined!"

If the Sous Chef finishes an order, the waiter whispers, "Info: Order delivered."

Without the Gossiping Waiter, no one knows what’s going on.

### **7.**wpc.h**– The Delivery Truck**

The Delivery Truck (Wirepas C-Mesh API) brings ingredients (data) to the kitchen and takes out finished dishes (messages).

All the chefs rely on the truck to deliver what they need.

If the truck breaks down, the kitchen grinds to a halt.

### **How It All Works**

Opening the Restaurant:

The Manager (main.c) opens the restaurant, hires the chefs, and sets up the waiters (D-Bus).

They check if the Delivery Truck (UART connection) is working.

Taking Orders:

A waiter (D-Bus) takes an order from a customer (sink hardware).

The order is passed to the appropriate chef (config, data, or otap module).

Cooking the Food:

The Head Chef (config) sets up the kitchen.

The Sous Chef (data) prepares the main course (data messages).

The Pastry Chef (otap) bakes desserts (firmware updates).

Delivering the Food:

The Delivery Truck (Wirepas API) takes the finished dishes to the customer.

The Gossiping Waiter (logger) announces, "Order delivered!"

Closing the Restaurant:

The Manager (main.c) shuts everything down, thanks the chefs, and locks the doors.

### **Summary**

The sink service is like a well-run restaurant:

Head Chef (config): Manages the kitchen setup.

Sous Chef (data): Handles the main orders.

Pastry Chef (otap): Takes care of firmware updates (desserts).

Manager (main.c): Keeps everything running smoothly.

Waiters (D-Bus): Communicate between the kitchen and customers.

Delivery Truck (Wirepas API): Moves data in and out.

Gossiping Waiter (logger): Keeps everyone informed