Display Hardware API

Version 1.4

Version	Release Date	Changes
v1.4	12.11.2024	Add CORS preflight requirement
v1.3	15.10.2024	Make uptime_seconds mandatory, removed alarm active_since, improve alarm categories
v1.2	06.09.2024	Reworked API: device-info, system-status, alarms and button-events
v1.1	15.02.2024	Add read-aloud subsystem health indicator (SystemHealth tts-problem-detected)
v1.0	14.09.2023	Initial Specification

Contents

1. Introduction

The Display Hardware API defines a standardized interface enabling access to hardware-specific functionalities for applications operating on passenger information displays. This API simplifies the process for hardware manufacturers to implement this access, while ensuring that the software deployed on the display remains independent from specific drivers, access patterns, or operating system APIs. The Display Hardware API is an HTTP REST API defined using the OpenAPI framework.

1.1. Goals

1. **Access to Hardware Functionality:** The primary goal of the Display Hardware API is to offer a structured means for software applications to interact with hardware-specific functionalities. By providing a consistent and standardized interface, the API enables developers to leverage display hardware capabilities seamlessly.
2. **Simplified Integration:** For hardware manufacturers, the API presents a straightforward integration process. The API's design minimizes complexities, reducing the effort required to incorporate hardware-specific features into the passenger information display system.
3. **Software-Hardware Decoupling:** The Display Hardware API facilitates separation between software and hardware components. This decoupling eliminates the necessity for software to be intricately linked to proprietary drivers, access methods, or operating system APIs.

1.2. Abbreviations

Abbreviation	Description
PID	Passenger Information Display
HTTP	HyperText Transfer Protocol
API	Application Programming Interface
JSON	JavaScript Object Notation
REST	Representational State Transfer
TTS	Text-to-Speech
URL	Uniform Resource Locator
UTF-8	Unicode Transformation Format 8-bit

1.3. License

The use of this document and the implementation of the specified API is permitted for commercial and non-commercial applications, for both server- and client-side.

2. General

The API design is based on OpenAPI Specification 3.1.0. In addition to this document, a formal protocol schema and description is provided in an *openapi.yaml* file.

2.1. JSON return values

All values accepted and returned by API methods must be in JSON format. JSON text content must be UTF-8 encoded.

For simplicity of implementation and further compatibility, no other formats are supported.

2.2. Status Codes

API uses HTTP status codes to convey results of client's request.

2.2.1. Success Codes

Code	Description
200 (ok)	Successfully processed the request

2.2.2. Failure Codes

Code	Description
404 (Not Found)	Invalid endpoint or object not found; Applicable for all unknown path requests
500 (Internal Server Error)	The server encountered an unexpected condition that prevented it from fulfilling the request
501 (Not Implemented)	The endpoint is not supported
503 (Service Unavailable)	The endpoint is supported, but currently not available

Note: 404 and 501 differ in meaning. 501 states device does not implement a function and when such is not defined as obligatory then it is not an error. 404 states an error.

2.3. Server response times

The server implementation and hardware must be able to handle queries at certain rates. Endpoints such as */buttons/event-counts* are expected to be queried often to provide a satisfying user experience.

endpoint	maximum response time (ms)
/system-status	200
/alarms	200
/buttons/event-counts	50
/watchdog/keepalive	50

The server must implement the abovementioned endpoints so that they can be queried concurrently. Specifically, a request to the */system-status* endpoint must not stall the request to the */buttons/event-counts* endpoint.

2.4. Network Topology and Security

The Device Hardware API by its nature shall run locally the display computer and be accessible only from within the boundaries of the local host.

Because of its simplicity and intended ease of implementation, it does not employ any additional security measures. The API server is forced to only listen on localhost and the firewall must not have that port open for external connections.

API must be served as an HTTP server and access to it shall not be secured by any security mechanism as defined in underlying technologies (OpenAPI).

2.4.1. CORS preflight request

The requests to the Display Hardware API server may be coming from a web-browser running on the same device. Some browsers enforce the use of a CORS preflight request to check if the server allows answering requests from the web-browser to localhost. For this reason, the server implementation must answer CORS preflight requests for all endpoints. The mechanism works as follows:

The client will send an HTTP `OPTIONS` request with the header `Access-Control-Request-Private-Network: true` to the requested endpoint.

```
HTTP/1.1 OPTIONS /alarms
Origin: https://example.com
Access-Control-Request-Private-Network: true
```

The server must answer with HTTP 204 (`no content`) with the header `Access-Control-Allow-Private-Network: true` and `Access-Control-Allow-Origin` set the the `Origin` of the request.

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Private-Network: true
Access-Control-Max-Age: 86400
```

Afterwards the client will send the actual GET/POST request to the endpoint. Ideally the server also sets a `Access-Control-Max-Age` header on the response to the OPTIONS request to allow the browser to cache the preflight request.

More details:

1. https://developer.chrome.com/blog/private-network-access-preflight
2. https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request

2.5. Optional Endpoints and Fields

The server side is not obliged to implement all endpoints and/or fields specified in this API. Some endpoints and fields are marked as optional. For requests to optional and unimplemented endpoints the server shall return HTTP status code 501. Please note that additional project documents (such as tender documents etc.) may require the implementation of certain or all optional endpoints and fields.

3. API endpoints

Base URL: http://127.0.0.1/display-api/v1/

3.1. GET /device-info

Retrieves general information about the device

Retrieves some general properties of the device that do not change during runtime. The following fields shall be provided.

Response fields

Name	Type	Required	Description
serial_number	string	yes	A unique identifier for this particular device. This ID **must not** be shared with any other display, even of the same model. It must be persistent across system reboots, software/firmware updates and configuration changes.
manufacturer	string	no	Name of the manufacturer of the device.
model_name	string	no	Manufacturer given model name for this kind of display.
hardware_revision	string	no	The hardware revision id or number of this device.
screen_count	integer	no	Number of screens that this device has. For example 1 for a single sided display and 2 for a double sided display.
horizontal_resolution	integer	no	The number of pixels (per screen) that this device has in the horizontal direction. For example 1920 for a full HD screen
vertical_resolution	integer	no	The number of pixels (per screen) that this device has in the vertical direction. For example 1080 for a full HD screen

The implementation of this endpoint is mandatory.

Example response (200 OK)

```
{
  "serial_number": "a21-b32-03",
  "manufacturer": "display-makers",
  "model_name": "dm-32-full-hd",
  "hardware_revision": "rev4",
  "screen_count": 2,
  "horizontal_resolution": 1920,
  "vertical_resolution": 1080,
}
```

3.2. GET /system-status

Retrieve system status information

The system status contains general information about system status and environment. The system status fields change during runtime, so this endpoint is polled periodically (about once per minute).

Response fields

Name	Type	Required	Description
uptime_seconds	integer	yes	Number of seconds that the device has been continuously running (since last restart).
screen_active	boolean	yes	Status of the screens (panels). True if the screens are active and false if they are currently disabled
internal_temperature	integer	no	Temperature within the housing of the device in degrees Celsius.

The implementation of this endpoint is mandatory.

Example response (200 OK)

```
{
  "uptime_seconds": 3645,
  "screen_active": true,
  "internal_temperature": 46,
}
```

3.3. GET /alarms

Get the currently active alarms of the system.

Returns the currently active alarms of the system. Alarms shall be reported for conditions that differ from the nominal and expected operating state of the device. An alarm has a `category` and a `description`.

Alarms are only reported while they are active. This means that for every alarm that the device reports it must have a defined condition under which the alarm becomes inactive and is hence no longer reported. The device must automatically detect when the alarm state is no longer present.

For example, if the device can detect an opening of the housing, then an alarm for an open housing shall only be reported while the housing is actually open. When the housing is closed, the alarm is no longer reported. Other examples are detection of broken panel or LED modules, which become inactive as soon as they are fixed and detected to be working again. Another example is an acoustic glass breakage sensor that is causing an alarm only for the time that the glass pane is detected to be broken and becomes inactive when the pane is replaced.

This endpoint is designed to be flexible and allow reporting multiple different kinds of alarms based on the sensors available in different display models. The implementer picks fitting categories and expressive descriptions for the detectable alarms. The valid categories are listed in the table below. The alarm descriptions must be in English and understandable by a technically versed person. No two active alarms shall have the same combination of category and description.

The returned object has a single key: `active_alarms`. The alarms are reported as an array under that key. Each individual alarm has the following attributes

Name	Type	Required	Description
category	string	yes	Category of the alarm. Must be one of `video-output`, `audio-output`, `temperature`, `water`, `damage`, `physical-security`, `power`, `fan`, `heater`, `network`, `self-diagnostics`, `software`, `other`
description	string	yes	Description of the alarm and its cause

The following table contains examples of useful alarms (non-exaustive):

Category	Description
video-output	Panel of screen 2 not connected
video-output	Broken LEDs detected
audio-output	TTS speaker not connected
temperature	Internal operating temperature exceeded
temperature	Panel temperature high
temperature	CPU temperature high
water	High humidity detected within housing
water	Water detected in housing
damage	Glass pane of screen 1 broken
physical-security	Housing door 1 is open
power	Power fluctuation detected
power	Spare power supply failure
power	Battery low
power	Low power supply voltage
fan	Fan not spinning
heater	Heater not working
network	Network connection unstable
network	Low signal strength
self-diagnostics	Out of storage space
self-diagnostics	No communication with diagnostics controller
self-diagnostics	Storage memory wear alarm (S.M.A.R.T.)
software	Repeatedly crashing application
other	Maintenance inspection overdue

The implementation of this endpoint is mandatory.

Example response (200 OK)

```
{
  "active_alarms": [
    {
      "category": "housing",
      "description": "Door 1 is open",
      "active_since": "2024-09-06T14:17:25Z"
    }
  ]
}
```

3.4. GET /buttons/event-counts

Get number of times that the different button events have occured since startup.

Returns an object containing the number of times that the different button events have occured since the start of the system.

Currently there are two event kinds that are supported: `tts-short-press` and `tts-long-press`. The short press event occurs if the user presses the TTS button and releases it after less than 2 seconds. If the user holds the button for more than two seconds, a tts long press event is counted instead. In contrast to the short press event, the long press event is registered directly after passing the 2 second threshold, even if the user hasn't yet released the button.

Response fields

Name	Type	Required	Description
tts_short_press	integer	no	Number of times that the tts button was pressed down for less than 2 seconds (since start of the server).
tts_long_press	integer	no	Number of times that the tts button was pressed down for at least 2 seconds (since start of the server).

The server is required to be able to handle at least 10 requests per second to this endpoint without causing significant system load.

Example response (200 OK)

```
{
  "tts_short_press": 2,
  "tts_long_press": 3,
}
```

3.5. POST /system-reset

Request full device restart.

A system reset may be requested in order to recover from certain error states. It is supposed to reset the hardware and software state of the display, ideally through performing a full power-cycle of the whole appliance. After the reset, the system must boot again automatically, restarting all software. This request shall not wipe persistent data or configuration files. At minimum, requests to this endpoint must result in rebooting the operating system, but projects are likely to require a more rigorous reset implementation.

The implementation of this endpoint is mandatory.

3.6. POST /screens/state

Request a screens state change.

Sets the state of the screens of the PID. Currently only contains the field active. When active is set to false, the server shall turn off or disable the screens. The exact method of disabling the screens is not prescribed, but the intent is to put the screens in a state where they do not display content and appear to be off. If technically possible, the screens shall also consume less power in the inactive/disabled state. For example, for LED screens the implementation may turn off all LEDs, for TFT screens the implementation may turn off the backlight and output black. The screen shall remain in this state until another /screens/state request changes the state.

The function is not obligatory.

Request fields

Name	Type	Required	Description
active	bool	yes	True for activating screens, false for disabling them.

Example request body

```
{
  "active": true
}
```

3.7. GET /watchdog/config

Returns the watchdog configuration parameters, which are needed for the client to determine a reasonable keepalive pace. See keepalive endpoint for more details.

The purpose of the watchdog is to allow the system to recover automatically from unforseen error cases and application crashes. The implementation of the watchdog functionality is mandatory. It is recommended that the configured timeout is not shorter than 60 seconds.

Response fields

Name	Type	Required	Description
timeout_seconds	integer	yes	Interval in seconds of keepalive messages expected by the server from the client application. The client must call the */watchdog/keepalive* endpoint at least every *timeout* seconds. If the client application fails to do so, the system service will reset the device.
enabled	bool	yes	True if the watchdog is enabled, false otherwise. The watchdog shall only be disabled for testing purposes. On productively deployed displays, the watchdog shall always be active.

The implementation of this endpoint is mandatory.

Example response (200 OK)

```
{
  "timeout_seconds": 120,
  "enabled": true,
}
```

3.8. POST /watchdog/keepalive

Calms the watchdog

Regular requests to the keepalive endpoint are expected by the server to assure the system is in a good state. If the server does not receive POST request to /watchdog/keepalive for longer than the configured timeout, it performas a system reset (as by POST /system-reset).

Implementers note: The watchdog server should consider having a longer timeout for the first keepalive after system startup, since it will take some additional time for the pinging application to start and be available.

The implementation of this endpoint is mandatory.