Display Gateway API

v1.2

Version	Release Date		Changes
1.2	14.09.2023		Reworked and extended DisplayProperties
1.1	20.06.2023		Added additional flags in DisplayProperties for diagnostics
1.0	16.06.2023		Initial specification

Contents

Introduction

This document describes the DisplayGateway API. It is based on gRPC, an open remote procedure call framework.

Some types of display hardware require custom proprietary communications protocols to the backend. This is especially the case for battery operated displays that need to conserve energy and may not run a full operating system. These protocols often make use of custom hardware features and are tightly integrated between manufacturer display hardware and their communications backend. For this reason, these protocols do not serve as a reasonable integration point between different systems.

Instead, this document proposes a display gateway API. This API is to be implemented by manufacturers that require proprietary communication protocols to their display hardware. The manufacturer provides a gateway server implementing this API, allowing other systems to provide data and settings to the displays through the gateway. The API is designed so that the systems are fully decoupled: The display manufacturer does not need any details about the data providing system which in turn does not need any details of the display manufacturer’s system.

This document uses inline code blocks for describing and documenting the message formats in protocol buffers syntax. The documentation of message fields is inside those protocol buffers blocks. This allows this document to be fully synchronized with the separate full proto file.

License

Service Methods

The display gateway server shall implement the following gRPC service:

```
// The DisplayGateway service defines the API that the display gateway server must provide. The
// system delivering the data will use this API to provide data and settings to the display gateway
service DisplayGateway {

    // Returns the API version implemented by the gateway. This function is for future backwards
    // compatibility for clients of this API.
    rpc GetApiVersion(google.protobuf.Empty) returns (ApiVersion);

    // Returns the list of available layouts.
    rpc GetAvailableLayouts(google.protobuf.Empty) returns (AvailableLayouts);

    // Sets the TTS dictionary used by the gateways and displays TTS system.
    rpc SetTtsDictionary(TtsDictionary) returns (google.protobuf.Empty);

    // Returns the list of all displays known to the gateway and their properties.
    rpc GetDisplayList(google.protobuf.Empty) returns (DisplayList);

    // Sets the list of departures that currently should be visible on the given display.
    // The given list is exhaustive and overwrites any prior departures.
    rpc SetDisplayDepartures(DisplayDepartures) returns (google.protobuf.Empty);

    // Sets the list of special texts that are currently active for this display.
    // The given list is exhaustive and overwrites any prior special texts.
    rpc SetDisplaySpecialTexts(DisplaySpecialTexts) returns (google.protobuf.Empty);

    // Sets the settings of a display.
    rpc SetDisplaySettings(DisplaySettings) returns (google.protobuf.Empty);

    // Returns a PNG encoded image of the content that is on the display. For conserving energy
    // the gateway is allowed to provide a reasonable server side rendering of the content,
    // instead of getting an actual screenshot from the display itself.
    rpc GetDisplayContent(GetDisplayContentRequest) returns (DisplayContent);
}
```

The details of the methods and their used message structures are documented in the following sections.

GetApiVersion

rpc GetApiVersion(google.protobuf.Empty) returns (ApiVersion)

Returns the API version implemented by the gateway. This function is for future backwards compatibility for clients of this API. This document describes version 1.2 of the API, hence servers implementing it shall answer requests to GetApiVersion with major set to 1 and minor set to 2.

```
message ApiVersion {
    uint32 major = 1;
    uint32 minor = 2;
}
```

GetAvailableLayouts

rpc GetAvailableLayouts(google.protobuf.Empty) returns (AvailableLayouts)

Returns the list of available layouts. These are the layouts implemented by the gateway and the connected displays respectively. The layouts shall contain a textual description of the layout that helps a person operating the system choose a layout for a display.

```
message AvailableLayouts {
    repeated LayoutDescription layouts = 1;
}

message LayoutDescription {
    // ID of the layout
    string id = 1;

    // Name of the layout
    string name = 2;

    // A text describing the layout to operators of the system.
    optional string description = 3;

    // A list of compatible display models. This layout can only be set on displays with a matching
    // model number in their DisplayProperties
    repeated string compatible_model_numbers = 4;
}
```

SetTtsDictionary

rpc SetTtsDictionary(TtsDictionary) returns (google.protobuf.Empty)

Sets the TTS dictionary used by the TTS system of gateway and displays.

```
message TtsDictionary {
    // The TTS dictionary is a list of replacements that are to be applied when generating the
    // text that is used for audio generation. The replacements are required to be applied in the
    // order that they appear in this list.
    repeated TtsReplacement replacements = 1;
}

message TtsReplacement {
    enum ReplacementKind {
        // Literal one to one text replacement, for example 'st.' -> 'street'
        LITERAL = 0;

        // The search_text contains a literal string, but the replacement is a phonetic text
        // using the X-SAMPA syntax. Example 'zoo' -> 'zu:'
        X_SAMPA = 1;

        // The search_text is a regular expression using PCRE2/Javascript syntax. The search_text
        // may contain capture groups and the replacement may use those captures. Example:
        // 'S([0-9]+)' -> 'S-Bahn $1'
        REGEX = 2;
    };

    // The kind of replacement to be applied.
    ReplacementKind kind = 1;

    // The text searched in the TTS text. All instances of the search_text must be replaced.
    string search_text = 2;

    // The text that the search_text is to be replaced with.
    string replacement = 3;
}
```

GetDisplayList

rpc GetDisplayList(google.protobuf.Empty) returns (DisplayList)

Returns the list of all displays known to the gateway and their properties.

```
message DisplayList {
    // List of all displays known to the gateway and their properties.
    repeated DisplayProperties displays = 1;
}


message DisplayProperties {
    // The unique ID of the display of which these are the properties. ID must be unique over
    // all displays. It may for example be a serial number.
    string id = 1;

    // Name of the display.
    string name = 2;

    // Manufacturer model number of the display. Note that this is not a serial number. Displays
    // with equivalent hardware should have the same model number.
    string model_number = 3;

    // Time since when the display has been connected to the gateway. Leave unset if the display is
    // currently not connected (offline). States of expected disconnection, for example for
    // energy conservation in between requests, still count as online.
    // Displays count as offline when they failed to contact the gateway server when the server
    // expected them to.
    optional google.protobuf.Timestamp connected_since = 4;

    // Timestamp denoting the last time that the housing of the display was opened. Leave unset
    // if the display does not have a sensor for detecting opening of the housing or if it has
    // never been opened yet.
    optional google.protobuf.Timestamp housing_last_opened = 5;
    // Timestamp denoting the last time that the housing of the display was closed. Leave unset
    // if the display does not have a sensor for detecting opening of the housing or if it has
    // never been closed yet.
    optional google.protobuf.Timestamp housing_last_closed = 6;

    // Charge level of the battery from 0.0 to 1.0 (0.5 means half-filled battery). Leave unset
    // if display doesn't have a battery.
    optional float battery_level = 7;

    // Current signal strength of the wireless connection in dBm (decibel-milliwatts). Leave unset
    // if display uses a wired connection or if the display does not have a way to measure the
    // signal strength.
    optional sint32 signal_strength = 8;

    // Flag indicating water, moisture or unsafe levels of humidity within the housing, detected
    // by a sensor in the device. Set to false if sensor shows safe operating conditions. Leave
    // unset if device does not have such a sensor.
    optional bool water_in_housing = 9;

    // Flag indicating breakage of the glass protecting the panels. This flag is used for quick
    // detection of vandalism and fast repairs. For proper detection of glass breakage, a dedicated
    // ultrasonic or optic sensor is usually needed per installed glass panel. Set to false if the
    // device has a glass breakage sensor but if no glass break is detected at the current time.
    // Leave unset if the device does not have this kind of sensor.
    optional bool glass_breakage = 10;

    // Flag indicating that the device is overheating. The condition of overheating is met when the
    // temperatures within the housing are so high that the lifespan of the device is reduced when
    // continuing to operate in that environment or when the device has to reduce functionality to
    // prevent damage. The exact temperature when overheating is reached depends on the used
    // hardware and its safe operating temperature range. Set to false if the device has a
    // temperature sensor that is indicating safe operating conditions. Leave unset if the device
    // does not have such a sensor.
    optional bool overheating = 11;

    // Flag indicating that there is an issue with the panel or the panel controller. This could
    // either be an outright failure to display anything or a detected problem of the panel or the
    // panel controller. Set to false if panel operates as expected. Leave unset if the display is
    // unable to detect panel issues. If the device has multiple panels set this to true if any of
    // the panels have an issue.
    optional bool panel_issue = 12;

    // Temperature in degree Celsius measured inside the housing. Leave unset if device does not
    // have a temperature sensor within the housing.
    optional int32 temperature_in_housing = 13;

    // Relative humidity measured within the housing, on a scale of 0 to 1. Leave unset if device
    // does not have a sensor for relative humidity.
    optional float humidity_in_housing = 14;

    // Temperature in degree Celsius of the panels. Temperature of the first panel is the first
    // entry, temperature of the second panel the second and so on. Leave list empty if device
    // does not have temperature sensors for the panels.
    repeated int32 panel_temperatures = 15;

    // Flag indicating if the heating element is active. Leave unset if the device does not have
    // a heating element.
    optional bool heating_active = 16;

    // Text containing version information about the software running on the device. May contain
    // information about multiple modules and services. In that case the string shall contain
    // one row per module, each row containing the name and the version of the module.
    optional string version_info = 17;

    // Time of the last time data was exchanged between gateway and the device. Leave unset if no
    // data was ever exchanged.
    optional google.protobuf.Timestamp last_data_exchange = 18;

    // Total number of bytes sent from the device to the gateway. Leave unset if device cannot
    // collect statistics on sent bytes.
    optional uint64 total_bytes_sent = 19;

    // Total number of bytes the device received from the gateway. Leave unset if device cannot
    // collect statistics on received bytes.
    optional uint64 total_bytes_received = 20;

    // IP address of the device, stored as formatted IPv4 or IPv6 address. Leave unset if the
    // address of the device cannot be determined.
    optional string ip_address = 21;
}
```

SetDisplayDepartures

rpc SetDisplayDepartures(DisplayDepartures) returns (google.protobuf.Empty)

Sets the list of departures that currently should be visible on the given display. The given list is exhaustive and overwrites any prior departures.

```
message DisplayDepartures {
    // The ID of the display of which the departures are set.
    string display_id = 1;

    // Full list of all departures that shall now be shown on the display.
    repeated Departure departures = 2;
}

message Departure {
    // Destination text of the departure, for example "Main Station"
    string destination_text = 1;

    // Line name, for example "M12"
    string line_name = 2;

    // Name of the platform from which the vehicle will depart. Useful for displays that show
    // departures for multiple platforms.
    string platform_name = 3;

    // ID of the operator serving this departure. Can be used to display an icon of the operator.
    string operator_id = 4;

    // Type of vehicle serving this departure. Can be used to show an icon indicating the serving
    // vehicle type, such as tram or bus. The exact possible values of this attribute need to be
    // agreed upon with the system delivering the data.
    string vehicle_type = 5;

    // List of via texts that can be shown for this departure. The order of this list denotes the
    // order in which the texts must be shown on the display. When there is not enough space to
    // show all texts, the system must drop the ones with the lowest priority first.
    repeated ViaText via_texts = 6;
    // Text explaining to passengers why this departure was cancelled. Only filled when the flag
    // is_cancelled is true. Note that even when is_cancelled is true, this field may be empty.
    string cancellation_reason = 7;

    // The scheduled arrival time of the vehicle at the stop, encoded as POSIX time. A value of 0
    // means that the field is unset.
    int64 scheduled_arrival_time = 20;

    // The scheduled departure time of the vehicle from the stop, encoded as POSIX time. A value of
    // 0 means that the field is unset.
    int64 scheduled_departure_time = 21;

    // The actual arrival time at the stop predicted by the ITCS, encoded as POSIX time. A value of
    // 0 means that the field is unset.
    int64 actual_arrival_time = 22;

    // The actual departure time at the stop predicted by the ITCS, encoded as POSIX time. A value
    // of 0 means that the field is unset.
    int64 actual_departure_time = 23;

    // Flag denoting that the vehicle is currently at the stop.
    bool at_stop = 30;

    // Flag denoting that this departure was cancelled.
    bool is_cancelled = 31;

    // Flag denoting that the vehicle serving this departure is currently in a traffic jam. The
    // given predictions for arrival and departure may be inaccurate in this case.
    bool traffic_jam = 32;

    // Flag denoting that this stop of the vehicle is only for letting passengers exit. Entering
    // the vehicle is not allowed.
    bool no_entry = 33;

    // Flag denoting that this vehicle is accessible by wheelchair.
    bool wheelchair_accessible = 34;
}

message ViaText {
    // Passenger facing text
    string text = 1;

    // Priority for showing this text if not all texts fit in allocated space
    uint32 priority = 2;
}
```

SetDisplaySpecialTexts

rpc SetDisplaySpecialTexts(DisplaySpecialTexts) returns (google.protobuf.Empty)

Sets the list of special texts that are currently active for this display. The given list is exhaustive and overwrites any prior special texts.

```
message DisplaySpecialTexts {
    // The ID of the display to which the special texts are addressed.
    string display_id = 1;
    // Full list of all special texts that shall now be visible on the display.
    repeated SpecialText texts = 2;
}

message SpecialText {
    enum TextKind {
        INFO = 0;
        ALERT = 1;
        CRISIS = 2;
    };

    // The kind of the special text. Diffent kinds of special texts may be shown differently. For
    // example, a layout may show an INFO kind text as a row of text under the departures, but a
    // ALERT text as a fullscreen overlay hiding all departures. Furthermore, the kind may also
    // influence the TTS behavior. In the mentioned example, ths TTS may no longer read the
    // departures when an ALERT kind text is active.
    TextKind kind = 1;

    // Text to be displayed
    string text = 2;
}
```

SetDisplaySettings

rpc SetDisplaySettings(DisplaySettings) returns (google.protobuf.Empty)

Sets the settings of a display.

```
message DisplaySettings {
    // The ID of the display to which the following settings are addressed.
    string display_id = 1;

    // The ID of the layout set for the display.
    string layout_id = 2;

    // The name of the stop that this display serves. Certain layouts may show this text. Moreover
    // the TTS system may announce this text as part of the generated audio, e.g. "You are at
    // {stop_name}. These are the next departures: ..."
    string stop_name = 3;

    // The geographical coordinates of the stop. This attribute allows showing the position of the
    // display on a map for layouts that contain a map view.
    optional LatLng geo_pos = 4;
}
```

GetDisplayContent

rpc GetDisplayContent(GetDisplayContentRequest) returns (DisplayContent)

Returns a PNG encoded image of the content that is on the display. For conserving energy the gateway is allowed to provide a reasonable server side rendering of the content, instead of getting an actual screenshot from the display itself.

```
message GetDisplayContentRequest {
    // The ID of the display for which a content image is requested.
    string display_id = 1;
}

message DisplayContent {
    // A PNG encoded image of the content that is shown on the display. The image can be a server
    // side rendering of the content; it does not have to be a fetch from the actual display.
    bytes content_image = 1;
}
```