Dashboard Skin Tutorial

For ETS2 HTML5 Mobile Dashboard
v2.2.0

Index

- Dashboard engine overview
- Dashboard menu
- Skin file structure
  - config.json
  - dashboard.html
  - dashboard.css
  - Telemetry data types
    - Booleans
    - Strings
    - Numbers and Meters
  - dashboard.js
- How to create a new skin from a template
- Testing your skin
- Publishing your skin

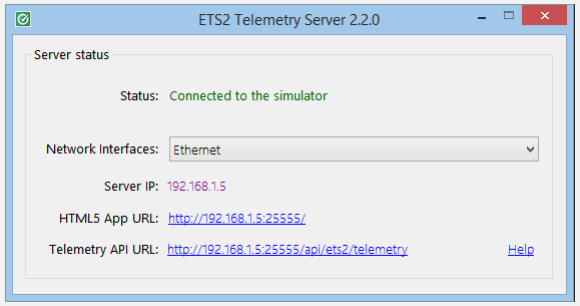
Dashboard engine overview

The telemetry server consists of 3 different parts, connected in the following way:

ETS2 Game

ets2-telemetry-server.dll

Memory Mapped File



ETS2 Telemetry Server

HTTP

+ WebSockets

HTML5 Dashboard Application

index.html

⇓

dashboard-host.html

↓

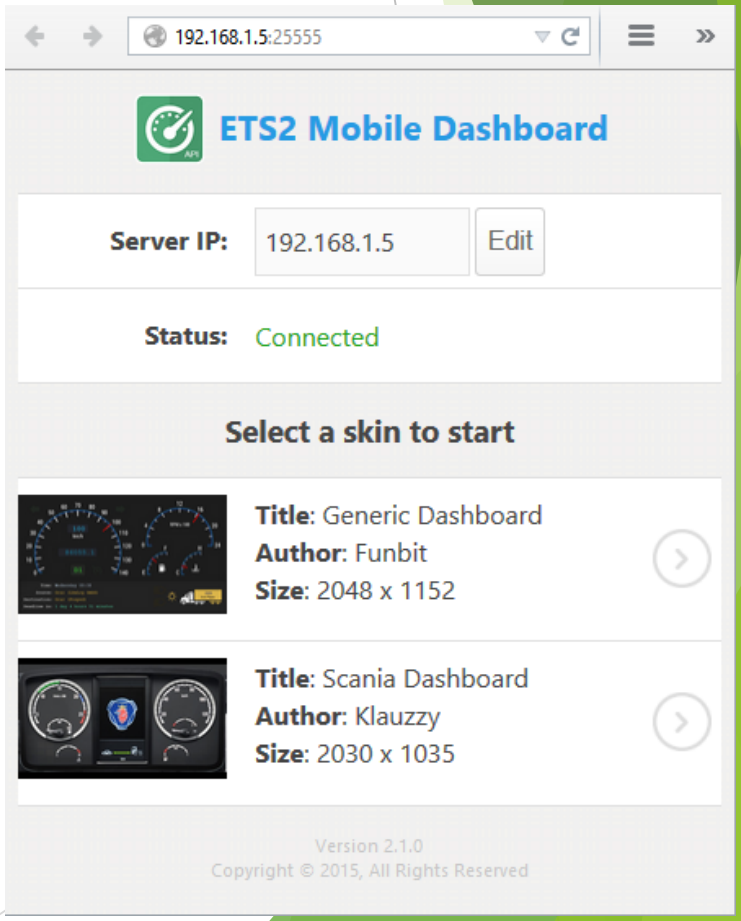
dashboard.html

Dashboard menu

When you open the dashboard by clicking on the “HTML5 App URL” link displayed by the server, the first thing you see is the main menu handled by the index.html.

When dashboard connects to the server, the server enumerates all skins inside "html/skins/*" path and loads their config.json configurations to display the list.

When you select a skin to load, the browser navigates to the dashboard-host.html file that dynamically loads files for the selected skin (html, css, etc.).



Skin file structure

Each skin must be located inside the "html/skins" directory with the name equal to the "name" property defined in the config.json file.

For example, if you want to name your skin "volvo", you should create a directory "volvo" inside the "skins": "html/skins/volvo".

All skins must have at least 5 files:

config.json – defines skin configuration

dashboard.html – defines all dashboard elements supported by the skin

dashboard.css – defines dashboard size and visual style for the dashboard elements

dashboard.jpg – defines dashboard splash screen for the menu (recommended size is 480x275px)

dashboard.js – defines custom dashboard rendering and telemetry property filtering

config.json overview

Every skin must have the following properties defined inside config.json:

- name: name of the skin. This value must be equal to the name of the sub-directory inside "html/skins". For example, if your skin is named "volvo", it must be placed inside "html/skins/volvo".
- title: human readable skin title that will be displayed in the top skin menu
- author: author name
- refreshRate: default telemetry data refresh rate (in milliseconds). Lower values may decrease visible latency but may also lead to UI stuttering due to performance problems (especially on Androids or slow networks). Recommended values are between 50ms and 250ms. The rate may be adjusted automatically at runtime by the dashboard core if connection is unstable.
- width: width of the skin in pixels
- height: height of the skin in pixels

dashboard.html overview

The dashboard.html file defines HTML DOM for all dashboard elements.

The minimal dashboard.html for a skin is a div with the "dashboard" class:

```
<div class="dashboard"></div>
```

The dashboard core will automatically fit this div tag into browser's viewport (keeping aspect ratio) using absolute positioning. The width and height in pixels will equal to the values defined inside skin's config.json file.

You have full freedom to choose what HTML tags will be used inside that div for each dashboard element (img, div, span, etc.) and what hierarchy they will have.

For example, you can create a skin that displays only current truck speed and nothing else.

The DOM would look like this:

```
<div class="dashboard">
    <div class="truckSpeed"></div>
</div>
```

dashboard.css overview

The dashboard.css file defines CSS styles for dashboard HTML elements.

There are absolutely no restrictions on the dashboard.css file structure. It is totally up to you and depends on the elements that you use in your skin.

For example, if you have a truckSpeed dashboard element defined in your HTML (i.e. a tag with "truckSpeed" class), you probably want to have CSS style for that element as well. It could be something like this:

```
.truckSpeed {
   color: #2491b9;
   font-weight: bold;
   font-size: 70px;
   position: absolute;
   left: 460px;
   top: 280px;
   width: 190px;
   height: 66px;
}
```

Coordinates for the absolutely positioned elements are relative to the skin size defined in the config.json

Telemetry data types

The dashboard core uses CSS class names to apply telemetry data to the HTML elements.

Telemetry JSON has three different data types:

- boolean (for properties like blinkerLeftOn, trailerAttached, connected, etc.)
- string (for properties like jobDeadlineTime, sourceCity, destinationCompany, etc.)
- number (for properties like truckSpeed, coordinateX, gear, trailerMass, truckSpeed, etc.)

Each data type is rendered in its own way. See next slides for more information.

Hint: To see all available properties you may click on the “Telemetry API URL” link displayed by the server.

Dashboard rendering: booleans

boolean (for properties like blinkerLeftOn, trailerAttached, connected, etc.)

When value is true the dashboard core adds "yes" CSS class to the respective element found by its class name. When value is false the "yes" class is removed.

For example, when "trailerAttached" property is true, the core searches for all HTML tags having "trailerAttached" class and adds additional "yes" class to them, so the element will look like this:

```
<div class="trailerAttached yes"></div>
```

And when the value is false the "yes" class disappears:

```
<div class="trailerAttached"></div
```

Because class names may not be unique, you may have multiple elements displaying same information in a different way (controlled by the dashboard.css)

Dashboard rendering: strings

string (for properties like jobDeadlineTime, sourceCity, destinationCompany, etc.)

For all strings the dashboard core will update HTML content of the respective elements. For example, if you have a sourceCity element defined in your dashboard.html:

```
<div class="sourceCity"></div>
```

And the current source city value is "Graz" the dashboard core will set the element's content as follows:

```
<div class="sourceCity">Graz</div>
```

Dashboard rendering: numbers (strings)

number (for properties like truckSpeed, coordinateX, gear, trailerMass, truckSpeed, etc.)

There are two ways to display numbers: as a string value or as an analog meter.

By default, all numbers are displayed just like strings, the dashboard core will update HTML content of the respective HTML elements. For example, if you have a trailerMass element defined in your dashboard.html:

```
<div class="trailerMass"></div>
```

And the current trailer mass value equals to 14000 the dashboard core will set the element's content as follows:

```
<div class="trailerMass">14000</div>
```

Hint: if you like to convert mass from kilograms to tons,

or kmh to mph see dashboard.js section for more information.

Dashboard rendering: numbers (meters)

If you like to render a number as an analog meter instead of a string you should set data-type="meter" attribute to an HTML element. The core will use that HTML element as a meter arrow and will rotate it depending on the value.

All meters must also have the following HTML data attributes defined:

- data-min: minimal possible value (as in JSON response), you may also use any telemetry property name for dynamic values
- data-max: maximum possible value (as in JSON response), you may also use any telemetry property name for dynamic values ("fuelCapacity" for example)
- data-min-angle: angle in degrees for the arrow for data-min value (0 = vertical, negative = left, positive = right)
- data-max-angle: angle in degrees for the arrow for data-max value (0 = vertical, negative = left, positive = right)

For example, in the default skin truck speed is displayed as a meter and it is defined in the HTML as follows:

```
<div class="truckSpeed" data-type="meter"
      data-min="0" data-max="140" data-min-angle="-114" data-max-angle="+114"></div>
```

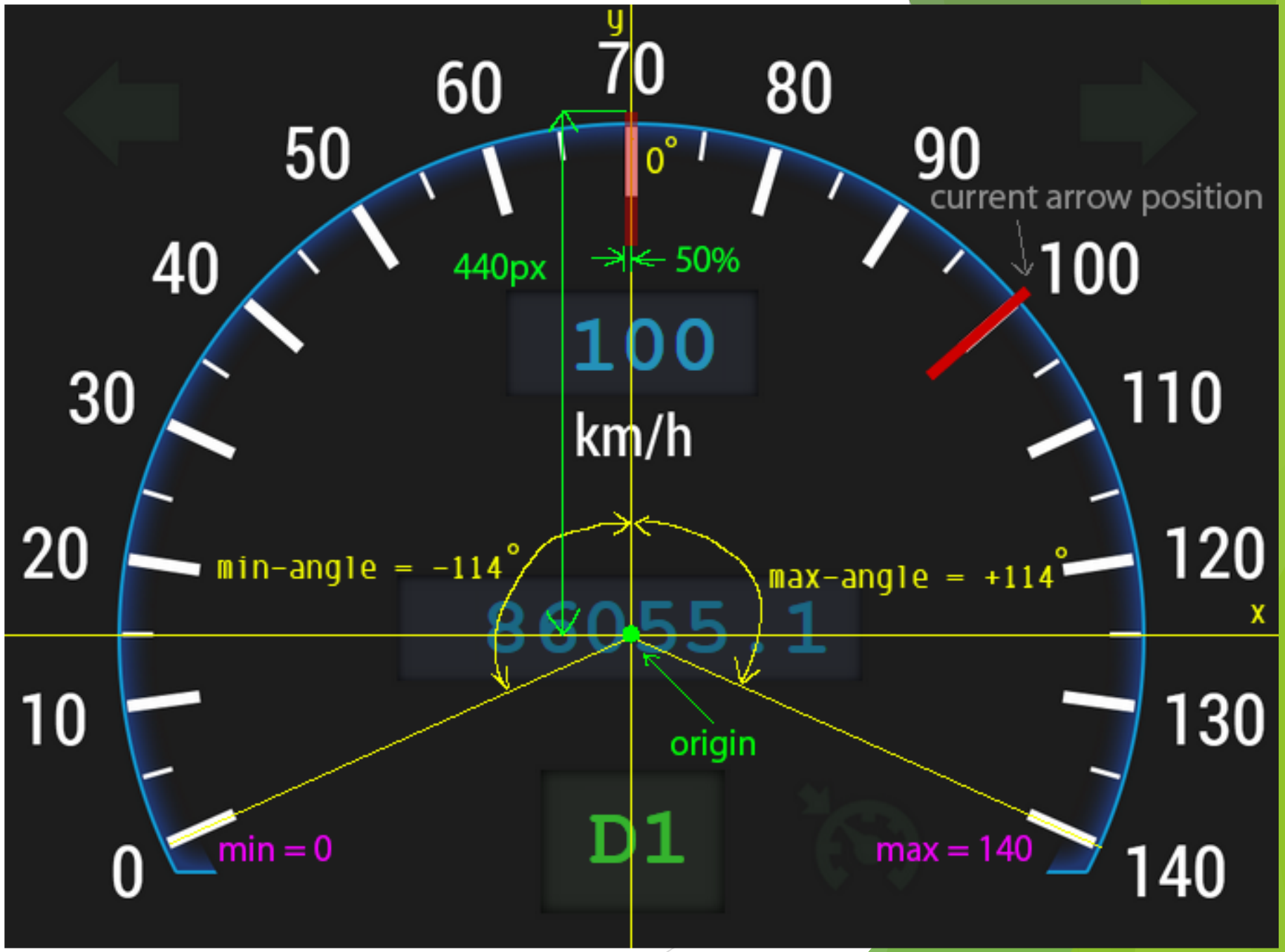
Here is how it looks in the browser:

When you define the styles for your arrow in the CSS, keep in mind that the default position must always be vertical (i.e. 0 degree rotation). The arrow will rotate around its origin between min-angle and max-angle depending on the current value. The origin must be defined in the CSS via transform-origin property. In the default skin it is defined as:

```
transform-origin: 50% 440px;
```

Which means that the origin is the point 50% to the right of the top-left corner of the arrow element (center of the small red transparent rectangle on the screenshot) and 440px down from the top. The origin is marked as a big solid green dot.

To get the correct coordinates for your own skin you should open your skin background in a graphic editor and check pixel offsets which will be relative to your skin size.

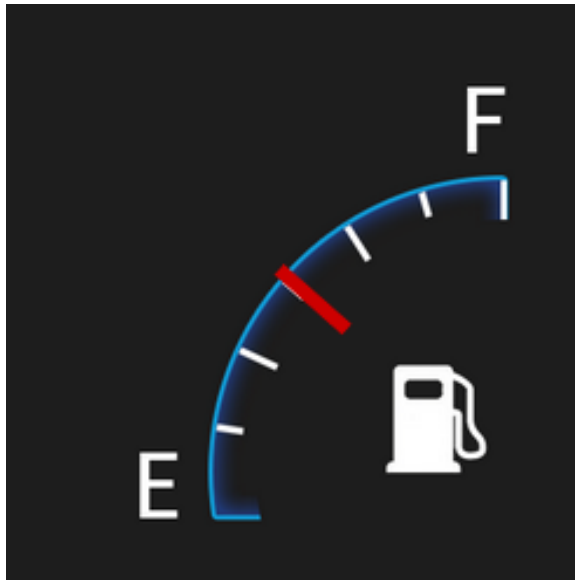


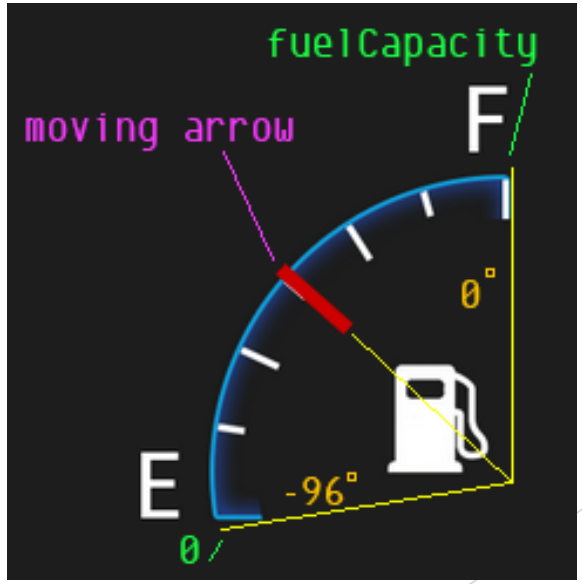
Dashboard rendering: numbers (meters)

Here is another example from the default skin. The fuel is displayed as a meter defined in the HTML as follows:

```
<div class="fuel" data-type="meter"
    data-min="0" data-max="fuelCapacity" data-min-angle="-96" data-max-angle="0"></div>
```

Here is how it looks in the browser ("min" & "max" are green, "min-angle" & "max-angle" are yellow):





Dashboard rendering: current value

When dashboard core renders an element (string, number or boolean) it always updates data-value attribute, so if you check the DOM inside your browser you will see something like this:

```
<div class="trailerMass" data-value="14000">14000</div>
<div class="sourceCity" data-value="Graz">Graz</div>
<div class="trailerAttached yes" data-value="true"></div>
```

This feature allows you to customize the element design depending on the current value.

For example, the following CSS selector may be used to set a special style for the neutral gear value:

```
.gear[data-value="N"]
{
   color: white;
}
```

Dashboard rendering: special properties

There are only 3 special (i.e. not related to the data read by the telemetry plugin) properties available for now:

- connected – boolean property updated by the telemetry server that defines whether telemetry server is connected to the simulator or not. You may use it to hide some dashboard indicators when server is not connected.
- statusMessage – human readable string property containing current dashboard core status ("Connected, waiting for the drive...", etc.). You should use it to inform user about current status. See "html/scripts/config.js" for the full list of strings.
- hasJob – boolean property updated by the telemetry server that defines whether job information is currently available (income, city names, trailer mass, etc.) or not. You may use it to hide some dashboard indicators when job information is not available.

dashboard.js overview

The dashboard.js file works like a small plugin for your skin.

It contains 3 JavaScript functions used by the dashboard core:

```
initialize = function (skinConfig) {
   …
}

filter = function (data) {
   …
}

render = function (data) {
   …
}
```

dashboard.js: initialize function

initialize = **function** (skinConfig) { … }

This function is called only once when skin is loaded.

You may use it to initialize your custom JavaScript objects, setup DOM,
preload your CSS images to speed up skin loading process, etc.

The skinConfig argument will contain skin configuration properties from the config.json file.
You may have your own properties there too (that may be used later inside dashboard.js).

dashboard.js: filter function

```
filter = function (data) { … }
```

This function is called every time when dashboard core gets fresh telemetry data from the server, but before it is rendered on the screen. The data argument contains all telemetry data properties.

You may use this function to adjust the current properties or calculate your own ones.
For example, you may convert trailerMass property from a number of kilograms to a ton string:

```
data.trailerMass = (data.trailerMass / 1000.0) + 't';
```

Or, you can introduce a new property called myTruckSpeedMph like this:

```
data.myTruckSpeedMph = data.truckSpeed * 0.621371;
```

This new property will be treated and rendered just like any standard property (i.e. dashboard core will find HTML elements having "myTruckSpeedMph" class and will apply the current value to it).

dashboard.js: render function

render = **function** (data) { … }

This function is called after dashboard core has finished its own rendering process. This place may be used for custom dashboard visual effect implementations, like animated bars, maps, animated odometers, etc. The data argument contains all available telemetry data properties.

If you plan to create your own CSS animations, it is recommended to apply the built-in "animated" CSS class to the element you wish to animate. The dashboard core will automatically set CSS transition property to that element with the proper duration. So all you have to do is to update CSS properties which you want to animate. For example, you may define a speed bar and animate its width depending on the current speed like this:

HTML:

```
<div class="truckSpeedBar animated"></div>
```

CSS:

```
.truckSpeedBar {
    left: 0px;
    top: 0px;
    width: 2048px;
    height: 10px;
    position: absolute;
    background-color: deeppink;
}
```

Render function:

```
$('.truckSpeedBar').width(data.truckSpeed * 10);
```

How to create a new skin from a template

In order to create a new skin please do the following steps:

1. Create a copy of the "html/skins/template" directory to something like "html/skins/myskin".
2. Open config.json file and set skin name to "myskin", change its title, author, size, etc.
3. Add dashboard element tags that you are going to support to the dashboard.html file
4. Add styles for these tags to the dashboard.css file
5. Edit dashboard.js if you like to implement custom data filtering or rendering (or leave it as is)
6. Create a splash dashboard.jpg image for your skin
7. Open "HTML5 App URL" and select your new skin from the menu to test it

Testing skin without ETS2

It is possible to run the server in the "test" mode when server uses JSON data from the Ets2TestTelemetry.json file instead of reading real game data. This might be very useful for skin testing, because it is very inconvenient to switch back and forth between the game and the dashboard just to check how new telemetry values are displayed on it.

To enable test mode open Ets2Telemetry.exe.config file and set UseEts2TestTelemetryData option to true and restart the server:

```
<add key="UseEts2TestTelemetryData" value="true" />
```

Please note that you may edit Ets2TestTelemetry.json file in real time (just edit and save), the dashboard will update the values and render them right away.

Publishing your own skin

If you have created a cool skin you may send it to me and I will include it in the official "ETS2 Dashboard Server" distribution published on GitHub.

All you have to do is to pack your skin in a ZIP file and send it to email: funbit @ gmail.com (if you haven't received any reply within a couple of days then most probably your email went to SPAM. You may contact me on the official SCS forum in this case).

Before sending a skin, please make sure that it satisfies the following conditions:

- the size of the skin ZIP is less than 2MB (use JPG images for photos and PNG for flat pictures)
- the skin has all 5 files (config.json, dashboard.html, css, jpg and js)
- the skin has a fresh screenshot for the menu (dashboard.jpg must be around 480x275px)

Happy driving! ☺