webfleet

PRO 8 Driver Terminal

Developer documentation

BRIDGESTONE
Solutions for your journey

1602849948866 - 1.3 - 2025-07-24

Contents

Welcome

The PRO 8 Platform is a solution to help your business move more efficiently. It is a platform to enable you to seamlessly connect your business applications with solutions from Webfleet Solutions. Now, you can easily create your own customised business applications that harness the power of TomTom's award-winning navigation technology and fleet management features from Webfleet Solutions.

The PRO 8 Developer documentation provides the growing base of developers a wealth of information and documentation.

When using the term **Driver Terminal** or **PRO 8** in this document, the following devices are referred to:

- PRO 8475
- PRO 8375
- TomTom PRO 8270
- TomTom PRO 8275

If there are any exceptions applicable to individual devices only, those are highlighted separately.

Develop

PRO 8 NavApp SDK

PRO 8 is Android

A PRO 8 Driver Terminal is a standard Android device, currently running Android 6.0, API level 23 (on PRO 8375 and TomTom PRO 827x) and Android 9.0, API Level 28 on PRO 8475. Minimal platform changes were done to improve the in-car experience, which means that a PRO 8 Driver Terminal remains an Android device. Any apps that you can run on other Android platforms, will run on the PRO 8 Driver Terminal as well, so you are free to embellish the PRO 8 Driver Terminal experience with APKs of your own making, or any third-party applications that your customers/users would find useful.

There is one notable exception to this: the launcher application. To provide as much power as possible to the user on their home screen, widgets are integrated directly into the default launcher. If you would like to use another launcher, be aware that the **Navigation** and **External camera** widgets will not be available to you.

The PRO 8 SDK offers seamless integration of solutions from Webfleet Solutions with any business application to help companies achieve their efficiency goals.

The SDK contains NavApp SDK and Map Library SDK which will enable you to integrate your business applications with TomTom's advanced navigation software. They are designed for ease of use: just add the SDKs to your application project and start navigating from within your own app. The SDKs are initialised through a single API call as well as most of the navigation API functions.

Downloading the NavApp SDK

You can download the **NavApp SDK version 9** from the **Downloads** section of the Downloads portal.

NavApp SDK javadoc documentation

NavApp SDK documentation - interfacing to navigation.

Installing the SDK

Copy the navappclient.jar file into your application project libs directory. The file can be found in the SDK which you downloaded from the portal.

Initialising the SDK

The NavApp SDK consists of a group of interfaces accessed through the NavAppClient class.

To use the NavApp SDK you need to create an instance of the NavAppClient, using the `NavAppClient.Factory.make(Context, ErrorCallback)` method.

The returned instance is used to access the individual APIs.

As instantiating the NavAppClient requires some background initialisation, the NavAppClient is preferably created once per application, e.g. in the Activity `onCreate` callback.

Once this call returns the NavAppClient instance is ready to use.

```
protected void onCreate(final Bundle savedInstanceState) {
     // Instantiate the NavAppClient passing in a Context.
     mNavappClient = NavAppClient.Factory.make(this, mErrorCallback);
}

private final ErrorCallback mErrorCallback = new ErrorCallback() {
    @Override
    public void onError(final NavAppError error) {
        Log.e(TAG, "onError(" + error.getErrorMessage() + ")\n" + er-
ror.getStackTraceString());
        mNavappClient = null;
    }
};
```

And should be closed in the activity `onDestroy` callback, using `NavAppClient.close()`

```
protected void onDestroy() {
    mNavappClient.close();
}
```

Using the API

The NavAppClient instance can be used to access the APIs.

Retrieving information about the map

Note: Any callback from the SDK will be done on the UI Thread. All APIs will throw an `Il-legalArgumentException` if the listener argument is `null`.

```
mSDKUtils = mNavappClient.getUtils();
mSDKUtils.getMapInfo(mMapInfoListener);

private MapInfo.Listener mMapInfoListener = new MapInfo.Listener() {
    public void onMapInfo(final MapInfo mapInfo) {
         Log.d(TAG, "name: " + mapInfo.getName() +
             "releaseNumber: " + mapInfo.getReleaseNumber() +
             "releaseDate: " + mapInfo.getReleaseDate() +
             "buildNumber: " + mapInfo.getBuildNumber() +
             "locationPath: " + mapInfo.getLocationPath());
 }
};
```

Planning a trip

```
mTripManager = mNavappClient.getTripManager();

final Routeable destination = mNavappClient.makeRouteable(DESTINATION_LATI-
TUDE, DESTINATION_LONGITUDE);
mTripManager.planTrip(destination, mPlanListener);

private Trip.PlanListener mPlanListener = new Trip.PlanListener() {
    public void onTripPlanResult(final PlanResult result) {
         Log.d(TAG, "onTripPlanResult result[" + result + "]");
    }
};
```

Importing and exporting routes

Importing routes into Navigation

The Navigation application handles the intent `ACTION_SEND and ACTION_SEND_MULTIPLE` with mime types `application/gpx` and `application/itn`. This is useful if you want to import routes directly into the Navigation application from a file explorer. However, if you want to import routes from your application into Navigation, you can do it one at a time, using the following intent.

For single .gpx or .itn file: `tomtom.intent.action.IMPORT_SINGLE_ROUTE`

The mime types are `application/gpx` and `application/itn`, for .gpx and .itn files respectively. The following is an example to import .gpx routes to Navigation from an external application.

Importing a single route to Navigation

```
private void importSingleRouteToNavigation(final File importfile) throws
 IOException {
        final Uri fileUri = Uri.fromFile(importfile);
        final Intent importToNavigationIntent = new Intent();
        importToNavigationIntent.setAction("tomtom.intent.action.IM-
PORT_SINGLE_ROUTE");
        importToNavigationIntent.putExtra(Intent.EXTRA_STREAM,
 fileUri).setType("application/gpx").setFlags(Intent.FLAG_ACTIVI-
TY_NEW_TASK);
        importToNavigationIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PER-
MISSION);
        startActivity(importToNavigationIntent);
    }
```

To share routes from your application's internal data files, use `FileProvider`. Routes imported into Navigation, can be found in the **My Routes** screen, in the main menu.

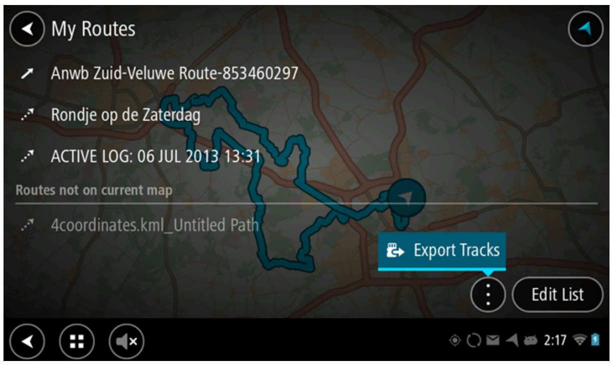
Exporting routes and saving routes to the device

All the routes (both .gpx files and .itn files appear on the **My Routes** screen. Only the .gpx Routes can be exported to another application or saved to device from the **My Routes** screen.

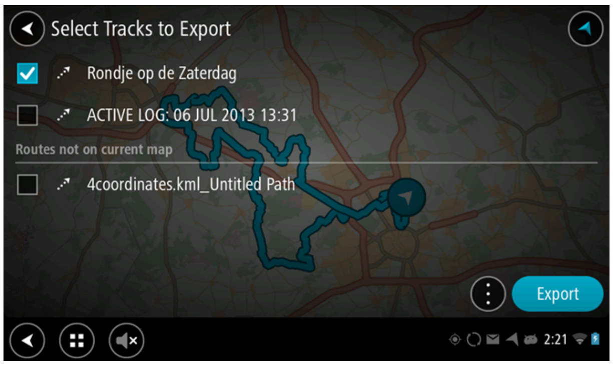
↗	Indicates the itinerary files (.itn), which cannot be exported
⋰	Indicates the route files (.gpx), which can be exported

The routes to be exported can be selected by choosing **Export Tracks** on the contextual menu on the **My Routes** screen.

Exporting .gpx routes from My Routes

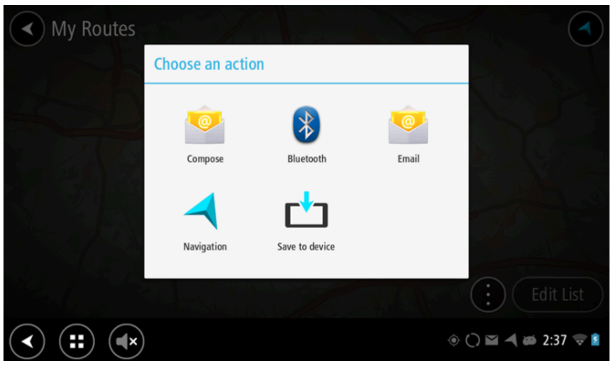

My Routes
Anwb Zuid-Veluwe Route-853460297
Rondje op de Zaterdag
ACTIVE LOG: 06 JUL 2013 13:31
Routes not on current map
4coordinates.kml_Untitled Path
Export Tracks
Edit List
2:17

Selecting routes to export


Select Tracks to Export
Rondje op de Zaterdag
ACTIVE LOG: 06 JUL 2013 13:31
Routes not on current map
4coordinates.kml_Untitled Path
Export
2:21

When a route or multiple routes are exported, you are presented with an option to send to an application that handles the route files. To save the route on the device, send the routes to **Save to device** application. This will save the selected routes in the **Routes** folder internally.

Saving routes to the device



If you want to send the route files to your own application, then your application must handle the `ACTION_SEND and ACTION_SEND_MULTIPLE` intents, with mine type `application/gpx`.

Intent filters in manifest file

```
<intent-filter>
    <action android:name="android.intent.action.SEND" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="application/gpx" />
</intent-filter>

<intent-filter>
    <action android:name="android.intent.action.SEND_MULTIPLE" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="application/gpx" />
</intent-filter>
```

The route files can then be retrieved from the intent, as shown in the example below.

Retrieving route files from intent

```
final ArrayList uriList = new ArrayList()
if (Intent.ACTION_SEND.equals(intent.getAction())) {
    final Uri uri = (Uri) intent.getExtras().get(Intent.EXTRA_STREAM);
    uriList.add(uri);
}
else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN && Inten-
t.ACTION_SEND_MULTIPLE.equals(intent.getAction())) {
  try {
    final ClipData clipData = intent.getClipData();
      if (clipData != null) {
        final Integer itemCount = (Integer) clipData.getItemCount();
        for (int i = 0; i < itemCount.intValue(); i++) {
        final Item clipDataItem = clipData.getItemAt(i);
        if (clipDataItem != null) {
          uriList.add(clipDataItem.getUri());
        }
      }
    }
  }
  catch (final IllegalArgumentException e) {
    if (Log.E) Log.e(TAG, e.getMessage());
  }
}
```

Generating input stream from file uri

```
final ParcelFileDescriptor inputPFD = mContentResolver.openFileDescrip-
tor(uri, "r");
final FileDescriptor fd = inputPFD.getFileDescriptor();
FileInputStream  inputStream = new FileInputStream(fd);
```

Recording new routes

New routes can be created in the Navigation application, by using the **Start Recording** option in your main menu. When the recording starts, you can see a red dot on your Navigation screen.

To stop the recording, go back to the main menu and select the **Stop Recording** option. Routes created this way, are listed in the **My Routes** screen in .gpx format. As such, they can be exported to another application or saved to the device.

Listening for speed limit changes

NavApp is broadcasting intent action: `tomtom.intent.action.SPEED_LIMIT_CHANGED` indicating that the speed limit has changed. An extra int `newSpeedLimitExtra` provides a new speed limit in meters per hour.

Map Library SDK

This section describes the usage of the Map Library SDK. This library allows you to visualise a map and interact with it. The Map Library SDK does not include any routing or navigation functionality like the NavApp SDK does.

Downloading Map Library SDK

You can dowload the **Map Library SDK version 11** from the **Downloads** section of this portal.

For information on how to use the SDKs, please visit the pages containing example code: Map Library SDK

Map Library SDK javadoc documentation

Map Library SDK - interfacing to a map view.

Accessing the API functionality

Below you will find some brief examples on how to achieve some of the functionality for your app.

Enabling and disabling zooming interaction

```
mMapController.setZoomEnabled(true);  // Allow zooming interaction.
mMapController.setZoomEnabled(false); // Disallow zooming interaction.
if (mMapController.isZoomEnabled()) {
    // Zooming is enabled for the user.
    ...
}
```

Zooming the map programmatically

```
final int minZoomLevel = mMapController.getMinZoomLevel();
final int maxZoomLevel = mMapController.getMaxZoomLevel();
final int zoomLevel = mMapController.getZoomLevel();
setZoomLevel(6);
```

Getting the map scale

The scale of a map is the ratio between a distance on the screen to that distance in the world. So a scale of 1 means that there is a 1:1 ratio which in turn means that a centimeter on screen is a centimeter in reality. A scale of 500 means that there is a 1:500 ratio which means that a centimeter on screen is 500 centimeters in reality. The same goes for any unit: 1:500 means 1 inch is 500 inches in reality, for example. In short, the higher the scale value, the farther zoomed out the map will look on screen.

The zoom levels indicate how far zoomed in the map is, which means the higher the zoom level, the lower the scale value. For example, zoom level 20 is defined as scale 1:500, while zoom level 13 is scaling the map to 1:64,000. The default scale value is 1:8000, which is equivalent to zoom level 16.

Finding out at which scale the map is drawn, can be done using the getMapScale() function:

```
    final int scale = mMapController.getMapScale();
```

Setting the map scale

For an explanation of what a scale value means, refer to Getting the map scale.

Setting the scale of the map is fairly simple. You call setMapScale(scale) and the map will be drawn in that scale.

```
    mMapController.setMapScale(8000);
    final int actualScale = mMapController.getMapScale();
```

The scale value must fall within a range that can be supported by both the map renderer and the map itself; a typical range could be 500 to 225.966.736. The value that you're trying to set might therefore not be the scale that will be used in the end. To verify what scale you ended up with, check getMapScale() afterwards. Values that fall outside of the supported range, will be clamped to the nearest possible value. So trying to set the scale to 100 when the map only supports 500 and higher, will scale the map to 1:500.

Enabling and disabling panning interaction

```
    mMapController.setPanEnabled(true);  // Allow panning interaction.
    mMapController.setPanEnabled(false); // Disallow panning interaction.
    if (mMapController.isPanEnabled()) { // Panning is enabled for the
 user.
        ...
    }
```

Enabling and disabling autocenter

```
    mMapController.setAutoCenterEnabled(true);  // Enables autocenter.
    mMapController.setAutoCenterEnabled(false); // Disables autocenter.
    if (mMapController.isAutoCenterEnabled()) { // Autocenter is enabled.
        ...
    }
```

Change map orientation

From version 16 (release 18.6) onward, you can determine the orientation of the map. Up until version 15, the map is always facing due North by default. Version 16 makes it possible to have the map align with the driving direction.

```
    mMapController.setMapOrientation(MapController.MAP_ORIEN-
TATION.DRIVING_DIRECTION);  // Align map to heading.
    ...
    if (mMapController.getMapOrientation() == MapController.MAP_ORIEN-
TATION.DRIVING_DIRECTION) { // Following device heading.
        ...
    }
    mMapController.setMapOrientation(MapController.MAP_ORIEN-
TATION.NORTH_UP); // Back to default behaviour.
```

Also note that with the map orientation set to DRIVING_DIRECTION it is not possible to pan the map, and disabling autocenter will be ignored. Panning is only possible while the map is oriented NORTH_UP.

Please also be aware that the options to set map orientation are only available on NDS builds. On earlier builds these calls will have no effect, or simply return default values.

Panning and setting the map center

If the requested location is near to the currently viewed location, the map will pan towards the location. If the location is too far away for smooth panning, it will instantly display the location. You can center on a Location object or use latitude and longitude directly, as well as centering the map to the current GPS location.

Getting the current map center

This will return the current location the map view has been panned to. This is not necessar-ily the current location of the device.

```
    final Location location = mMapController.getMapCenter();
```

Set map center using Location class

```
    final Location location = new Location("TomTom");
    location.setLatitude(52.3764293);
    location.setLongitude(4.908397);
    mMapController.setMapCenter(location);
```

Set map center using latitude and longitude

```
    mMapController.setMapCenter(52.3764293, 4.908397);
```

Set map center using current location

```
    mMapController.setMapCenterToCurrentLocation();
```

Auto-centering map

```
    mMapController.setAutoCenterEnabled(true); // Map will follow my cur-
rent position.
    if (mMapController.isAutoCenterEnabled()) {
        // The map is currently following my current position.
        ...
    }
```

Using map markers

Markers are locations on the map that have a visual indicator, also known as push pins. Map library provides a means to add default and custom markers to the MapView that the user can interact with. You will be notified of touch events and can act on them using the information of the selected marker, which includes world location and screen position.

Markers will pan with the map to keep indicating the location they were placed at. When the map is zoomed, marker icons will stay the same size.

Adding a default marker

To add a marker using the default marker icon, you call the `createMarkerAt` function.

```
    final Marker marker = mMapController.createMarkerAt(52.3764293,
 4.908397);
```

Removal of markers is done through this call:

```
    mMapController.removeMarker(marker);
```

Adding a marker

Markers can be grouped into layers, which can be individually hidden and shown. This allows for choosing which markers to show at which time. So in order to create a Marker you'll first need a MarkerLayer. With that layer you can create the markers that should be part of that layer.

```
    final MarkerLayer layer = mMapController.createLayer();
    final Marker marker = layer.createMarkerAt(52.3764293, 4.908397);
```

Removal of markers is done through this call:

```
    layer.removeMarker(marker);
```

Giving a marker a custom icon

If the default icon is not what you want, you have the option to give each marker a different one. The Map library can read image files from internal/external storage and also accepts `Drawable's`.

The anchor point is the point on the icon that sticks to the location, or points to where the location of the marker is. The center of the icon is at (0, 0). The right side equals 1, while the left side equals -1; similar for the bottom and top sides. So, the bottom left corner of the icon would be at (-1, 1).

```
    // Using a file path.
    final String iconPath = "/absolute/path/to/an/icon.png";
    marker.setIcon(iconPath);
    marker.setAnchorPoint(-1, 1);
    ...
    // Using a Drawable.
    marker.setIcon(getResources().getDrawable(R.drawable.marker_icon));
```

Query for markers

To obtain a list of all markers currently on the map, you call `getMarkers`. This will return a list of all(!) markers, not just the ones that are currently visible.

```
    final List<Marker> markers = mMapController.getMarkers();
```

To obtain a list of all markers of a `MarkerLayer`, you call `getMarkers`. This will return a list of all the markers of that layer, not just the ones that are currently visible.

```
    final List<Marker> markers = layer.getMarkers();
```

Using map layers

Layers are groups of visual elements, either stock or custom. Layers can be hidden or shown individually and they are considered as an ordered set for visualisation and selection.

Stock layers

Stock layers are those defined in the StockLayers class. These layers contain predefined elements such as the route, traffic, etc. By default these stock layers are hidden. To change visibility of a stock layer, use the showLayer method.

```
    final MapController controller = mMapView.getMapController();
    controller.setLayerVisibility(StockLayers.ROUTE_LAYER, true);
    controller.setLayerVisibility(StockLayers.TRAFFIC_LAYER, true);
```

Renderable layers

A RenderableLayer is where Renderables, like Polylines, are stored so they will be drawn onto the map. MapController has one RenderableLayer that is always present and can be obtained by calling getRenderableLayer(). Since RenderableLayer is a composite, it's possible to place one or more RenderableLayer inside another. Just create a new RenderableLayer and add it to an existing RenderableLayer.

Renderables have a Z-index property. The Z-index determines where in the layer this Renderable will be drawn, relative to other Renderables. A higher value means it will get drawn above any Renderable with a lower Z-index. The order in which Renderables with the same Z-index are drawn is undefined; if order is important, use the Z-index property to control the order. The default Z-index is zero.

When you do not need a Renderable any more, you need to free its allocated resources by calling Renderable.release().

Polylines

The Maplib SDK offers the ability to draw polylines on the map. A polyline is a continuous line composed of one or more line segments defined in GeoPoint, consisting of a latitude and a longitude value.

A PolylineBuilder object is used to create a new Polyline. You can add the points to it, and set other properties like line width and color. Next you add the polyline to a RenderableLayer. The line segments are drawn between the points in the order in which you added them to the PolylineBuilder.

The following code snippet illustrates how to add a polyline to a map:

```
        final RenderableLayer renderableLayer = mMapController.getRender-
ableLayer();
        renderableLayer.add(new PolylineProperties()
                .color(Color.RED)
                .width(5f)
                .add(new GeoPoint(52.375663, 4.907869))
                .add(new GeoPoint(52.377307, 4.900070))
                .add(new GeoPoint(52.378617, 4.902816))
                .build()
                );
```

This will produce a red polyline on the map, 5 pixels wide, in the Amsterdam area. Note that this polyline will be opaque since the colour is not given an alpha value. Transparent polylines need an alpha value less than the maximum. See the Android Color class for details.

To remove a Polyline from the map, use RenderableLayer.remove(Renderable) and use your Polyline as an argument.

Polygons

The Maplib SDK offers the ability to draw polygons on the map. A polygon is an enclosed shape that can be used to highlight areas on the map. The outline is a continuous line composed of one or more line segments defined in `GeoPoint`, consisting of a latitude and a longitude value. Polygons are self closing, which means that there is no need to repeat the first point at the end in order to get a closed shape; the line segment from the last point back to the first is implied.

Currently supported polygon types are convex and (weakly) simple concave polygons. Please note that self-intersecting polygons will not be filled but drawn as closed polylines.

A `PolygonBuilder` object is used to create a new `Polygon`. You can add the points to it, and set other properties like stroke width and fill colour. The points should be added in counter clockwise order; the result is undefined when they are not. Next you add the polygon to a `RenderableLayer`.

To briefly explain counter clockwise points: if you imagine the hours on the face of a clock to be 12 points of a polygon, then the list of points will have to be added counter clockwise: 12, 11, ..., 2, 1. The starting point is irrelevant as long as the direction is correct. Another way to look at it is that the left side of each line segment is the interior of the polygon.

The following code snippet illustrates how to add a polygon to a map:

```
        final RenderableLayer renderableLayer = mMapController.getRender-
ableLayer();
        renderableLayer.add(new PolygonProperties()
                .fillColor(Color.argb(127, 80, 80, 80))
                .strokeColor(Color.BLUE)
                .strokeWidth(7f)
                .zIndex(2)
                .add(new GeoPoint(52.376217, 4.907824))
                .add(new GeoPoint(52.376225, 4.907883))
                .add(new GeoPoint(52.376315, 4.908585))
                .add(new GeoPoint(52.376466, 4.908602))
                .add(new GeoPoint(52.376536, 4.908104))
                .add(new GeoPoint(52.376476, 4.907687))
                .build()
                );
```

This will produce a gray polygon with a blue outline on the map, the maximum of 7 pixels wide, in the Amsterdam area. Because it is given a Z-index of 2, it will be drawn on top of any other `Renderable` with a Z-index of 1 or lower.

Also note that we get 50% transparency, by giving our `fillColor` an alpha value of 50%; 127 is about half of 255, the maximum value. The same can be done for the `strokeColor`.

To remove a `Polygon` from the map, use `RenderableLayer.remove(Renderable)` and use your `Polygon` as an argument.

Tiling images to fill polygons

By default, polygons are filled using a colour but one can also use an image. This texture is a `Drawable` from your app's resources. Possible applications are zoning and spatial planning, or any other reason to indicate land use like no-go areas, range indication, permit zones, etc.

Textures are tiled horizontally and vertically, like the background of an HTML page, and they have the size of their original source, the `Drawable`. When zooming the map in or out, this size will remain the same; so the texture is not resized when the zoom level changes! This ensures that the tiles are always recognisable, instead of them getting smaller and smaller when you zoom out, for example.

The texture has its own alpha channel, with which you can make the polygon interior (partially) transparent. Changing it with `setTextureAlpha(190)` for example, which is 75% of 255, the fully opaque value, will let the map underneath your polygon shine through somewhat. Using a partially transparent `Drawable` will do the same, giving you the option to have completely transparent areas between partially transparent areas.

The texture can be made (partially) transparent. Changing it with `setTextureAlpha(190)` for example, which is 75% of 255, the fully opaque value, will let the map underneath your polygon shine through somewhat. The texture is drawn taking the source image's transparency information into account. This means that any transparent pixels in the original image will be just as transparent when used as a texture.

```
        final Polygon polygon = new PolygonBuilder()
                .add(52.373620, 4.908018)
                .add(52.373545, 4.909971)
                .add(52.374750, 4.910250)
                .add(52.374973, 4.908222)
                .texture(this, R.drawable.dots)
                .textureAlpha(190)
                .build();
```

Image overlay

Polylines and polygons provide a way to draw shapes onto the map, that are relatively easy to define, since we are going from point to point which all have the same colour. What if we want to show something more complex over an area on the map, like weather radar images, full featured floor plans for big buildings like shopping malls, or contour lines like isobars including gradients in between. Then we need to overlay that image on the map and dispense with points and lines.

The image is locked in place by two coordinates: the North-West and the South-East corner of the image. The image is rectangular and will be stretched across the map between the given locations. Non-rectangular shapes can be obtained by using transparency in the image itself, which is a separate value from the overall transparency of the entire image.

Like with the `Polyline` and `Polygon`, an `ImageOverlay` is constructed by using a builder: `ImageOverlayBuilder`. You set the required properties in the builder and let it create the `ImageOverlay` for you, which you can then add to a `RenderableLayer` so it will be displayed.

```
        final ImageOverlay overlay = new ImageOverlayBuilder()
                .nw(52.3031428, 4.9479324)
                .se(52.30177, 4.9510746)
                .image(this, R.drawable.weather)
                .alpha(190)
                .build();
```

Listen for map events

Touch events are processed by views. Therefore, listening for touch events in the Map Library is done by registering a `MapListener` with the `MapView`.

```
    mMapView.addMapListener(new MyMapListener());
```

Your `MapListener` implementation can then respond to touch events on the map.

```
    private class MyMapListener implements MapListener {
        @Override
        void onMarkerSelected(final Marker marker, final TouchType
 touchType) {
            ...
        }

        @Override
        void onRenderableSelected(final Renderable renderable, final int x,
 final int y,
                final TouchType touchType) {
            ...
        }

        @Override
        void onMapTouched(final int x, final int y, final TouchType
 touchType) {
            ...
        }

        @Override
        public void onDrag(final int prevX, final int prevY, final int
 newX, final int newY) {
            ...
        }
    }
```

`onMarkerSelected`, `onRenderableSelected`, `onMapTouched` functions will be called with:

Touch type	**When**
SINGLE_TAP	A single tap on a marker, renderable or map.
LONG_PRESS_BEGIN	The user is touching a marker, renderable or map for a prolonged time.
LONG_PRESS_END	The user stopped touching the marker, renderable or map for a prolonged time.

In order to obtain the screen position of that marker, you can use the `LocationConverter`. The `fromLocation` function, giving it the world location of that marker, will return a `Point` with the on screen position. You can use this to position popup menu's, for example.

Similiarly, to get the geopoints from screen position (x, y), also use the `LocationConverter`. For example:

```
        final Location location = mMapController.getLocationConvert-
er().toLocation(x, y);
        final GeoPoint geoPoint = new GeoPoint(location);
```

`onMarkerSelected()` callback has the highest priority. If your marker and renderable (Polygon, Polyline, Image) overlap, then you will receive an `onMarkerSelected()` callback when you touch the overlapping area.

`onRenderableSelected()` is received only when the renderable is made selectable by setting `setSelectable(true)`; Renderable's are not selectable by default. When selectable renderables overlap, the topmost renderable is returned by this callback.

`onMapTouched()` is received when there is no marker or renderable at the touch point.

Using Sensor Location

There are two ways to obtain the current location in your application. First is a map matched location which is calculated with a location provider's data and a map. Second is a non-map matched location (Sensor Location) which is your absolute location without map influence.

Enabling Sensor Location

By default your application uses the map matched location. To enable Sensor Location you need use the following `MapController` method:

```
    void setSensorLocationEnable(final boolean enable, final Drawable
 icon);
```

enabled	When `true` then map sensor location is enabled; when `false`, the matched location is used
icon	An icon to show the location as reported by the sensor. The icon must point to due north at the top of the image.

You can provide your own icon using a drawable. This icon will change its orientation depending on the bearing, the top of the image pointing to the direction of travel.

Important! Always disable the Sensor Location when not needed since listening to the location consumes more battery power.

```
    @Override
    public void onStart() {
        super.onStart();
        final MapController controller = mMapView.getMapController();
        controller.setSensorLocationEnable(true);
    }
    ...
    @Override
    public void onStop() {
        super.onStop();
        final MapController controller = mMapView.getMapController();
        controller.setSensorLocationEnable(false);
    }
```

Retrieving Sensor Location state

To know the state of the current location provider you can use this `MapController` method:

```
    boolean isSensorLocationEnabled();
```

`true`	The current location is continously set using the GNSS sensor information.
`false`	The current location is never set using the GNSS sensor information.

Installing the Map Library SDK

Copy the **maplibsdk.jar** and **mapviewer2.jar** files into your application project `libs` directory. The files can be found in the `libs` directory in the SDK Zip file which you downloaded.

Because the Map Library communicates with NavKit via a network interface, your application is required to have the `INTERNET` permission, even though you might not do anything with networking. So add this to your applications `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Defining the view in the layout XML

```
    ...
    <com.tomtom.pnd.maplib.MapView
        android:id="@+id/mapview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
    ...
```

Initialising and interfacing with the SDK

The Map Library uses the same MapKit engine as the Navigation application does. The MapKit engine is used via runtime connections, which means that these connections might disappear at some point. For this reason you cannot just control your map, you need to maintain a connection to it. You are shielded from the details of this and only have to monitor two simple events to know whether or not your map is ready for use.

You need to implement the `MapEventsCallbacks` interface and set an instance of this callback on the `MapView` object. The first step is to implement the callback interface:

```
public class MainActivity extends FragmentActivity
    implements MapEventCallbacks {
    ...
}
```

Initialise the `MapView` object in your application and request a `MapController` instance with the `getMapControllerAsync` call:

```
    @Override
    protected void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mMapView = (MapView) findViewById(R.id.mapview);
        mMapView.getMapControllerAsync(this);
        ...
    }
```

Use the `onMapConnected(MapController)` callback method to get a handle to the `MapController` object. The callback is triggered when the `MapController` is ready to be used. It provides a non-null instance of `MapController`.

You can use the `MapController` object to set the view options for the map or add markers, for example. In `onMapDisconnected()` you need to set your reference to the `MapController` to null and stop interacting with the map until the connection is restored.

```
    @Override
    public void onMapConnected(final MapController controller) {
        mMapController = controller;
    }

    @Override
    public void onMapDisconnected() {
        mMapController = null;
    }
```

The overloaded `onResume()` and `onPause()` need to inform the `MapView` when the activity enters these states. Otherwise MapKit will keep using resources like a foreground application.

```
    @Override
    protected void onResume() {
        super.onResume();
        if (mMapView != null) {
            mMapView.onResume();
        }
    ...

    @Override
    protected void onPause() {
        super.onPause();
        if (mMapView != null) {
            mMapView.onPause();
        }
    ...
```

Development tools

Taking Screenshots

With the PRO 8 Driver Terminal connected to a development PC, you can take screenshots directly to the PC by using eclipse DDMS camera functionality. In eclipse go the the DDMS view, select the PRO 8 Driver Terminal and click the **Camera** icon.

Alternatively you can take a screenshot via adb using `screencap -p >/sdcard/screen-shot.png`.

To take screenshots without a PC attached long press the power button and select **Take a screenshot**. The screenshots are stored in the following location: `/sdcard/Pic-tures/Screenshots`.

System intents

Intents listened for

The PRO 8 Driver Terminal has a few additional intents that it responds to. Find a list of these intents and their effect below. Please refer to their respective documentation for more detailed information.

Home screen

Open the home screen on the currently active page

Action	action.MAIN
Activity	com.tomtom.navpad.navapp/.NavPadNavAppActivity
Data URI scheme	None
MIME Type	None

Go to home and scroll to page number X

Action	action.VIEW
Activity	com.tomtom.navpad.navapp/.NavPadNavAppActivity
Data URI scheme	page:navigation
MIME Type	None
Category	category.HOME

Example: Go to page 2 on home screen

```
adb shell am start -a android.intent.action.VIEW -c android.intent.catego-
ry.HOME -d "page:2"
```

Go to home and scroll to the page holding the navigation widget

Action	action.VIEW
Activity	com.tomtom.navpad.navapp/.NavPadNavAppActivity
Data URI scheme	page:navigation
MIME Type	None
Category	category.HOME

Example:

```
adb shell am start -a android.intent.action.VIEW -c android.intent.catego-
```

```
ry.HOME -d "page:navigation"
```

Open up the navigation page on home screen, or full screen navigation whichever was last in use

Action	tomtom.intent.action.SHOW_NAVAPP
Data URI scheme	None
MIME Type	None

Disable or enable items in group with an index number

Action	action.VIEW	
Activity	com.tomtom.navpad.navapp/.NavPadNavAppActivity	
Data URI scheme	None	
MIME Type	None	
Extras	**enable**	true/false
	index	[0-256]
Category	category.HOME	

Navigation application

Open the full screen Navigation

Action	action.MAIN
Activity	com.tomtom.navpad.navapp/.NavPadNavAppActivity
Data URI scheme	None
MIME Type	None
Category	**category.LAUNCHER**
	category.APP_MAPS
	category.DEFAULT

Example:

```
adb shell am start com.tomtom.navpad.navapp/.NavPadNavAppActivity
```

Import a single route into the Navigation application

Action	tomtom.intent.action.IMPORT_SINGLE_ROUTE
Data URI scheme	File URI to the route
MIME Type	application/gpx application/gpx+xml application/itn
Category	category.DEFAULT

Import a list of routes into the Navigation application

Action	tomtom.intent.action.IMPORT_MULTIPLE_ROUTES
Data URI scheme	ArrayList of File URIs to the routes
MIME Type	application/gpx application/gpx+xml application/itn
Category	category.DEFAULT

Show the map at the given latitude and longitude

Action	action.VIEW
Data URI scheme	`geo:<latitude>,<longitude>` `geo:<latitude>,<longitude>?z=<zoom>` (zoom is currently ignored) `geo:0,0?q=<latitude>,<longitude>(<label>)` (with a string label)
MIME Type	None
Category	category.DEFAULT

Plan a route to the given latitude and longitude

Action	action.VIEW
Data URI scheme	`google.navigation:q=<latitude>,<longitude>` `google.navigation:q=<latitude>,<longitude>?z=<zoom>` (zoom is currently ignored) `google.navigation:q=<latitude>,<longitude>(<label>)` (with a string label)
MIME Type	None
Category	category.DEFAULT

Software updater

Remove APKs which are not installed by Software update application

Action	tomtom.intent.action.REMOVE_THIRDPARTY_APKS
Data URI scheme	None
MIME Type	None

Example:

```
adb shell am broadcast -a tomtom.intent.action.REMOVE_THIRDPARTY_APKS
```

External camera application

Stop the external camera application

Action	com.tomtom.videodockcamera.intent.action.FINISH
Data URI scheme	None
MIME Type	None

Example:

```
adb shell am broadcast -a com.tomtom.videodockcamera.intent.action.FINISH
```

System bar

Set Quick Launch button X with intent ‘intent_uri’

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

Action	tomtom.intent.action.SET_QUICKLAUNCHBUTTON	
Extras		
	button	0/1(Integer)
	intent_uri	URI string for the intent, starting with #intent..(String)

Example: Set the Quick Launch button 1 with **Settings** application for **Restricted** user

```
adb shell "am broadcast --user 10 -a tomtom.intent.action.SET_QUICKLAUNCH-
BUTTON --ei button 1 -e intent_uri '#Intent;action=android.intent.ac-
tion.MAIN;category=android.intent.category.LAUNCHER;component=com.an-
droid.settings/.Settings;end'"
```

Clear Quick Launch button X

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

Action	tomtom.intent.action.CLEAR_QUICKLAUNCHBUTTON

Extras	button	0/1(integer)

Example: Clear the Quick Launch button 0 for **Owner** user

```
adb shell am broadcast --user 0 -a tomtom.intent.action.CLEAR_QUICKLAUNCH-
BUTTON --ei button 0
```

Intents broadcast

PRO 8 Driver Terminal also broadcasts intents on certain occasions. Please refer to their respective documentation for more detailed information.

Receive a single route

Action	action.SEND
Data URI scheme	None
MIME Type	application/gpx application/gpx+xml
Category	category.DEFAULT

Receive multiple routes

Action	action.SEND_MULTIPLE
Data URI scheme	None
MIME Type	application/gpx application/gpx+xml
Extras	
Category	category.DEFAULT

Navigation app starts after a reboot

Action	com.tomtom.action.FIRSTRUN
Data URI scheme	None
MIME Type	None
Category	category.DEFAULT

Update has started

Action	tomtom.intent.action.UPDATE_STARTED
Data URI scheme	None
MIME Type	None

System was successfully updated

Action	tomtom.intent.action.UPDATE_FINISHED
Data URI scheme	None
MIME Type	None

Software update application encountered an error

Action	tomtom.intent.action.UPDATE_ERROR
Data URI scheme	None
MIME Type	None

Device needs to suspend/shutdown

Action	android.intent.action.ACTION_REQUEST_CONFIRM_SUSPEND_SHUT-DOWN
Data URI scheme	None
MIME Type	None
Category	category.DEFAULT

State of the ignition signal has changed

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

Action	tomtom.intent.action.IGNITION_STATE_CHANGED
Data URI scheme	None
MIME Type	None

Extras		
	tomtom.intent.extra.IGNITION_ON	A boolean extra that indicates if the ignition is on or not. The ignition state can also be queried from the TomTom addon with the `DockControl.getIgnitionOn()` API.

Microphone connectivity state

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

Action	tomtom.media.TOMTOM_DOCK_MIC_PLUG_ACTION
Data URI scheme	None
MIME Type	None

Suspend Shutdown hooks

The PRO 8 Driver Terminal allows you to intervene when it is about to be suspended or shut down. It will look for apps listening for the `ACTION_REQUEST_CONFIRM_SUSPEND_SHUT-DOWN` intent and if one is installed, control over the suspend/shutdown flow is handed over to this app. This can be useful for situations where you do not want the device to suspend/shutdown because some process is not finished or users first need to log out of their own systems, for example.

The intent comes with two integer extras: `poweroff_state` and `poweroff_reason`. The former indicates the state the system is trying to get to, `1` for a shutdown and `2` to be suspended, and the reason gives an indication why the device is trying to get to that state:

Value	Reason
0	By user request, eg. the power button was pressed
2	Screen timeout
3	Disconnected from power
5	Application request

Once you have decided to proceed with the suspend/shutdown it's the app's responsibility now to make that happen!

PRO 8270 and PRO 8275

On PRO 8270 and PRO 8275 devices, the following permissions and a static broadcast receiver for ‘android.intent.action.ACTION_REQUEST_CONFIRM_SUSPEND_SHUTDOWN’ are required:

```
<!-- Permissions needed -->
<uses-permission android:name="android.permission.DEVICE_POWER" />
<uses-permission android:name="android.permission.SHUTDOWN" />
...
<!-- Priority needed > 5 -->
<intent-filter android:priority="10" >
  <action android:name="android.intent.action.ACTION_REQUEST_CONFIR-
M_SUSPEND_SHUTDOWN" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Suspending a device is not possible anymore with the public Android API (starting with Android 5) and requires the use of reflection:

```
 private void suspend() {
        // With API 21, goToSleep was removed from the PowerManager API but
 is still
        // publicly available via reflection.
        try {
            final Class<?> PowerManagerClazz = Class.forName("an-
droid.os.PowerManager");
            final Method goToSleep = PowerManagerClazz.getMethod("go-
ToSleep", long.class);
            final PowerManager pm = (PowerManager) context.getSystemSer-
vice(Context.POWER_SERVICE);
            goToSleep.invoke(pm, SystemClock.uptimeMillis());
        } catch (final SecurityException | NoSuchMethodException | Ille-
galAccessException | InvocationTargetException | ClassNotFoundException e)
 {
            Log.e(TAG, e.getMessage(), e);
        }
}
```

PRO 8475

On PRO 8475 devices, the following permissions and a static broadcast receiver for 'com.webfleet.prosystemapp.ACTION_REQUEST_CONFIRM_SUSPEND_SHUTDOWN' are required:

```
<uses-permission android:name="com.webfleet.prosystemapp.permission.RE-
QUEST_SUSPEND_SHUTDOWN"/>
...
<receiver
    android:name=".SuspendShutdownReceiver"
    android:exported="true"
    android:enabled="true"
    android:permission="com.webfleet.prosystemapp.permission.RE-
QUEST_SUSPEND_SHUTDOWN">
    <intent-filter android:priority="10">
        <action android:name="com.webfleet.prosystemapp.ACTION_REQUEST_CON-
FIRM_SUSPEND_SHUTDOWN" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</receiver>
```

The priority of the broadcast receiver should be higher than 0 to receive the power intent.

An example broadcast receiver looks as follows:

```
public class SuspendShutdownReceiver extends BroadcastReceiver {
    public enum PowerOffReason {
        USER(0),
        TIMEOUT(2),
        UNPLUGGED(3),
        SYSTEM(5),
        UNDEFINED(-1);

        private final int val;

        PowerOffReason(int val) {
            this.val = val;
        }

        public int getVal() {
            return this.val;
        }

        public static PowerOffReason fromInt(int val) {
            for (PowerOffReason reason : PowerOffReason.values()) {
                if (val == reason.getVal()) {
                    return reason;
                }
            }
            return UNDEFINED;
        }
    }

    public enum PowerOffState {
        SHUTDOWN(1),
        SUSPEND(2),
        UNDEFINED(-1);

        private final int val;

        PowerOffState(int val) {
            this.val = val;
        }

        public int getVal() {
            return this.val;
        }

        public static PowerOffState fromInt(int val) {
            for (PowerOffState state : PowerOffState.values()) {
                if (val == state.getVal()) {
                    return state;
                }
            }
            return UNDEFINED;
        }
    }

    public static final String ACTION_REQUEST_CONFIRM_SUSPEND_SHUTDOWN =
  "com.webfleet.prosystemapp.ACTION_REQUEST_CONFIRM_SUSPEND_SHUTDOWN";
    public static final String REQUEST_SHUTDOWN_ACTION = "com.tomtom.p-
nd.navpadsystemapp.ACTION_REQUEST_SHUTDOWN";
    public static final String REQUEST_SUSPEND_ACTION = "com.tomtom.p-
nd.navpadsystemapp.ACTION_REQUEST_SUSPEND";
    public static final String POWEROFF_REASON_KEY = "poweroff_reason";
    private static final String POWEROFF_STATE_KEY = "poweroff_state";
    private static final String PRO_SYSTEM_APP = "com.tomtom.pnd.navpadsys-
temapp";

    @Override
    public void onReceive(final Context context, final Intent intent)
    {
        if (ACTION_REQUEST_CONFIRM_SUSPEND_SHUTDOWN.equals(intent.getAc-
tion()))
        {
            final Bundle bundle = intent.getExtras();
            if (bundle != null)
            {
                final KeyguardManager km = (KeyguardManager) getApplica-
tionContext().getSystemService(Context.KEYGUARD_SERVICE);
                final int powerOffReasonValue = bundle.getInt(POWEROF-
F_REASON_KEY, -1);
                final PowerOffReason powerOffReason = PowerOffRea-
son.fromInt(powerOffReasonValue);
                final int powerOffStateValue = bundle.getInt(POWEROFF_S-
TATE_KEY, -1);
                final PowerOffState powerOffState = PowerOffState.fromIn-
t(powerOffStateValue);

                switch (powerOffReason)
                {
                    case UNPLUGGED:
                        // intended fall-through
                    case SYSTEM:
                        // intended fall-through
                    case USER:
                        // If there a screen lock in place, always suspend
 or shutdown the device immediately
                        if (km.isKeyguardLocked()) {
                            // In order to suspend the device, an intent
 needs to be sent back to the PRO system app
                            if (powerOffState == PowerOffState.SUSPEND) {
                                final Intent suspendIntent = new Intent(RE-
QUEST_SUSPEND_ACTION);
                                suspendIntent.setPackage(PRO_SYSTEM_APP);
                                context.sendBroadcast(suspendIntent);
                            // In order to shut down the device, an intent
 needs to be sent back to the PRO system app
                            } else if (powerOffState == PowerOffState.SHUT-
DOWN) {
                                final Intent shutdownIntent = new Inten-
t(REQUEST_SHUTDOWN_ACTION);
                                shutdownIntent.setPackage(PRO_SYSTEM_APP);
                                context.sendBroadcast(shutdownIntent);
                            } else {
                                Log.w("SuspendShutdownReceiver", "Unsup-
ported power off state: " + powerOffState);
                            }
                        }
                        break;
                    case TIMEOUT:
                        // In case of a screen timeout we directly suspend
 the device
                        final Intent suspendIntent = new Intent(RE-
QUEST_SUSPEND_ACTION);
                        suspendIntent.setPackage(PRO_SYSTEM_APP);
                        context.sendBroadcast(suspendIntent);
                        break;
                    default:
                        break;
                }

            }
        }
    }
}
```

Power off states and reasons

The following power off states and reason are sent when registering for suspend/shut-down hooks on PRO 8270, PRO 8275 and PRO 8475 devices:

Value	Power off state
1	Shutdown
2	Suspend

Value	Power off reason
0	By user request, eg. the power button was pressed

Value	**Power off reason**
2	Screen timeout
3	Disconnected from power
5	Application request

Additional features

Connecting to an external camera

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

PRO 8 Driver Terminal devices support an external video camera connection – available with each cradle of PRO 8475, PRO 8375 or via the video cradle for TomTom PRO 8275 (shipped with TRUCK version).

Applications that want to use an external camera, should use the standard Android Camera API.

Here are some hints for using the external camera in app development:

1. The external camera is identified as the front facing camera.
2. The external camera has a constant resolution of 640x480. It does not depend on the actual camera hardware that is used.
3. The intent sent when the external video becomes available is:
   ```
    "tomtom.intent.action.VIDEO_AVAILABLE"
   ```
   and when it becomes unavailable:
   ```
   "tomtom.intent.action.VIDEO_UNAVAILABLE"
   ```
4. When `tomtom.intent.action.VIDEO_AVAILABLE` is sent, it means that the video dock was just connected and the initialisation of the video camera has started. It might take up to 3 seconds to set up the camera. In this case, `Camera.open(camId)` will throw a `RuntimeException`. To deal with this situation consider the following piece of code:
   ```
     int count = 0;
     Camera camera = null;
     do {
         try {
             for (int camId = 0; camId < Camera.getNumberOfCameras(); camId
   ++) {
                 CameraInfo camInfo = new CameraInfo();
                 Camera.getCameraInfo(camId, camInfo);
                 if (camInfo.facing == CameraInfo.CAMERA_FACING_FRONT) {
                     camera = Camera.open(camId);
                     break;
                 }
             }
         } catch (RuntimeException ex) {
         }
         if (camera == null) {
             try {
                 Thread.sleep(100);
             } catch (InterruptedException e) {
                 Log.w(TAG, "Sleep was interrupted.", ie);
                 break;
             }
         }
     } while ((camera == null) && (count++ < 30));
   ```

Android application installer

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

App installer was developed to allow a quick installation of Android applications without using the Software Update mechanism. It is not intended for final configuration of the device.

Note: All apps installed by the Android application installer will be removed if a user does **Personal Data Reset** or a factory reset.

To install an android app on the device follow the next steps:

1. Put an Android app .apk file to the root folder of a micro SD card.
2. Insert the micro SD card in the PRO 8 Driver Terminal.

   A dialog will appear on the screen, prompting to install one or more Android Applications from the SD card.
3. Click **Install** and the standard Android installation dialog will appear.
4. Follow the steps in the dialog to install the application(s).

Important! If **Unknown sources** is disabled in **Security settings**, the user will not be prompted for installation.

Privacy disclaimers

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

If your application records or collects privacy sensitive information, you should inform the user of this. To keep this clear to the user, we've provided a mechanism to show all these disclaimers in one location: **Settings** > **About your device** > **Legal information** > **Your information**. You'll see all apps listed there and tapping them will show you their privacy information.

To list the privacy disclaimer of your application there, you need to add a `meta-data` element called `privacy_disclaimer` to the application element of the application's `AndroidManifest.xml`, defining a reference to a string resource via the `android:resource field`. For example:

`AndroidManifest.xml`

```
    <application android:name="CustomApp"
                 android:label="@string/application_name"
                 android:icon="@mipmap/ic_launcher_custom_app" >

        <meta-data android:name="privacy_disclaimer"
                   android:resource="@string/privacy_disclaimer"/>
        ...
    </application>
```

`strings.xml`

```
    ...
    <string name="privacy_disclaimer"><b>General</b>\nThis is a stylized
 <i>privacy disclaimer</i>.</string>
    ...
```

Wake on SMS

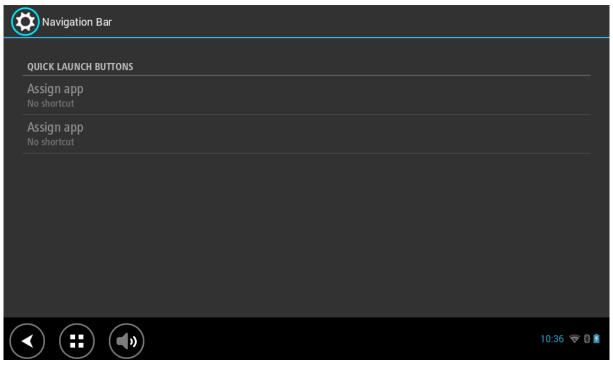
Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

PRO 8 Driver Terminal can be woken up from sleep by sending an SMS to it. The screen is not turned on by default, but an intent `tomtom.intent.action.MODEM_WAKEUP` is broadcasted by the system. Upon receiving the intent, applications can decide whether to turn on the display or to do some work and sleep again. The intent `tomtom.intent.action.MODEM_WAKEUP` is only sent to the current user to prevent multiple instances of the same application from retrieving the intent.

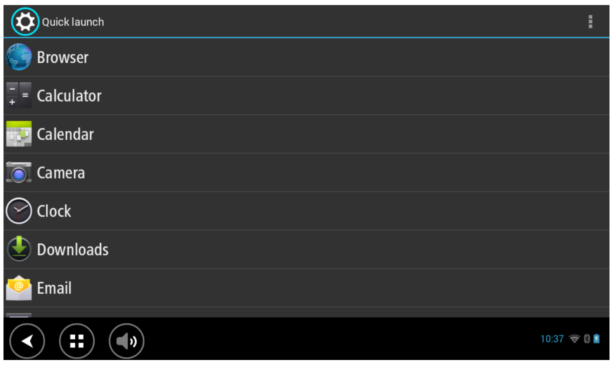
Customise

System Bar

The Navigation Bar allows you to add quick launch buttons to make switching between (or launching) applications easier. To customise the bar go to the settings screen and click the **System Bar**. You can add up to two quick launch buttons to the system bar.



Select **Assign app** and a list of the applications available on the system will appear:



Select the desired app and it will appear in the system bar. To delete previous quick launch items, long press the button you used to assign the app in the **System Bar Settings** screen.

Home screen

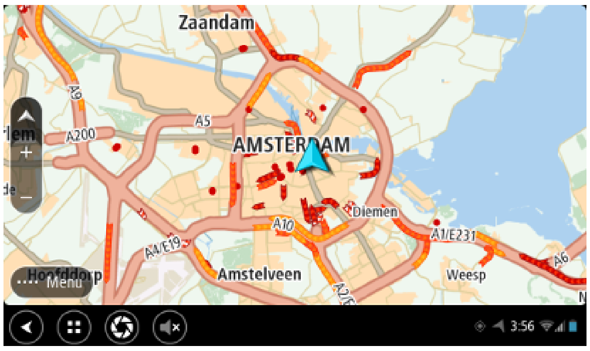
About the Home screen

End users generally interact with the PRO 8 Driver Terminal in three ways:

1. Via the home screen (sometimes called the launcher)

   In this mode of operation users can follow the navigation via the navigation widget, clicking on it to go into full screen navigation mode. Users can interact with third party widgets and applications installed onto the home screen.
2. Full screen Navigation

   In this mode the user can interact with the full screen navigation application. It is not possible to place widgets or application onto this fullscreen view.

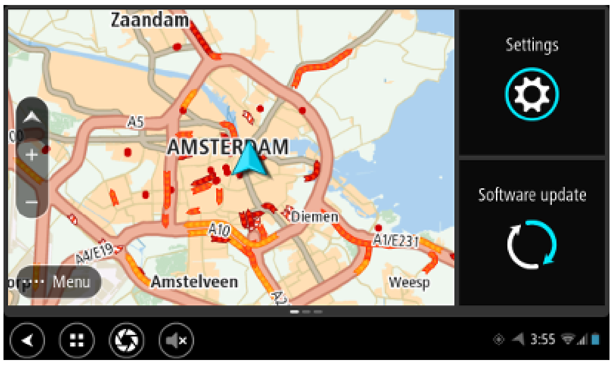
3. A 3rd party or pre-installed application

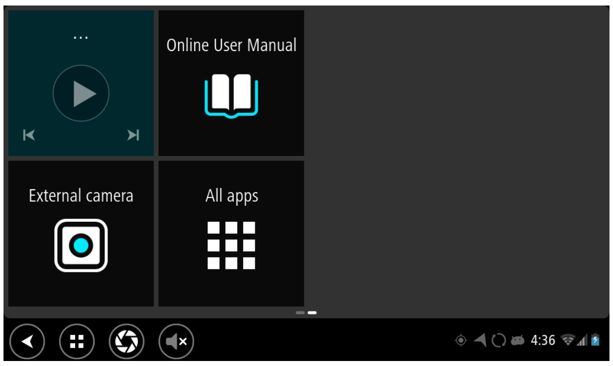
   In this mode of use an application has been launched and is displayed full screen for the user to interact with. Pushing home will return the user to the home screen.

Driver Terminal

The default PRO 8 Driver Terminal home screen consists of eight user-definable tiles, laid out in two rows by four columns. This configuration provides an open environment for you to develop applications and configure the PRO 8 Driver Terminal for your end users.

The default configuration begins with a home screen containing a 3/4 screen navigation widget and two tiles on the right side of the screen. Page two of the home screen contains shortcuts to more apps and the **All Apps** browser.





Customising the home screen

It is possible to define multiple pages containing tiles, although in practice 2 or 3 pages should suffice. On startup page 1 is shown - the user can then swipe between the pages. Tiles are identified by a number in the following format: {page}.{row}{column}

So for page 1, tile numbers 1.11 to 1.24 are available. In this example, only one home screen page will be displayed.

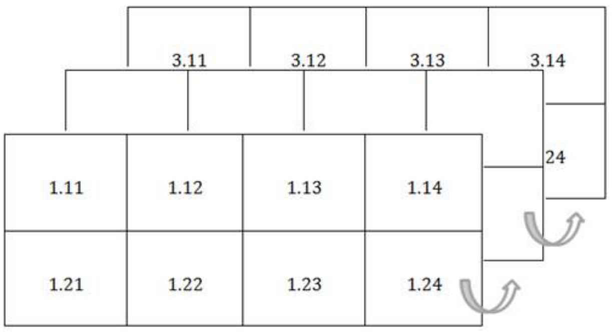
1.11	1.12	1.13	1.14
1.21	1.22	1.23	1.24

Create multiple pages

To define a second page, just change the '1' into a '2' for two pages or a '4' to define four pages.

Creating blank pages

To create blank pages, define at least 3 pages and only put items on pages 1 and 3.

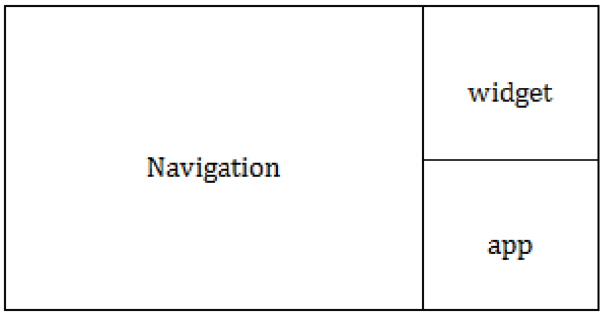


Configuring the tiles

Tiles can be populated with short cuts to applications or interactive widgets. Short cuts are buttons with an icon and title text, when the user clicks on them, they will launch a certain application full screen. Widgets are interactive views which can span multiple tiles, they can display dynamically updated or animated information, users can sometimes interact with them or launch fullscreen applications by clicking on them. Widgets on a PRO 8 Driver Terminal are standard Android widgets.

The PRO 8 Driver Terminal home screen can be configured to show a navigation widget on one of it's pages, spanning multiple tiles. You can also configure a tile to display an **All apps** drawer which when clicked on will display tiles for each application currently installed on the PRO 8 Driver Terminal.

Below is an example layout of a home screen page.



Screen orientation

If screen rotation is enabled on a device with a custom tile layout it is important to consider this when adding home screen layouts so that layouts for both orientations are covered, if desired.

In the event that no valid integrator configurations are present to support screen rotation then the current screen orientation will be kept and an error message will be shown until the user rotates the display back to the previous orientation.

Note that for a legacy 4x2 layout (in widgets.json) rotation will not be possible - landscape will be enforced.

Using a legacy configuration (widgets.json)

Some older setups (17.1 and earlier) may use tile layouts in a file called widgets.json, located in the `/sdcard/` directory. If this is already present it will be used as a fall back in the event that no valid configurations are found in the `/sdcard/homescreen/` directory. In case valid configurations are present in both locations the ones in `/sdcard/homescreen/` will take precedent.

Section types

The widgets.json file is split into sections, each section defining the location of items relating to that section. You can define items for the following sections:

- navigation
- widget
- application
- video
- app-browser

navigation

This (simple) section is used to define where the navigation widget will be displayed and the dimensions of the widget. This is an optional section, you can choose not to have a navigation widget on the home screen - in this case access to navigation should be provided by adding an application short cut (to the navigation apk) on one of your home screen pages, alternatively the application shortcut will always be available in the application browser.

Requirements:

- Only one instance of navigation is allowed
- Minimum height and width are 2 x 2 tiles
- Maximum width is 4 tiles, maximum height 2 tiles

Explanation of the fields:

- **'startcell'** [mandatory] is to define the start position of navigation. Value has to be lower than endcell. e.g. startcell is 1.11 and endcell is 1.12
- **'endcell'** [mandatory] is to define the end position of navigation

widget

This section explains how to add standard Android widgets to your home screen. Widgets can encompass more than one tile.

Widgets will only appear on the home screen if they are installed and configured correctly. If your widget is not appearing in the home screen then you should check the error logs for more details.

To see which widgets are installed on your device, please read Exporting a package list below in this document.

Requirements:

- Widgets are optional components of the home screen
- Widgets should at least be the minimum size allowed
- A widget can span over more than one tile but height and width should limit it to one page

Explanation of the fields:

- **'package_name'** [mandatory]. Read below Exporting a package list
- **'class_name'** [mandatory]. Read below Exporting a package list
- **'startcell'** [mandatory] is to define the start position of a widget. Value has to be lower than 'endcell', unless the widget is 1 X 1 size.; e.g. startcell is 1.11 and endcell is 1.12
- **'endcell'** [mandatory] is to define the end position of a widget. Value has to be the same or higher than 'startcell' e.g. startcell is 1.11 and endcell is 1.12 or higher
- **'background_color'** [optional] is to define the background color of an application tile. It uses the Android constant values e.g. 0xffffffff

```
"widget":
[
    {
        "package_name: "",
        "class_name": "",
        "startcell": "",
        "endcell": "",
        "background_color": ""
    }
]
```

application

In the application section you can create short cuts to Apps installed on your device. To see which Apps are installed and more importantly to get the package and class name information, read Exporting a package list below.

Requirements:

- Apps are optional components of the home screen
- Apps cannot span, they are fixed to one tile

To define apps in the widgets.json file there are mandatory and optional fields:

- **'startcell'** [mandatory] is to define the start position of an instance
- **'package_name'** [mandatory]. Read below Exporting a package list
- **'class_name'** [mandatory]. Read below Exporting a package list
- **'background_color'** [optional] is to define the background colour of an application tile. It uses the Android constant values e.g. `0xffffffff`

  The background colour can also be transparent: value 0x00000000. Syntax: `'background_color': '0xffcccccc'`
- **'text_color'** [optional] is to define the text color of an application tile. It uses the Android constant values e.g. `0xff000000`. Syntax: `'text_color': '0xffcccccc'`
- **'icon'** [optional] is to define a user icon instead of the default Android icon. Supported filetype is .png
- **'round_corners'** [optional] is to define (the amount of) rounded corners on the background tile. Values range from 0 (no roundness) to 255 (Max roundness) - default is 0
- **'enable'** [optional] takes an enable slot identifier (0-255), multiple items can use the same slot, slots can be enabled or disabled via intents, default is that slot 0 is disabled and slots 1-255 are enabled. Use this functionality to disable (sets of) icons runtime (for example - disabling items when driving)

There are 255 slots available. Each slot can be either enabled or disabled. Application shortcuts which you do not allocate a slot to (with 'enable') will always be enabled and cannot be disabled, ever. Slots are setup automatically at bootup, slot 0 is disabled, all other slots enabled.

```
"application":
[
    {
        "name": "app 1",
        "package_name": "",
        "class_name": "",
        "startcell": "",
        "background_color": "",
        "text_color": "",
        "icon": "user_icon.png"
    }
]
```

video

This section is used to define where the video widget will be displayed and the dimensions of the widget. This is an optional section, you can choose not to have a video widget on the home screen. Unlike the Video application which is always fullscreen, you can specifiy the video widget dimensions on the home screen. When the device is connected to a video feed, the streaming video should appear in the video widget area. Clicking on the video widget will start a fullscreen Video application. The fullscreen Video application is also triggered when it detects a video feed. You can stop the autostart of the fullscreen Video application by adding a file `rear_view_camera_no_autostart` in the `/mnt/sdcard/` directory of your device.

Requirements:

- Only one instance of video is allowed
- No size restrictions, however since it retains its aspect ratio its better to have it in 1x1 or 2x2 tiles
- Maximum width is 4 tiles, maximum height 2 tiles
- To prevent VideoApplication from autostart, add a blank file named rear_view_camera_no_autostart in the /mnt/sdcard/ directory of your device

Explanation of the fields:

- **'startcell'** [mandatory] is to define the start position of video. Value has to be lower than 'endcell'. e.g. startcell is 1.11 and endcell is 1.12
- **'endcell'** [mandatory] is to define the end position of video

app-browser

In the application section of the widgets.json you can enable apps installed as on your device. To see which apps are installed, read Exporting a package list.

Requirements:

- Apps are optional components of the home screen
- Apps cannot span, they are fixed to one tile

To define apps in the widgets.json file there are mandatory and optional fields:

- 'startcell' [mandatory] is to define the start position of an instance
- "package_name" [mandatory]. Read below Exporting a package list
- "class_name" [mandatory]. Read below Exporting a package list
- "background_color" [optional] is to define the background colour of an application tile. It uses the Android constant values e.g. `0xffffffff`

  The background colour can also be transparent: value `0x00000000`. Syntax: `'background_color': '0xffcccccc'`
- 'text_color' [optional] is to define the text colour of an application tile. It uses the Android constant values e.g. `0xff000000`. Syntax: `'text_color': '0xffcccccc'`
- 'icon' [optional] is to define a user icon instead of the default Android icon. Supported filetype is .png
- 'round_corners' [optional] is to define (the amount of) rounded corners on the background tile. Values range from 0 (no roundness) to 255 (Max roundness) - default is 0
- 'enable' [optional] takes an enable slot identifier (0-255), multiple items can use the same slot, slots can be enabled or disabled via intents, default is that slot 0 is disabled and slots 1-255 are enabled. Use this functionality to disable (sets of) icons runtime (for example - disabling items when driving)

```
"application":
[
    {
        "name": "app 1",
        "package_name": "",
        "class_name": "",
        "startcell": "",
        "background_color": "",
        "text_color": "",
        "icon": "user_icon.png"
    }
]
```

Updating application icons

When you want to update the icon of an app, add a folder in the same directory where widgets.json is located. The code should look like this:

```
{
    "name": "Earth",
    "package_name": "com.google.earth",
    "class_name": ""com.google.earth.EarthActivity,
    "startcell": "1.24",
    "background_color": "0xffcccccc",
    "icon": "/folder/icon.png"
}
```

Exporting a package list

To find out what packages are installed on your device you can use the app PackageLister. After executing this app two buttons are available: **Generate homescreen-list** and **Generate package-list file**.

A file will be exported to `/sdcard/packagelister/homescreen-list`. This file contains all installed widgets and apps on that specific Android device. You can use these to define the home screen.

The PackageLister is a developer tool and can be downloaded from the downloads section.

Example output of PackageLister:

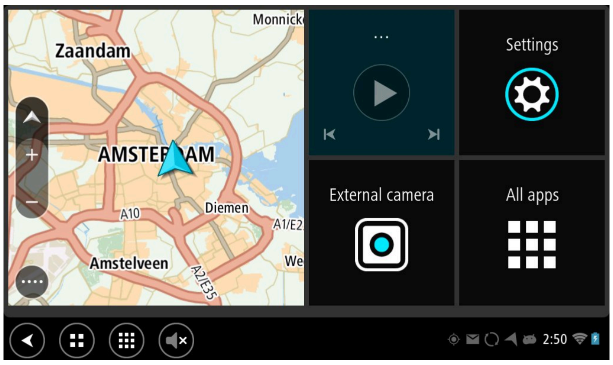
```
{
"widget":
    [
        { "package_name": "com.android.contacts", "class_name": "com.an-
droid.contacts.socialwidget.SocialWidgetProvider" },
        { "package_name": "com.google.android.apps.currents", "class_name":
 "com.google.apps.dots.android.app.appwidget.PostListAppWidgetProvider" },
        { "package_name": "com.google.android.deskclock", "class_name":
 "com.android.AnalogAppWidgetProvider" }

    ],

"application":
    [
        { "package_name": "com.android.calculator2", "class_name": "com.an-
droid.calculator2.Calculator" },
        { "package_name": "com.android.contacts", "class_name": "com.an-
droid.contacts.activities.PeopleActivity" },
        { "package_name": "com.google.android.deskclock", "class_name":
 "com.android.deskclock.DeskClock" }

    ],
...
```

Example 1 home screen layout JSON

This is an example of the code:

```
{
    "navigation":
    [
        {
            "startcell": "1.11",
            "endcell": "1.22"
        }
    ],
    "widget":
    [
        {
            "package_name": "com.google.android.gm",
            "class_name": "com.google.android.gm.widget.GmailWidget-
Provider",
            "startcell": "1.13",
            "endcell": "1.23",
            "background_color": "0xfffccccc"
        }
    ],
    "application":
    [
        {
            "name": "Calculator",
            "package_name": "com.android.calculator2",
            "class_name": "com.android.calculator2.Calculator",
            "startcell": "1.14",
            "background_color": "0xffcccccc",
            "icon": "icon.png"
        },
        {
            "name": "Earth",
            "package_name": "com.google.earth",
            "class_name": ""com.google.earth.EarthActivity,
            "startcell": "1.24",
            "background_color": "0xffcccccc",
            "icon": "icon.png"
        }
    ]
}
```

The result will look as follows – Application view of one home screen page



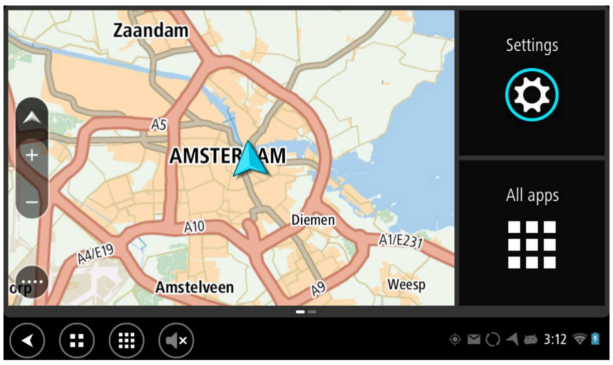
Example 2 home screen layout JSON

This is an example of the JSON code.

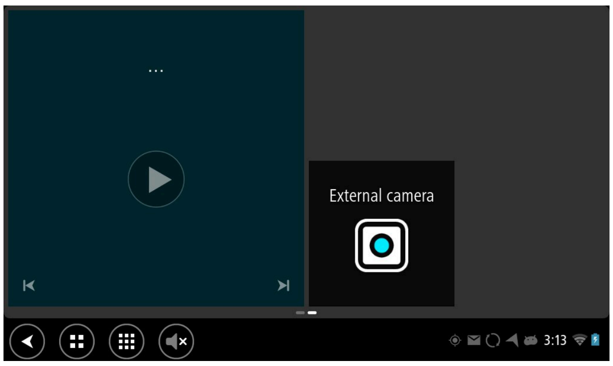
```
{
    "navigation":
    [
        {
            "startcell": "1.11",
            "endcell": "1.23"
        }
    ],
    "application":
    [
        {
            "class_name": "com.android.settings.Settings",
            "package_name": "com.android.settings",
            "startcell": "1.14",
            "endcell": "1.14",
            "background_color": "0xcc000000",
            "text_color": "0xffffffff"
        }
    ],
    "widget":
    [
        {
            "package_name": "com.tomtom.pnd.musicplayer",
            "class_name": "com.tomtom.pnd.widget.MusicWidgetProvider",
            "startcell": "2.11",
            "endcell": "2.22"
        }
    ],
    "app-browser":
    [
        {
            "startcell": "1.24",
            "endcell": "1.24",
            "background_color": "0xcc000000",
            "text_color": "0xffffffff"
        }
    ],
    "video":
    [
        {
            "startcell": "2.23",
            "endcell": "2.23",
            "background_color": "0xcc000000"
        }
    ]
}
```

Application view. Note the page indicator:

Application view of home screen page 1



Application view of home screen page 2



Adding multiple widgets and apps

```
  "widget":
     [
         {
             "package_name": "",
             "class_name": "",
             "startcell": "",
             "endcell": "",
             "background_color": ""
         },  -- note the comma when adding an extra application
         {
             "package_name": "",
             "class_name": "",
             "startcell": "",
             "endcell": "",
             "background_color": ""
         }  -- no comma after the last package
     ],
  "application":
     [
         {
             "package_name": "",
             "class_name": "",
             "startcell": "",
             "text_color": ""
             "background_color": ""
         },  -- note the comma when adding an extra application
         {
             "package_name": "",
             "class_name": "",
             "startcell": "",
             "text_color": "",
             "background_color": ""
         }   -- no comma after the last package
     ],
```

Enabling and disabling shortcuts dynamically

Application shortcuts which have been allocated an ‘enable slot identifier’ can be enabled or disabled by sending intents to the home screen process.

Testing this functionality can be achieved by sending intents from the command line:

```
adb shell am start -n com.tomtom.navpad.navapp/com.tomtom.navpad.navap-
p.NavPadNavAppActivity --es reason "enable" --ei index "3" --ez enable
```

```
 "false"
```

Programmatically an intent needs to be sent to the home screen with the following extra data.

```
(string) "reason", "enable"
(int) "index", enable-slot-identifier
(boolean) "enable", TRUE|FALSE
```

This functionality can be used for example for monitoring the vehicle speed in a background service and sending intents to enable or disable icons in the home screen when a certain speed is reached. Or preventing access to applications if the user has not logged in to a customers backend server.

Configuring the home screen with drag and drop

In the above sections, we have seen how the home screen can be configured by manually editing the widgets.json file. However, manually editing the widgets.json can introduce errors. In case of major errors in the widgets.json file, the device will revert to default layout of home screen.

To avoid this, you can make your home screen configurable. A configurable home screen enables the user to make changes to the home screen layout, by dragging and dropping rather than editing the file manually. Drag and drop is enabled in the default layout. If you do not want the default layout and have your own widgets.json file, then it can be made configurable by adding the following line to the widgets.json:

```
 "configurable": "true"
```

Below is an example layout for a configurable home screen.

```
{
    "configurable": "true",
    "navigation":
    [
        {
            "startcell": "1.11",
            "endcell": "1.23"
        }
    ],
    "application":
    [
        {
            "class_name": "com.android.settings.Settings",
            "package_name": "com.android.settings",
            "startcell": "1.14",
            "endcell": "1.14",
            "background_color": "0xcc000000",
            "text_color": "0xffffffff"
        }
    ],
    "widget":
    [
        {
            "package_name": "com.tomtom.pnd.musicplayer",
            "class_name": "com.tomtom.pnd.widget.MusicWidgetProvider",
            "startcell": "2.11",
            "endcell": "2.22"
        }
    ]
}
```

A configurable home screen allows the user to add, move and delete widgets and applications on the homescreen. Unlike applications which occupy only one tile on the home screen, a widget can be resized on a configurable home screen. The user makes a change to home screen layout by dragging and dropping, a new layout is written to a file called the x-user.json, where x corresponds to the current tile layout (e.g. 5x3-user.json). This is a per user setting and is stored in the `/sdcard/homescreen/user` directory.

If you want to allow your users to add their own apps and widgets from a custom layout remember to include **All apps** and **Add widget** to their home screen.

Settings

In order to roll out Android settings to devices you must first export these settings from one device and then put that backup in a configuration for other devices.

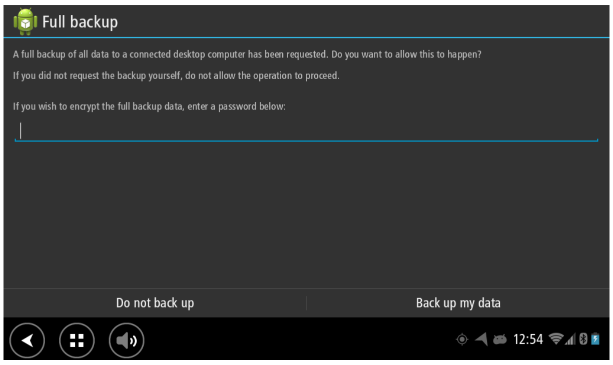
Note: The settings that are rolled out to users will overwrite all their current settings.

Exporting Settings

In order to configure all devices in a fleet or set up an SD card which will contain settings to be cloned onto multiple devices, you must first set up your own device via the settings menu to the correct configuration. When you have set the correct settings you can export the settings to a file using the following command in adb:

```
adb backup -system android com.android.providers.settings
```

The following pop-up will be shown:



Full backup

A full backup of all data to a connected desktop computer has been requested. Do you want to allow this to happen?

If you did not request the backup yourself, do not allow the operation to proceed.

If you wish to encrypt the full backup data, enter a password below:

Click **Back up my data** and the settings will be exported to a file called `backup.ab` on your computer.

Rolling out settings to devices

To learn how to roll out settings to your PRO 8 Driver Terminal refer to the FAQ [How do I create a settings zip file]() in our Support portal on [www.webfleet.com/support]().

External Camera Application

Settings Customisation

The External Camera application settings can be pre-set and distributed to the end users. Use the user interface of the **External Camera** application to change to the desired settings. These changes to the settings are automatically exported to the following file: `/sdcard/tomtom/camera_settings_export.json`.

Once the settings are customised this file should be renamed to `camera_settings_import.json` before distributing it to the end user devices as `/sdcard/tomtom/camera_settings_import.json`. This settings file is read every time the application starts.

The values of the settings file will only be imported into the end users application if:

- The file `/sdcard/tomtom/camera_settings_export.json` does not exist on the end-user device. (None of the default - or previously imported - settings have yet been changed using the user interface), and
- The `reset` attribute of the `camera_settings_import.json` file equals `true`.

Settings Attributes

The structure of the settings file is as follows, using the default values as an example:

```
{
    "reset": false,
    "configurable": true,
    "view_mode": "fit",
    "aspect_ratio": "auto",
    "full_screen": false,
    "mirrored": false
    "guidelines": {
        "enable": false,
        "left_top_x": 38,
        "left_bottom_x": 30,
        "right_top_x": 62,
        "right_bottom_x": 70,
        "top_y": 41,
        "bottom_y": 98
    }
}
```

reset	Whether or not these settings should overwrite the current settings of the application. The accepted value is a boolean `true` or `false`.
configurable	Whether or not the user is allowed to modify settings of the application. The user interface will show/hide configuration buttons based on this setting. The accepted value is a boolean `true` or `false`.

view_mode	Is the image position in the settings. Accepted value is a string `fit`, `fill` or `stretch`.

Important! The value is applied depending on the device's orientation (portrait or landscape)!

fit	Fits the image to the screen. Aspect ratio of the image is maintained and the whole image is always displayed. If you have a 4:3 image then you will have bars on the sides (landscape) or top and bottom (portrait).
fill	Aspect ratio of the image is maintained, but the whole screen will be used. If you have a 4:3 image then the top/bottom of the image will be cut (landscape) and right/left of the image will be cut (portrait).
stretch	Will stretch or squeeze the image to fit the screen, aspect ratio is not maintained. If you have a 4:3 image the image will be stretch left to right (in landscape) or top to bottom (in portrait).

aspect_ratio	Aspect ratio of the video preview. Accepted value is a string `sixteen_nine`, `four_three` or `auto`.
full_screen	Whether or not the video preview should be shown in full screen. The accepted value is a boolean `true` or `false`.
mirrored	Whether or not the video preview should be mirrored. The accepted value is a boolean `true` or `false`.
guidelines	The relative position of Guideline points as a percentage of the screen width (x) or height (y).

enable	Whether or not the guidelines view should be enabled. The accepted value is a boolean `true` or `false`.
left_top_x	Distance of the left top point, from left side of screen as percentage of the total width.
left_bottom_x	Distance of the left bottom point, from left side of screen as percentage of the total width.
right_top_x	Distance of the right top point, from left side of screen as percentage of the total width.
right_bottom_x	Distance of the right bottom point, from left side of screen as percentage of the total width.
top_y	Distance of the top line, from top side of screen as percentage of the total height.
bottom_y	Distance of the bottom line, from top side of screen as percentage of the total height.

All the guideline points are validated as follows:

- All negative values are converted to `0`.
- All values above 100 are converted to 100.
- If `right_top_x` is less than `left_top_x`, then the values will be switched.
- If `right_bottom_x` is less than `left_bottom_x`, then the values will be switched.
- If `bottom_y` is less than `top_y`, then the values will be switched.
- The minimum distance between two points horizontally is 10% of screen width and will be enforced, if not set correctly, by adjusting the point at the right.
- The minimum distance between two points vertically is 10% of screen height and will be enforced, if not set correctly, by adjusting the point at the bottom.

Start-up experience

Note: Available only for TomTom PRO 827x devices and not for PRO 8475 and PRO 8375.

Custom boot animation

Your PRO 8 Driver Terminal device will display the Webfleet Solutions screen after pushing the power button to start up the device. This actions is not customisable. After a few seconds and during start-up of the Android OS, it is possible to display your own android boot animation.

To create your own boot animation you can follow the information from this tutorial on android boot animations.

The format of the boot animation should be ZIP and the file should be called `bootanimation.zip`.

The file should be placed in the following directory: `/sdcard/bootanim/`

Please note, the bootanimation.zip creation requires some special attention:

- The ZIP should not be compressed! Files should be using the STORE method, which means they get added to the ZIP verbatim. Android does not want to be bothered with decompression at boot time.
- If you use PNG's, make sure they are not interlaced and have no transparency. You can also use JPEG's but usually animation requires the more crisp loss-less compression of PNG over the lossy photo data compression of JPEG to prevent artifacts.
- The animation will always be centered in both dimensions. Therefore, you can keep the images small and achieve a higher frame rate without delaying the actual booting too much. For this reason and to be compatible with both landscape and portrait devices, the suggestion is to make them square.
- To ensure this is rolled out to all devices correctly using the software update be sure to store the boot animation in a directory call bootanim and then zip the whole directory.

Copyright notices