Custom Screens and Dashboards

Method of creating custom screens requires at least essential knowledge of developing Android and XML files system. Otherwise we recommend to use already created skins. To edit them, use just simple NotePad or any text editing software with XML syntax highlight (for example Notepad++)

For most people who want to create their own screens should be enough using much simpler **dashboard** system that offers similar functionality in much less painful way.

However, **Dashboard** is available only for Pro version and misses some advanced possibilities like rotating images, completely free layout etc.

Method using Eclipse

1. Download and install Eclipse and Android SDK tools from here >>.
2. When all works, create a new android project. This will create basic data structure. You'll not need to program anything.
3. The project contains *res/layout/main.xml* file - that's our target. Learn to work with Eclipse GUI builder and edit this file.
4. Put resources (best to use 9-patch images) into *res* folders (supported - *drawable, drawable-hdpi, drawable-mdpi, drawable-ldpi*)
5. After compilation, take *mail.xml* file and image resources from compiled source (this is because of 9-patch images. If you want to use basic .png files, you can add them directly into result. 9-patch have to be pre-compiled!).

Using basic text editor

You do not have to do previous steps. You can edit xml file directly in any editor. Previous steps are just recommended way for easy “Drag and drop” creating and also only way to support 9-patch images!

How to make it work

root directory: *./Locus/data/customScreen/*

Create .ZIP file that contains this structure:

```
/assets/ - for custom fonts
/drawable/ - (9-patch, or universal)
/drawable-hdpi/
/drawable-ldpi/
```

```
    /drawable-mdpi/
    /layout/ - for layout specification (for landscape and portrait or...
below... for separate by orientation)
    /layout-land/
    /layout-port/
    /values/ (supported colors.xml, styles.xml)
```

Place your main layout file into layout directory (or two files, one to layout-land, second to layout-port) and rename it to *main.xml*. That is important because *main.xml* file is gate to whole layout, it has to be included!

You can place these files also in root of you .ZIP file (optional)

- **icon.png** - small (48x48px) logo of your skin
- **info.html** - HTML page that will contain description of your skin (some links, donate button or similar is possible)

Finally, place this .zip file into the root directory described above!

Supported Views

Containers

- LinearLayout
  - android:orientation
  - android:gravity
- RelativeLayout
- ScrollView
- TableLayout
- TableRow

Views

- TextView extends View
  - android:text - this contain ACTIONS below
  - android:textSize
  - android:textColor
  - android:gravity - full support on all values, also like “center_vertical|right”
  - android:lines
  - android:ellipsize
  - android:shadowColor - and optional android:shadowDx, android:shadowDy and android:shadowRadius
  - locus:textFont
  - locus:textFormat

- Button extends TextView
  - locus:actionClick
  - locus:actionVisibility

- ImageView extends View
  - android:src
- ImageButton extends ImageView
  - locus:actionClick - as Button
  - locus:actionVisibility - as Button
- MapScreenView (custom locus view) extends View
  - no custom parametres
- RotateView (custom locus view) extends View
  - locus:action - attach to which rotation action… for example locus:action="{orient_course}"
  - locus:rotateImage - reference to drawable, actually centered and rotated around center by angle defined by action
  - locus:rotatePivotX - move rotation point by X value (+X to bottom)
  - locus:rotatePivotY - move rotation point by Y value (+Y to bottom)
  - locus:rotateStartAngle - angle at which rotation starts with rotateStartValue value (- value for counter-clockwise orientation)
  - locus:rotateStartValue - value for variable that starts at rotateStartAngle (in base units, so metres, second, ..)
  - locus:rotateEndAngle - angle at which rotation ends with rotateEndValue value
  - locus:rotateEndValue - value for variable that ends at rotateEndValue
- **SlideView** (custom Locus view) extends View
  - locus:action - attaches to which slide action … for example locus:action="{orient_course}"
  - locus:slideImage - reference to a drawable item used for slide
  - locus:slideStartPosition - position of image pixel that matches *slideStartValue* value.
  - locus:slideStartValue - action value for *slideStartPosition* (in base units - metres, seconds, ..)
  - locus:slideEndPosition - position value of the last image pixel. Also (*slideEndPosition* - *slideStartPosition*) define width of image
  - locus:slideEndValue - action value for *slideEndPosition*
  - locus:slideInfinite - if slide image is repeated to cover whole parent view (true/false) [false]

Sample - let's imagine this image of compass:

To make it work in full screen width you need to define these parameters:

```
android:layout_width="match_parent"
android:layout_height="X"
locus:action="{orient_course}"
locus:slideImage="X"
locus:slideStartPosition="0dip"
locus:slideStartValue="180"
locus:slideEndPosition="match_parent"
```

```
    locus:slideEndValue="-180"
    locus:slideInfinite="[true]"
```

By this definition you say that:

- 1. if orientation angle is 180°, the first pixel (slideStartPosition) will be drawn really as the first pixel. Because the defined whole range is from 0° - 360° and end value has position on the end (*locus:slideEndPosition="match_parent"*), image will be stretched over whole screen. Middle of image (**S**), will be in the middle, as we want!
- 2. if orientation is for example 90°, Locus Map firstly computes percent part of image that should be moved. It's computed by **(current value - slideStartValue) / (slideEndValue - slideStartValue)**, so in this case it's (90 - 180) / (-180 - 180) = **25%**!. This means that image will be moved by 25%

Global attributes

On all Views can be applied these attributes:

- android:id
- android:layout_width
- android:layout_height
- android:layout_weight - if parent view is LinearLayout
- RelativeLayout.LayoutParams - if parent view is RelativeLayout
- android:layout_margin
- android:layout_marginLeft
- android:layout_marginRight
- android:layout_marginTop
- android:layout_marginBottom
- android:padding
- android:paddingLeft
- android:paddingRight
- android:paddingTop
- android:paddingBottom
- android:background (image, color in RGB, ARGB)

Custom attributes

locus:actionClick

- *{track_record_start}*
- *{track_record_stop}*
- *{track_record_pause}*
- *{map_zoom_in}*
- *{map_zoom_out}*
- *{map_center}*

locus:actionVisibility

- *{lat_gps}* - GPS latitude (map center latitude if GPS off)
- *{lon_gps}* - GPS longitude (map center longitude if GPS off)
- *{altitude}* - GPS altitude (0 if GPS off)
- *{accuracy}* - GPS accuracy (0 if GPS off)
- *{gps_sats_used}* - actual used satellites for GPS fix
- *{gps_sats_all}* - all visible satellites
- *{declination}* - actual declination
- *{orient_course}* - course orientation (source depend on selection on GPS screen - GPS or internal compass)
- *{orient_course_opposit}* - course orientation (source depend on selection on GPS screen - GPS or internal compass)
- *{orient_pitch}* - pitch orientation
- *{orient_roll}* - roll orientation
- *{orient_gps_shift}* - orientation computed as bearing from previous to current GPS location
- *{orient_gps_angle}* - angle computed as {orient_gps_shift} - {orient_course}, so it's true moving direction.

- *{time}* - actual time
- *{speed}* - GPS speed (0 if GPS off)

- *{rec_dist}* - track recording - recorded distance
- *{rec_dist_down}* - track recording - downhill distance
- *{rec_dist_up}* - track recording - uphill distance
- *{rec_alt_min}* - track recording - minimum altitude
- *{rec_alt_max}* - track recording - maximum altitude
- *{rec_alt_down}* - track recording - downhill altitude
- *{rec_alt_up}* - track recording - uphill altitude
- *{rec_alt_cumu}* - track recording - cumulated altitude
- *{rec_time}* - track recording - total record time
- *{rec_time_move}* - track recording - time only when move
- *{rec_speed_avg}* - track recording - speed average
- *{rec_speed_avg_move}* - track recording - speed average only when move
- *{rec_speed_max}* - track recording - speed maximum
- *{rec_points}* - track recording - num of recorded points
- *{rec_pace}* - track recording - num of recorded points

- *{map_center_lat}* - map center latitude
- *{map_center_lon}* - map center longitude
- *{map_rotate}* - map rotate value

- *{guide_wpt_name}* - guide target waypoint name
- *{guide_wpt_lat}* - guide target waypoint latitude
- *{guide_wpt_lon}* - guide target waypoint longitude
- *{guide_wpt_alt}* - guide target waypoint altitude
- *{guide_wpt_dist}* - guide target distance to waypoint
- *{guide_wpt_dist_to_finish}* - guide target distance to last waypoint of track
- *{guide_wpt_azim}* - guide target azimuth to waypoint
- *{guide_wpt_angle}* - guide target angle value (computed as {guide_wpt_azim} - {orient_course})
- *{guide_wpt_time}* - guide target time to waypoint

- *{guide_wpt_time_to_finish}* - guide target time to last waypoint of track

locus:textFont

- link to font stored in MySkin/assets directory

locus:textFormat

- parametres should be {i}, {d}, {i.d}, {u} “integer part, double part, both, only units”
- for example - *“locus:textFormat=”{i.d} - {u}“*. Can be applied to: speed, distance, altitude and accuracy now