Skin XML schema documentation version 2.0

Seeing samples is a good way how to understand PocketGamepad xml file. All skins are possible to export from application. Open skins (Menu/Skins), long tap desired item and tap Export. Exported file has extension pgpad, which is zip file. Just rename pgpad to zip.

Element: PocketGamepadSkin

PocketGamepadSkin is a root element

XML instance representation:

```
<PocketGamepadSkin>
    <GeneralInfo>…</GeneralInfo> [1]
    <Colors>…</Colors> [0..1]
    <Screen>…</Screen> [1..*]
</PocketGamepadSkin>
```

Element: GeneralInfo

GeneralInfo is an element, which represents basic information about the skin.

`minXMLParserVersion`	Minimum Pocket Gamepad parser version. If parser has lower value, skin must not work. Using the last version of Pocket Gamepad guarantees the last version of XML parser.
`versionCode`	Version of skin, number from 1 till … When it is created new version of skin, this number must be increased
`versionName`	Version of skin which is displayed to users
`name`	Name of the skin
`description`	Description (long name)
`controlling`	Notes about controlling the game

Each skin can be designed for several screen aspect ratios (minimum is one). Application will find and use the closest possible (depend of device screen aspect ratio). For each defined `DesignedScreen` element you must also define the `Screen` element (parent `PocketGamepadSkin`) with the same aspect ratio.

XML instance representation:

```
<GeneralInfo
    minXMLParserVersion="integer" [1]
    versionCode="integer" [1]
    versionName="string" [1]
    name="string" [1]
    description="string" [0..1]
    controlling="string" [0..1]
    <Author>…</Author> [0..*]
    <Game>…</Game> [0..*]
    <DesignedScreen>…</DesignedScreen> [1..*]
</GeneralInfo>
```

Element: Author

Information about author of the skin.

`name`	Name of the author
`www`	Web page. Must begin with http:// or https://
`email`	Author's email address

XML instance representation:

```
<Author
    name="string" [1]
    www="string" [0..1]
    email="string" [0..1]
</Author>
```

Element: Game

Information about the game which is skin designed for.

`name`	Name of the game
`www`	Web page of the game. Must begin with http:// or https://

XML instance representation:

```
<Game
    name="string" [1]
    www="string" [0..1]
</Game>
```

Element: DesignedScreen

Information about the screen aspect ratio. For each `DesignedScreen` element must be created `Screen` element (parent `PocketGamepadSkin`).

`aspectRatio`	Can be: 4:3, 3:2, 8:5, 5:3 or 16:9

XML instance representation:

```
<DesignedScreen
    aspectRatio="16:9" [1]
</DesignedScreen>
```

Element: Colors

Each color used for skin must be defined in this element.

XML instance representation:

```
<Colors>
    <ColorDefinition>…</ColorDefinition> [1..*]
<Colors/>
```

Element: ColorDefinition

Color represents by Red, Green, Blue and Alpha channel. Each color must have two attributes:
1. name
2. rgba_int or rgba_float

`Name`	Unique name of the color
`rgba_int`	"red_int green_int blue_int alpha_int" Each int can be [0..255] Sample for dark green color "0 128 0 255"
`rgba_float`	"red_float green_float blue_float alpha_float" Each float can be [0..1] Sample for crimson color "0.86 0.08 0.22 1.0"

XML instance representation:

```
<ColorDefinition
    name="string" [1]
    rgba_int="string" or rgba_float="string" [1]
</ColorDefinition>
```

Element: Screen

This element contains all information for displaying the skin. Application will choose only one `Screen` (the selector is `aspectRatio`) if xml contains more `Screen`s.

`aspectRatio`	It indicates for which aspect ratio is element `Screen` designed. Can be: 4:3, 3:2, 8:5, 5:3 or 16:9. Every file can contain elements `Screen` for each aspect ratio. One aspect ratio can be used only once at the file. If the device aspect ratio is not defined, application will use the closest.
`designedWidth`	Width in pixels
`designedHeight`	Height in pixels

All `Screen` elements use the coordinate system, where:
Left bottom pixel has coordinates 0,0
Right top pixel has coordinates `designedWidth`-1, `designedHeight`-1

Sample:
Screen1: `aspectRatio="16:9"` `designedWidth="1920"` `designedHeight="1080"`
Point A: 960,540

Screen2: `aspectRatio="16:9"` `designedWidth="640"` `designedHeight="360"`
Point B: 320,270

Point A from screen 1 will be displayed at the same position like Point B from the screen 2.

If `designedWidth` > `designedHeight` it is landscape mode
If `designedWidth` < `designedHeight` it is portrait mode

XML instance representation:

```
<Screen
    aspectRatio="string" [1]
    designedWidth="string" [1]
    designedHeight="string" [1]
    <Style>…</Style> [0..*]
    <Tab>…</Tab> [1..*]
</Screen>
```

Element: Style

Application enables creating and using styles (design of elements which could be displayed). Any style can inherit style from another (parent) style. Parent style must be defined before child style. If your style inherit from another (parent) style, you can use all parent attribute(s), add new attribute(s) or redefine some of parent attribute(s).

Naming convention: When some style inherits form another, its name is parent + "." + xxx.

Example of style name:

```
<Style name="Button" ...
<Style name="Button.4corners" parent="Button"  ...
<Style name="Button.4corners.Arrow" parent="Button.4corners" ...
```

Each style must contain at least one `Appearance` element. Only one `Appearance` element is usually used for non-interactive elements and its `state="idle"`. If you want to create style for interactive elements like buttons (`Polygon` or `Circle`), you must add 2 `Appearance` elements. First with `state="idle"` and the second with `state="pressed"`

`name`	Style name
`parent`	Parent name (it must be defined before)

XML instance representation:

```
<Style
    name="string" [1]
    parent="string" [0..1]
    <Appearance>…</Appearance> [1..2]
</Style>
```

Element: Appearance

Appearance of the displayed element

`state`	Can be *`"idle"`* or *`"pressed"`* *`"idle"`* is used for displaying not pressed object state *`"pressed"`* is used for displaying pressed object state

XML instance representation:

```
<Appearance
    state="string" [1]
    <Shape>…</Shape> [0..1]
    <Texture>…</ Texture> [0..1]
</Appearance>
```

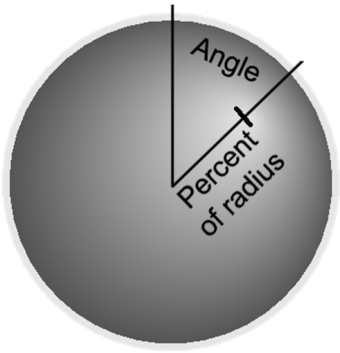
There must be defined `Shape` or `Texture` or both.

Element: Shape

Definition of the shape

`cornerRadiusList`	In case using `Shape` for polygon object there is defined radiuses of the polygon corners. If polygon has more corners than is defined here, it will be repeated. If you want to have the same all corner radiuses, `cornerRadiusList` must have only one value. Example:
	`cornerRadiusList="10"`
	In case of more corner values, values must be split by space.
	`cornerRadiusList="10 20 20 10"`

`color`	Solid color of the object. Name of color defined in `Colors`
`colorList`	In case using `Shape` for polygon object, it is possible to define color for each corner and display color gradient. There must be defined color for each corner. Example for triangle: `colorList="buttonDark buttonDark buttonlight"` In case using `Shape` for circle object there must be defined two colors. Example: `colorList="buttonLight buttonDark"`. At this case there must be defined also `angle` and `percentOfRadius`. Look bellow. The first color is the point defined by `angle` and `percentOfRadius`. The second color is the circle border.
`angle`	[0..359]
`percentOfRadius`	[0..100]



XML instance representation:

```
<Shape
    cornerRadiusList ="int list" [0..1]
    color="string" [0..1]
    colorList="string list" [0..1]
    angle="string" [0..1]
    percentOfRadius ="int" [0..1]
    <Stroke>…</Stroke> [0..1]
</Shape>
```

Element: Stroke

Definition of the stroke

`Width`	Width of the stroke
`Color`	Solid color of the stroke. Name of color defined in `Colors`

XML instance representation:

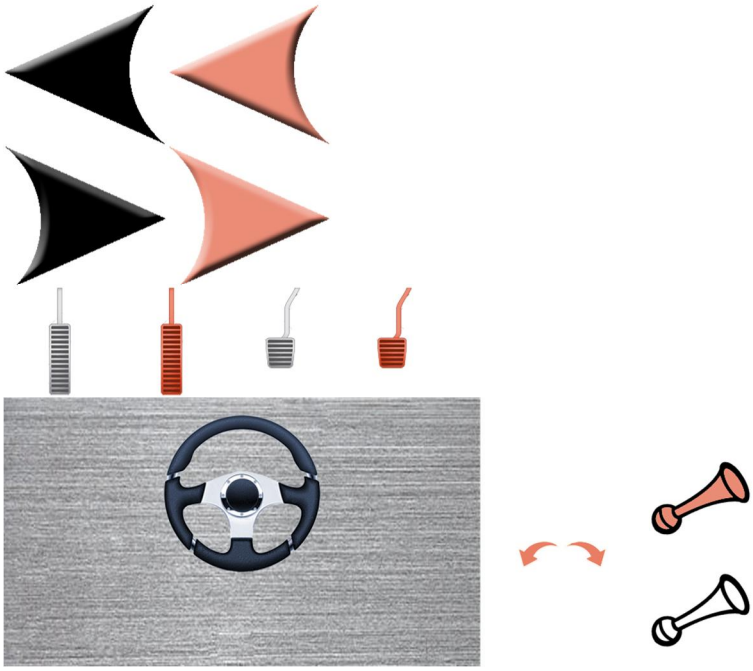
```
<Stroke
    width="int" [1]
    color="string" [1]
</Stroke>
```

Element: Texture

Definition of the object texture.

Textures must be saved in png files. Width and height of the png pictures must be power of 2 (2, 4, 8, 16, 32, 64, 128, 512, 1024, etc.). Texture files must be saved at the same folder as xml file. Textures

uses the alpha channel.



Texture example, size 1024x1024 pixels

Displaying the horn:

```
<Appearance state="idle">
    <Texture fileName="CrashDrive.png" wrap="fitToCenter" left="860" right="1010" bottom="0" top="150"/>
</Appearance>
<Appearance state="pressed">
    <Texture fileName="CrashDrive.png" wrap="fitToCenter" left="860" right="1010" bottom="150" top="300"/>
</Appearance>
```

`filename`	Name of source file name. If you use inbuilt texture (for example for displaying system buttons) the name of the texture consists of @ and texture name without extension (example `filename="@buttons_texture"`).
`Wrap`	*`coverBoundaryRect`* – texture fills whole object boundary rect. Aspect ratio of the texture is not maintained. *`fitToCenter`* – texture fills object boundary rect as big as possible and maintain the texture aspect ratio.
`Left`	Left coordinate of the texture file
`Right`	Right coordinate of the texture file
`Bottom`	Bottom coordinate of the texture file
`Top`	Top coordinate of the texture file
`Padding`	Set all paddings from object boundary rect. Bigger padding value, texture is smaller and vice versa.
`paddingLeft`	Padding left, it rewrites `padding`
`paddingRight`	Padding right, it rewrites `padding`
`paddingBottom`	Padding bottom, it rewrites `padding`
`paddingTop`	Padding top, it rewrites `padding`

XML instance representation:

```
<Texture
    fileName="string" [1]
    wrap="string" [1]
    left="int" [1]
    right="int" [1]
    bottom="int" [1]
    top="int" [1]
    padding="int" [0..1] />
    paddingLeft="int" [0..1] />
    paddingRight="int" [0..1] />
    paddingBottom="int" [0..1] />
    paddingTop="int" [0..1] />
</Texture>
```

Element: Tab

Each screen can have one or more tabs. It is possible to switch among the tabs by `jumpTo="tab_name"` command. All objects which are displayed at the device screen are defined inside `Tab`.

If you want to the tab sends joystick inputs based on the device attitude, you must define `ControlAxis` element inside `Tab`. In case of multi `Tab` in one `Screen`, you must define `ControlAxis` in all `Tab`s.

You can also define more axes for one attitude change (pitch or roll). Typical using it is for car racing game where you use pitch for braking and accelerating.
`minUsablePos="0"`, `maxUsablePos="1"` for braking
`minUsablePos="-1"`, `maxUsablePos="0"` for accelerating
For more information see explanation of `ControlAxis` element.

`Name`	Name of the tab (must be unique inside the `Screen` element)
`backgroundColor`	Background color name (defined in `Colors`)
`numLock`	If defined, application switch-on/off the NumLock. If not defined, application do not change NumLock. Possible values: *`on`* or *`off`*
`sendingWithoutConnection`	If *`true`*, Tab can send UDP commands without establishing connection. Default value is *`false`*.

XML instance representation:

```
<Tab
    name="string" [1]
    backgroundColor="string" [1]
    numLock="string" {“on“, “off“} [0..1]
    sendingWithoutConnection ="boolean" {“true“, “false“} [0..1]
    <ControlAxis>…</ControlAxis> [0..*]
    <Polygon>…</Polygon> [0..*]
    <Circle>…</Circle> [0..*]
    <SeekBar>…</SeekBar> [0..*]
    <AttitudeIndicator>…</AttitudeIndicator> [0..*]
</Tab>
```

Element: ControlAxis

`ControlAxis` element is used for defining and simulating joystick inputs. For each axis must be defined one `ControlAxis` element.

`Name`	Name of the axis. Can be *`1`*, *`2`*, *`3`*, *`4`*, *`5`*, *`6`*, *`7`* or *`8`*. Axis called *`1`* is usually used at most games for roll inputs Axis called *`2`* is usually used at most games for pitch inputs
`Attitude`	*`pitch`* for pitch inputs *`roll`* for roll inputs No default value
`minUsablePos`	Max possible deflection of each device is defined in range [-1..1]. You can use only part of this range defined by `minUsablePos` and `maxUsablePos`. Roll left is positive [1…0] Roll right is negative [0..-1] Pitch back, to you, pulling is positive [1…0] Pitch front, from you (pushing) is negative [0..-1] Default values = *`-1.0`*
`maxUsablePos`	Described above. Default values = *`1.0`*
`maxLeftValue` `maxPitchBackValue`	If the device reaches max left or back pitch deflection defined by `minUsablePos` and `maxUsablePos` the application sends this value. Default value = 0
`maxRightValue` `maxPitchFrontValue`	If the device reaches max right or front pitch deflection defined by `minUsablePos` and `maxUsablePos` the application sends this value. Default value = 32767

XML instance representation for roll inputs:

```
<ControlAxis
    name="string" [1]
    minUsablePos="float" [0..1]
    maxUsablePos="float" [0..1]
    maxLeftValue="int" [0..1]
    maxRightValue="int" [0..1]
    attitude="string" [1]
</ControlAxis>
```

XML instance representation for pitch inputs:

```
<ControlAxis
    name="string" [1]
    minUsablePos="float" [0..1]
    maxUsablePos="float" [0..1]
    maxPitchBackValue="int" [0..1]
    maxPitchFrontValue="int" [0..1]
    attitude="pitch" [1]
</ControlAxis>
```

Element: Polygon

This element is used for displaying polygons, typically buttons.
- The look of polygon is defined at the `Appearance` element
- If the polygon is active (`clickable="true"` or there are defined `Attitude` element) you must define at least one but usually two `Action` elements. The first what to do when polygon is clicked: `type="actionDown"` and when polygon is released: `type="actionUp"`.
It is also necessary to define two `Appearance` elements. The first one with `state="idle"` and the second one with `state="pressed"`.
- If you want to element be attitude sensitive, you must define the `Attitude` element.

`tag`	Must be unique in the `Tab` element.
`style`	Style of polygon. Every items of style can be rewritten in the `Appearance` element
`coordinates`	Coordinates of the polygon. Points must be split by space. Example: `coordinates="1200,350 1850,350 1850,600 1200,600"`
`tessellate`	*true* or *false* If the polygon is concave, it must be set to *true*. Default value = *false*
`rectTouchTest`	*true* or *false* If polygon is a rectangle, you can set this attribute to *true*. It can save some energy. Default value = *false*
`clickable`	*true* or *false* If the polygon is a touchable button `clickable="true"` Default value = *false*
`visibility`	*true* or *false* For hiding this object `visibility="false"` Default value = *true*

XML instance representation:

```
<Polygon
    tag="string" [1]
    style="string" [0..1]
    coordinates="list of points" [1]
    tessellate="true/false" [0..1]
    rectTouchTest="true/false" [0..1]
    clickable="true/false" [0..1]
    visibility="true/false" [0..1]
    <Action>…</Action> [0..2]
    <Appearance>…</Appearance> [0..2]
    <Attitude>…</Attitude> [0..1]
</Polygon>
```

Element: Circle

This element is used for displaying circles, ellipses or their sectors.
- The look of polygon is defined at the `Appearance` element
- If the polygon is active (clickable or attitude active) you must define at least one but usually two `Action` elements. The first what to do when polygon is clicked: `type="actionDown"` and when polygon is released: `type="actionUp"`.

It is also necessary to define two `Appearance` elements. The first one with `state="idle"` and the second one with `state="pressed"`.
- If you want to element be attitude sensitive, you must define the `Attitude` element.

This element can di displayed like circle, if there is defined `radius` or like ellipse if there are defined `radiusX` and `radiusY`.

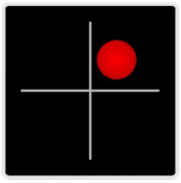
If you want to display only sector you must define `startAngle` and `endAngle`.

`tag`	Must be unique in the `Tab` element.
`style`	Style of polygon. Every items of style can be rewritten in the `Appearance` element
`center`	`Circle (ellipse) center` `center="600,500"`
`radius`	`Radius of the circle`
`radiusX`	`Radius for the X axis of the ellipse`
`radiusY`	`Radius for the Y axis of the ellipse`
`startAngle`	In case of sector, start angle for drawing
`endAngle`	In case of sector, start end for drawing
`Clickable`	*true* or *false* If the polygon is a touchable button `clickable ="true"` Default value = *false*
`Visibility`	*true* or *false* For hiding this object `visibility="false"` Default value = *true*

XML instance representation:

```
<Circle
    tag="string" [1]
    style="string" [0..1]
    center="point" [1]
    radius="int" [0..1]
    radiusX="int" [0..1]
    radiusY="int" [0..1]
    startAngle="float" [0..1]
    endAngle="float" [0..1]
    clickable="true/false" [0..1]
    visibility="true/false" [0..1]
    <Action>…</Action> [0..2]
    <Appearance>…</Appearance> [0..2]
    <Attitude>…</Attitude> [0..1]
</Circle>
```

Element: AttitudeIndicator



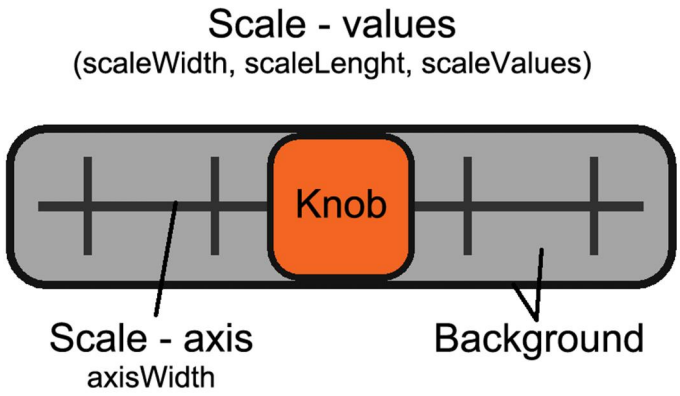
This is a passive object, which shows the device attitude. The dimensions are defined by `left`, `right`, `bottom` and `top` position.
Object contains 3 objects: `Background`, `Knob` and `Axis`.
If you want to `Knob` stay inside the `Background`, you must define `Knob`'s `padding`.
`Knob`'s center must have coordinates 0,0. Its displaying is relative to the `AttitudeIndicator` center.

tag	Must be unique in the Tab element.
left	Left position
Right	Right position
bottom	Bottom position
top	Top position

XML instance representation:

```
<AttitudeIndicator
    tag="string" [1]
    left="int" [1]
    right="int" [1]
    bottom="int" [1]
    top="int" [1]
    <Background>…</Background> [0..1]
    <Knob>…</Knob> [1]
    <Axis>…</Axis> [0..1]
</AttitudeIndicator>
```

Element: SeekBar



Seek bar is an active object. You can use it for setting joystick axis – joystick based games. Typically usage is for engine throttle, wing flaps, aircraft ruder etc. Orientation can be `vertical` or `horizontal`.

The dimensions are defined by `left`, `right`, `bottom` and `top` position.

Object consists from 3 objects: `Background`, `Knob` and `Axis`.

If you want to `Knob` stay inside the `Background`, you must define `Knob`'s `padding`. `Knob`'s center must have coordinates 0,0. Its displaying is relative to the `SeekBar` center.

Max deflection values of the `SeekBar` are:
left: -1, right: +1
bottom: -1, top: +1

tag	Must be unique in the Tab element.
left	Left position
Right	Right position
Bottom	Bottom position
Top	Top position
Clickable	*true* or *false* Default value = *true*
orientation	*vertical* or *horizontal* Default value = *vertical*
snapToScale	*true* or *false*

	If true *true* the Knob can stay only at the predefined positions at the Scale element. Default value = *false*
defKnobPos	This is a default Knob position when skin launched. Value can be in the range [-1..1] Default value = *0* (center)
returnToDefaultSpeed	At this version only *0* and *1* is allowed. *1* means, when user release the Knob, Knob returns to the defKnobPos. Default value = *0* (no returning when releasing the Knob)

XML instance representation:

```
<SeekBar
    tag="string" [1]
    left="int" [1]
    right="int" [1]
    bottom="int" [1]
    top="int" [1]
    clickable="true/false" [0..1]
    orientation="vertical/horizontal" [0..1]
    snapToScale="true/false" [0..1]
    defKnobPos="float" [0..1]
    <Background>…</Background> [0..1]
    <Knob>…</Knob> [1]
    <Scale>…</Scale> [0..1] Scale must be the last element.
</SeekBar>
```

Element: TouchPad

This is a rectangle object, which can be used like laptop touch pad for the PC mouse movement. This object contains only one object: Background

tag	Must be unique in the Tab element.
left	Left position
Right	Right position
bottom	Bottom position
top	Top position
clickable	*true* or *false* Default value = *true*
mmToPixels	One millimeter on touch pad = mmToPixels pixels on PC screen Default value is 8
rotation	In case that x axis is not from left to right side and y axis is not from bottom to top use this. Possible value (clockwise degrees) can be: *90*, *180*, *270*

XML instance representation:

```
<TouchPad
    tag="string" [1]
    clickable="true/false" [0..1]
    left="int" [1]
    right="int" [1]
    bottom="int" [1]
    top="int" [1]
```

```
    mmToPixels="float" [0..1]
    rotation="90,180,270" [1]
    <Background>…</Background> [1]
</TouchPad>
```

Element: PointingStick

This is a circle object, which can be used like laptop pointing stick (track point) for the PC mouse movement. This object contains only one object: `Background`

`tag`	Must be unique in the `Tab` element.
`center`	Center of the `PointingStick`, `center="600,500"`
`radius`	Right position
`clickable`	*`true`* or *`false`* Default value = *`true`*
`mmToPixels`	One millimeter on moving from touch down position = `mmToPixels` pixels on PC screen each 20 ms. Default value is 1
`rotation`	In case that x axis is not from left to right side and y axis is not from bottom to top use this. Possible value (clockwise degrees) can be: *`90`*, *`180`*, *`270`*

XML instance representation:

```
<PointingStick
    tag="string" [1]
    clickable="true/false" [0..1]
    center="point" [1]
    radius="int" [1]
    mmToPixels="float" [0..1]
    rotation="90,180,270" [1]
    <Background>…</Background> [1]
</PointingStick>
```

Element: MouseScrollWheel

This is a rectangle object, which can be used like mouse scroll whell. This object contains only one object: `Background`

`tag`	Must be unique in the `Tab` element.
`left`	Left position
`right`	Right position
`bottom`	Bottom position
`top`	Top position
`clickable`	*`true`* or *`false`* Default value = *`true`*
`mmToPoints`	One millimeter on touch pad = `mmToPoints` points Default value is 50
`rotation`	In case that x axis is not from left to right side and y axis is not from bottom to top use this. Possible value (clockwise degrees) can be: *`90`*, *`180`*, *`270`*
`stepX`	If you want to use scroll in X axis, it must be greater than zero. If there is for example `stepX="3"`, application can send scroll X values: 3, 6, 9 etc.
`stepY`	If you want to use scroll in Y axis, it must be greater than zero.

XML instance representation:

```
<MouseScrollWheel
    tag="string" [1]
    clickable="true/false" [0..1]
    left="int" [1]
    right="int" [1]
    bottom="int" [1]
    top="int" [1]
    mmToPoints="float" [0..1]
    rotation="90,180,270" [1]
    stepX="int" [0..1]
    stepY="int" [0..1]
    <Background>…</Background> [1]
</MouseScrollWheel>
```

Element: HatSwitch

This is an object, which replaces joystick hat switch (POV).
Element contains 3 elements: `Background`, `DeadZone`, `Knob`. All elements can contain only element `Circle`. They cannot contain element `Polygon`.
Radius of the elements `Background` and `DeadZone` are defined in `HatSwitch` attributes.

tag	Must be unique in the `Tab` element.
clickable	*true* or *false* Default value = *true*
Name	Name of the hat switch. Can be *1*, *2*, *3*, and *4*
center	Circle (ellipse) center center="600,500"
radius	Radius of the circle
radiusX	Radius for the X axis of the ellipse
radiusY	Radius for the Y axis of the ellipse
deadZoneRadius	Radius of the dead zone If user keep finger in this area, hat switch is in its neutral position
deadZoneRadiusX	Radius for the X axis of the dead zone ellipse
deadZoneRadiusY	Radius for the Y axis of the dead zone ellipse
type	*4directions* or *8directions*. *4directions*

```
<HatSwitch
    tag="string" [1]
    clickable="true/false" [0..1]
    name="1/2/3/4" [1]
    center="point" [1]
    radius="int" [0..1]
    radiusX="int" [0..1]
    radiusY="int" [0..1]
    deadZoneRadius="int" [0..1]
    deadZoneRadiusX="int" [0..1]
    deadZoneRadiusY="int" [0..1]
    <Background>…</Background> [1]
    <DeadZone>…</DeadZone> [1]
    <Knob>…</Knob> [1]
</HatSwitch>
```

Sample:

```
<HatSwitch
    tag="hat_switch_1"
    center="463,630"
    radius="182"
    deadZoneRadius="45"
    name="1"
    type="8directions"
    clickable="true" >
    <Background>
        <Circle style="buttonStyle" />
    </Background>
    <DeadZone>
        <Circle style="buttonStyle " />
    </DeadZone>
    <Knob padding="35" >
        <Circle style="buttonStyle.seekBarButton" radius="30" />
    </Knob>
</HatSwitch>
```

Element: PingIndicator

This is an object, which shows response between smart phone and PC server. Quality of connection is displayed like a bar, which can has green, yellow or red color. Size of the bar is also depended of the connection quality.
Element contains element `Bar` and can contain object `Background`.

`Tag`	Must be unique in the `Tab` element.
`Left`	Left position
`Right`	Right position
`Bottom`	Bottom position
`Top`	Top position
`clickable`	*`true`* or *`false`* Default value = *`true`*
`orienatation`	*`horizontal`* or *`vertical`* Default value = *`vertical`*. In case of vertical, bar changes its height (depended of the connection quality). In case of horizontal, bar changes its width.

XML instance representation:

```
<PingIndicator
    tag="string" [1]
    clickable="true/false" [0..1]
    left="int" [1]
    right="int" [1]
    bottom="int" [1]
    top="int" [1]
    orientation="horizontal/vertical" [1]
    <Background>…</Background> [0..1]
    <Bar>…</Bar> [1]
</PingIndicator>
```

Element: Bar

This element is used for `PingIndicator`. It has only `padding` attributes.

`Padding`	Set all paddings from object boundary rect. Bigger padding value, texture is smaller and vice versa.
`paddingLeft`	Padding left, it rewrites `padding`
`paddingRight`	Padding right, it rewrites `padding`
`paddingBottom`	Padding bottom, it rewrites `padding`
`paddingTop`	Padding top, it rewrites `padding`

XML instance representation:

```
<Bar
    padding="int" [0..1] />
    paddingLeft="int" [0..1] />
    paddingRight="int" [0..1] />
    paddingBottom="int" [0..1] />
    paddingTop="int" [0..1] />
</Bar>
```

Element: Background

This element is used for `AttitudeIndicator`, `SeekBar` and `TouchPad`.
It has not any attributes, only `Polygon`.
XML instance representation:

```
<Background>
    <Polygon>…</Polygon> [1]
</Background>
```

Element: Knob

This element is used for `AttitudeIndicator` and `SeekBar`.

The knob is displayed relative to `AttitudeIndicator` and `SeekBar`, so its center should have coordinates 0,0. If you want to `Knob` stay inside the `Background`, you must define `Knob`'s `padding`.

The `Knob` element can contain `Polygon` or `Circle` element.

`padding`	Set all paddings from object boundary rect. Bigger padding value, texture is smaller and vice versa.
`paddingLeft`	Padding left, it rewrites `padding`
`paddingRight`	Padding right, it rewrites `padding`
`paddingBottom`	Padding bottom, it rewrites `padding`
`paddingTop`	Padding top, it rewrites `padding`

XML instance representation:

```
<Knob
    padding="int" [0..1]
    paddingLeft="int" [0..1]
```

```
    paddingRight="int" [0..1]
    paddingBottom="int" [0..1]
    paddingTop="int" [0..1]
    <Polygon>…</Polygon> or <Circle>…</Circle> [1]
</Knob>
```

Element: Axis

This element is used for `AttitudeIndicator`.

The `Polygon` inside of the `Axis` must have `tessellate="true"` and must not have filled `coordinates`. Coordinates of the `Polygon` will be calculated (based on `AttitudeIndicator` dimensions, `padding` and `width`).

Example:

```
<Axis padding="40" width="5" >
    <Polygon style="buttonStyle.axis" tessellate="true" >
    </Polygon>
</Axis>
```

`padding`	Set all paddings from parent object boundary rect.
`paddingLeft`	Padding left, it rewrites `padding`
`paddingRight`	Padding right, it rewrites `padding`
`paddingBottom`	Padding bottom, it rewrites `padding`
`paddingTop`	Padding top, it rewrites `padding`
`width`	Width of axis

XML instance representation:

```
<Axis
    padding="int" [0..1]
    paddingLeft="int" [0..1]
                                                    paddingRight="int" [0..1]
    paddingBottom="int" [0..1]
    paddingTop="int" [0..1]
    width="int" [1]
    <Polygon tessellate="true">…</Polygon> [1]
</Axis>
```

Element: Scale

This element is used for `SeekBar`.

The `Polygon` inside of the `Scale` must have `tessellate="true"` and must not have filled `coordinates`. Coordinates of the `Polygon` will be calculated (based on `SeekBar` dimensions, `padding` and `width`, `scaleWidth`, `scaleLenght`, `scaleValues` and `Knob padding`).

`padding`	Set all paddings from parent object boundary rect.
`paddingLeft`	Padding left, it rewrites `padding`
`paddingRight`	Padding right, it rewrites `padding`
`paddingBottom`	Padding bottom, it rewrites `padding`
`paddingTop`	Padding top, it rewrites `padding`

axisWidth	Width of the axis
scaleWidth	Width of the scale
scaleLength	Length of the scale
scaleValues	List of values. Each value must be inside this interval [-1..1] and their order must be from smallest to biggest. scaleValues="-1,-0.5,0,0.50,1"

XML instance representation:

```
<Scale
    padding="int" [0..1]
    paddingLeft="int" [0..1]
    paddingRight="int" [0..1]
    paddingBottom="int" [0..1]
    paddingTop="int" [0..1]
    axisWidth="int" [1]
    scaleWidth="int" [0..1]
    scaleLength="int" [0..1]
    scaleValues="list of int" [0..1]
    <Polygon tessellate="true">…</Polygon> [1]
</Scale>
```

Element: Action

This element is used for making an action, when object is pressed (tapped) or when object is attitude sensitive and the attitude makes this object pressed.

Action element has only one attribute -type. The type. Can be "pressedAction" or "releasedAction"
- When object is pressed, application runs command(s) encapsulated in Action element where type="pressedAction"
- When object is released, application runs command(s) encapsulated in Action element where type="releasedAction"

Example of menu button Action:

You can see, there are only vibration and play sound at the pressedAction Action.
After releasing the button application runs releasedAction Action. In this Action there is the systemCommand="system_menu for displaying options menu.

```
<Action type="pressedAction">
    <Command vibrateTime="50" soundFileName="button_click.wav" />
</Action>
<Action type="releasedAction">
    <Command systemCommand="system_menu" />
</Action>
```

Example of left arrow button Action:

When the button is pressed, application sends to the PC information, that left arrow is pressed and keep it pressed. When button is released application sends at this moment to the PC information that left arrow is released.

This is typically example for Action which is used for game controlling.
When using keyDown do not forget to use keyUp too.

```
<Action type="pressedAction">
    <Command keyDown="vk_arrow_left" />
</Action>
<Action type="releasedAction">
    <Command keyUp="vk_arrow_left" />
</Action>
```

Each Action element can contain one or more Command elements.

type	Can be *"pressedAction"* or *"releasedAction"*

XML instance representation:

```
<Action
    type="string" [1]
    <Polygon>…</Polygon> [1..*]
</Action>
```

Element: Command

Command element is used for the execution of the commands and it is encapsulated at the Action element.

keyDown	Press a key. Key codes are bellow
keyUp	Release a key. Key codes are bellow
systemCommand	Execute a system command. Use this command at the Action, where type="releasedAction" *"system_menu"* displaying the options menu *"system_exit"* close application *"system_connect"* connect to the PC *"system_calibrate_accelerometer"* calibrate accelerometer *"system_attitude_start_stop"* keeping device attitude at the neutral position (switch – start/stop)
jumpTo	If Screen contains more Tabs, you can switch among them by this command: jumpTo="*tab_name*", where *tab_name* is the name of the Tab which will be displayed.
soundFileName	File name of the mp3 or wav file. File must be located at the same directory as skins.xml file.
vibrateTime	Time in milliseconds
pulseControl	Can be only "" or *"constantAll"*. In case of pulseControl="constantAll", the application sends pulses keyUp and keyDown until user releases the object. The length of the pulses is interval value. Default value = ""
interval	Time in milliseconds for pulseControl="constantAll" Default value = *"100"*
link	Open link in default browser. link="*http://www.funair.cz*"

Example for machine gun, fire trigger is the space bar:

```
<Action type="pressedAction">
    <Command keyDown="vk_space_bar" pulseControl="constantAll" interval="80"/>
</Action>
<Action type="releasedAction">
    <Command keyUp="vk_space_bar" />
</Action>
```

XML instance representation:

```
<Command
    keyDown="string" [0..1]
    keyUp="string" [0..1]
    systemCommand="string" [0..1]
    soundFileName="string" [0..1]
    vibrateTime="int" [0..1]
    pulseControl="string" [1]
    interval="int" [0..1]
</Command>
```

Table of key codes:

no_input	No inout
vk_lbutton	Left mouse button
vk_rbutton	Right mouse button
vk_cancel	Control-break processing
vk_mbutton	Middle mouse button (three-button mouse)
vk_xbutton1	X1 mouse button
vk_xbutton2	X2 mouse button
vk_back	BACKSPACE key
vk_tab	TAB key
vk_clear	CLEAR key
vk_enter	ENTER key
vk_shift	SHIFT key
vk_ctrl	CTRL key
vk_alt	ALT key
vk_pause	PAUSE key
vk_caps	CAPS LOCK key
vk_kana	IME Kana mode
vk_hangul	IME Hangul mode
vk_junja	IME Junja mode
vk_final	IME final mode
vk_hanja	IME Hanja mode
vk_kanji	IME Kanji mode
vk_esc	ESC key
vk_convert	IME convert
vk_nonconvert	IME nonconvert
vk_accept	IME accept
vk_mode_change	IME mode change request
vk_space_bar	SPACEBAR
vk_page_up	PAGE UP key

vk_page_down	PAGE DOWN key
vk_end	END key
vk_home	HOME key
vk_arrow_up	LEFT ARROW key
vk_arrow_down	UP ARROW key
vk_arrow_left	RIGHT ARROW key
vk_arrow_right	DOWN ARROW key
vk_select	SELECT key
vk_print	PRINT key
vk_execute	EXECUTE key
vk_print_screen	PRINT SCREEN key
vk_ins	INS key
vk_del	DEL key
vk_help	HELP key
vk_0	0 key
vk_1	1 key
vk_2	2 key
vk_3	3 key
vk_4	4 key
vk_5	5 key
vk_6	6 key
vk_7	7 key
vk_8	8 key
vk_9	9 key
vk_a	A key
vk_b	B key
vk_c	C key
vk_d	D key
vk_e	E key
vk_f	F key
vk_g	G key
vk_h	H key
vk_i	I key
vk_j	J key
vk_k	K key
vk_l	L key
vk_m	M key
vk_n	N key
vk_o	O key
vk_p	P key
vk_q	Q key
vk_r	R key
vk_s	S key
vk_t	T key
vk_u	U key
vk_v	V key
vk_w	W key
vk_x	X key

vk_y	Y key
vk_z	Z key
vk_lwin	Left Windows key (Natural keyboard)
vk_rwin	Right Windows key (Natural keyboard)
vk_apps	Applications key (Natural keyboard)
vk_sleep	Computer Sleep key
vk_numpad_0	Numeric keypad 0 key
vk_numpad_1	Numeric keypad 1 key
vk_numpad_2	Numeric keypad 2 key
vk_numpad_3	Numeric keypad 3 key
vk_numpad_4	Numeric keypad 4 key
vk_numpad_5	Numeric keypad 5 key
vk_numpad_6	Numeric keypad 6 key
vk_numpad_7	Numeric keypad 7 key
vk_numpad_8	Numeric keypad 8 key
vk_numpad_9	Numeric keypad 9 key
vk_multiply	Multiply key
vk_add	Add key
vk_separator	Separator key
vk_subtract	Subtract key
vk_decimal	Decimal key
vk_divide	Divide key
vk_f1	F1 key
vk_f2	F2 key
vk_f3	F3 key
vk_f4	F4 key
vk_f5	F5 key
vk_f6	F6 key
vk_f7	F7 key
vk_f8	F8 key
vk_f9	F9 key
vk_f10	F10 key
vk_f11	F11 key
vk_f12	F12 key
vk_f13	F13 key
vk_f14	F14 key
vk_f15	F15 key
vk_f16	F16 key
vk_f17	F17 key
vk_f18	F18 key
vk_f19	F19 key
vk_f20	F20 key
vk_f21	F21 key
vk_f22	F22 key
vk_f23	F23 key
vk_f24	F24 key
vk_num_lock	NUM LOCK key
vk_scroll_lock	SCROLL LOCK key

vk_lshift	Left SHIFT key
vk_rshift	Right SHIFT key
vk_lctrl	Left CONTROL key
vk_rctrl	Right CONTROL key
vk_lmenu	Left MENU key
vk_rmenu	Right MENU key
vk_browser_back	Browser Back key
vk_browser_forward	Browser Forward key
vk_browser_refresh	Browser Refresh key
vk_browser_stop	Browser Stop key
vk_browser_search	Browser Search key
vk_browser_favorites	Browser Favorites key
vk_browser_home	Browser Start and Home key
vk_volume_mute	Volume Mute key
vk_volume_down	Volume Down key
vk_volume_up	Volume Up key
vk_next_track	Next Track key
vk_prev_track	Previous Track key
vk_media_stop	Stop Media key
vk_play_pause	Play/Pause Media key
vk_start_mail	Start Mail key
vk_select_media	Select Media key
vk_start_app_1	Start Application 1 key
vk_start_app_2	Start Application 2 key
vk_oem_1	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard; the ';:' key
vk_oem_plus	For any country/region; the '+' key
vk_oem_comma	For any country/region; the '; ' key
vk_oem_minus	For any country/region; the '-' key
vk_oem_period	For any country/region; the '.' key
vk_oem_2	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard; the '/?' key
vk_oem_3	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard; the '`~' key
vk_oem_4	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard; the '[{' key
vk_oem_5	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard; the '\\|' ke
vk_oem_6	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard; the ']}' key
vk_oem_7	Used for miscellaneous characters; it can vary by keyboard. For the US standard keyboard; the 'single-quote/double-quote' key
vk_oem_8	Used for miscellaneous characters; it can vary by keyboard.
vk_oem_102	Either the angle bracket key or the backslash key on the RT 102-key keyboard
vk_process_key	IME PROCESS key
vk_packet	Used to pass Unicode characters as if they were keystrokes.
vk_attn	Attn key
vk_crsel	crsel key
vk_exsel	ExSel key
vk_erof	Erase EOF key
vk_play	Play key
vk_zoom	Zoom key
vk_pa1	PA1 key
vk_oem_clear	Clear key

vk_arrow_up_mid	UP ARROW key on the middle of extended keyvbard
vk_arrow_down_mid	DOWN ARROW key on the middle of extended keyvbard
vk_arrow_left_mid	LEFT ARROW key on the middle of extended keyvbard
vk_arrow_right_mid	RIGHT ARROW key on the middle of extended keyvbard
vk_page_up_mid	PAGE UP key on the middle of extended keyvbard
vk_page_down_mid	PAGE DOWN key on the middle of extended keyvbard
vk_end_mid	END key on the middle of extended keyvbard
vk_home_mid	HOME key on the middle of extended keyvbard
vk_ins_mid	INSERT key on the middle of extended keyvbard
vk_del_mid	DELETE key on the middle of extended keyvbard
joystick_button_1	Joystick button 1
joystick_button_2	Joystick button 2
joystick_button_3	Joystick button 3
joystick_button_4	Joystick button 4
joystick_button_5	Joystick button 5
joystick_button_6	Joystick button 6
joystick_button_7	Joystick button 7
joystick_button_8	Joystick button 8
joystick_button_9	Joystick button 9
joystick_button_10	Joystick button 10
joystick_button_11	Joystick button 11
joystick_button_12	Joystick button 12
joystick_button_13	Joystick button 13
joystick_button_14	Joystick button 14
joystick_button_15	Joystick button 15

Element: Attitude

`Attitude` makes button attitude sensitive.

Attitude value is in neutral device position 0. When device is deflected value can be [-1..1] (at max deflection positions). We will take into account here absolute value, so value can be only [0..1].
0 – neutral position
1 – max deflection position
The `direction` which element `Attitude` takes into account can be:
"rollLeft"
"rollRight"
"pitchBack"
"pitchFront"

There are is also possible do define threshold for setting object to the pressed state. The `threshold` is defined like percent of the deflection. It can be [0..99].

Example of simple `Attitude` element. Object state will be set to the pressed state when device roll left value exceeds 10% of maximum deflection:

```
<Attitude direction="rollLeft" threshold="10" />
```

Pulse controlling

There are also possible to use pulse controlling. Application offers two type of pulse controlling at the `Attitude` element: *`constantInterval`* and *`constantPressedTime`*.

`constantInterval`

pressed pulse + released pulse = `interval`. There are defined: `minTime and interval`.

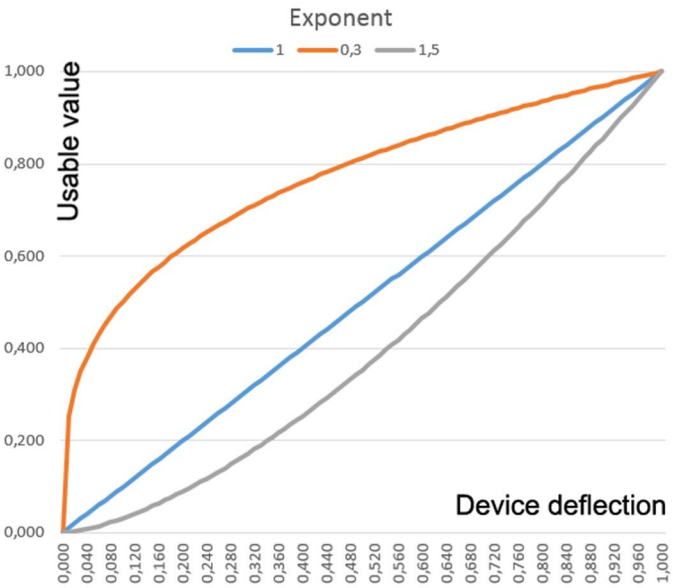
- When the device deflection cross the `threshold` from neutral position, it start sending pulses. Pressed pulses has value `minTime` and released pulses has value `interval`-`minTime`.
- As deflection of the device is bigger, pressed pulses are longer and released pulses are shorter, but still pressed pulse + released pulse = interval
- When the device reaches max deflection position, pulse controlling stopped and object is set to pressed state.

Example:

```
<Attitude direction="rollLeft" threshold="7" pulseControl="constantInterval" minTime="10"
interval="100" exponent="0.7" />
```

`constantPressedTime`

pressed pulse time = `pressedTime`, relaased time can vary from `minReleasedTime` till `maxReleasedTime`. It is depended of device deflection.



As you can in the sample, there is an attribute called `exponent`. This attribute is used for changing the deflection linearity to curve. `Attitude` element do not use directly deflection value but deflection value powered by exponent.

`direction`	*`"rollLeft"`* or *`"rollRight"`* or *`"pitchBack"`* or *`"pitchFront"`*
`threshold`	Value [1..99]
`exponent`	Value [0.3..1.5] Default value = 1
`pulseControl`	*`"constantInterval"`* or *`"constantPressedTime"`*
`minTime`	Used for *`"constantInterval"`* Minimum pressed time in milliseconds
`interval`	Used for *`"constantInterval"`* Whole length of the pressed + release time in in milliseconds
`pressedTime`	Used for *`"constantPressedTime"`* Pressed time in milliseconds

minReleasedTime	Used for *"constantPressedTime"* Minimum released time in milliseconds
maxReleasedTime	Used for *"constantPressedTime"* Maximum released time in milliseconds

XML instance representation:

```
<Attittude
    direction ="string" [1]
    threshold="float" [1]
    exponent="float" [0..1]
    pulseControl="string" [0..1]
    minTime="int" [0..1]
    interval="int" [0..1]
    pressedTime="int" [0..1]
    minReleasedTime="int" [0..1]
    maxReleasedTime="int" [0..1]
</ Attittude >
```