

This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. The article is believed to be still valid for the original topic scope.



Contents

- [1 Introduction](#)
- [2 Screen resolution](#)
- [3 Font size](#)
- [4 Memory](#)
- [5 The green plus sign](#)
- [6 Connection logo might block relevant content](#)
- [7 See also](#)

Intoduction

Even though WidSets is available for all devices supporting MIDP2, there are some details that should be taken into consideration to ensure smooth operation on multiple platforms.

This article also introduces some guidelines for the widget development process. Even though WidSets as a platform gives the developers freedom to do whatever they want when it comes to developing their own widgets, there are some commonly used UI conventions that help the end users to keep track of what their widgets are doing.

Screen resolution

Screen resolutions vary greatly from one mobile phone to another. The WidSets service introduces a built-in mechanism to scale graphics (and layouts) down whenever it runs on a low resolution device. To handle this well, devices are divided into three categories based on their screen resolution. These resolution sets are Small, Medium, and Large.

The resolution model is decided by the minimum amount of width or height of the resolution by the following formula:

```
Small <= 140 pixels < Medium <= 200 pixels < Large
```

All resources you create for a widget should be in the Large resolution. WidSets automatically scales down the graphics for smaller resolution phones; please see the example line of code below. The scaling factors are 0.75 for the Medium resolution and 0.5 for the Small resolution.

Whenever you define resources in the *widget.xml* document, you can add a scale parameter with the value `?true?`. This tells the WidSets server that this asset should be resized according to the resolution model of the end user device. Usually this should always be done for logos, but it might not be necessary for some resources such as graphics that paint the background of your widget. Test the scaling to find a suitable

solution.

Introducing a single image resource in the *widget.xml*:

```
<resources>
  ...
  <img src=?logo.png? scale="true"/>
  ...
</resources>
```

In addition to image scaling, you can use the *?sp?* unit to define the position of elements and views in your layout. SP stands for Scaled Pixel. On a high resolution device 100 sp is 100 px, on a medium resolution device 100 sp is 75 px, and on a low resolution device 100 sp is 50 px. In full mode, the maximum image size for the minimum view is 160 pixels. To ensure the scaling is working properly, test your design on as many resolutions as possible.

Font size

Different devices have different fonts and font sizes vary even between devices with the same resolution. This should be taken in account when creating layouts that accommodate text. For example, if your minimized view displays the title of the feed, make sure that the view is large enough, and that enough space is given to the feed title component so that the text does not get clipped. You can do this by defining heights in em units: 1 em is the height of the capital letter M. Remember to use the em unit in calculation of the minimum height and where you decide to use it.

The following element definition creates a margin of 5 pixels on each side and has a height of 1 em, enough to fit one line of medium sized text:

```
<view>
  ...
  <label class=?feedTitle? top="0px" right="100%-5px" bottom="1em" left="5px">
    ${feedtitle}
  </label>
  ...
</view>
```

Note: If you are using the small font size in your style, please use the correct unit, *es*. For the large font size, the unit is *el*.

Memory

On some lower end devices, memory available for a Java application is very limited. As a rule of thumb, the data in your widget should not exceed 30 KB (excluding the PNG files used for Web display).

Also try to keep the number of elements defined in a view to a minimum, since these eventually eat up your memory and processing power.

To optimize your widget and to get it running smoothly, use the stylesheet as much as possible to provide visual effects. If something is not doable as a style within a certain element or a view, then you can make a new decorative area and style it the way you want. Note that creating multiple decorative areas requires much

more computing power than simply adding a style to an element.

The green plus sign

For widgets that are based on fetching content from the Internet, having a commonly known visual element in each widget is a useful way to indicate to the user when there is new content. For this purpose, we are encouraging developers to use the green plus sign that can be found in the RSS reader example widget. While developing your own graphics, do keep the minimized viewNew view in your widget accompanied by the green plus icon that is to be located in the lower right corner of the widget.

Connection logo might block relevant content

Most of the devices with Java support have a connection logo that can block the view of the content area if the widget design does not take this into account. For example, if the maximized widget view starts from the top of the screen with date aligned to left or right corner, this can sometimes leave the user unable to see the date.

The connection logo is usually on the left side of the screen in Nokia devices, and on the right side of the screen in Sony Ericsson devices. To avoid the logo blocking relevant content, either create a banner that takes up the space where the connection logo is shown, or use styles to add padding or a margin to move the content beyond that area.

See also

- [Introduction to developing WidSets widgets](#)
- [Widget files](#)
- **Mobile developing and design guidelines**
- [Publishing your own widgets](#)
- [Widget Configuration 2.0](#)
- [Stylesheet](#)