

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 Setup](#)
- [2 Info](#)
- [3 Services](#)
- [4 Parameters](#)
- [5 Variables](#)
- [6 Resources](#)
- [7 Layout](#)
- [8 Dimensions](#)
- [9 Elements](#)
 - ◆ [9.1 Label](#)
 - ◆ [9.2 Text](#)
 - ◆ [9.3 Script](#)
 - ◆ [9.4 Decorate](#)
 - ◆ [9.5 Img](#)
- [10 See also](#)

Setup

A widget configuration defines all properties of a single widget. Version 2.1 consists of one XML-file where the widget is defined. A widget can also have other resources, such as images, a CSS-like stylesheet, language files, script code, and binary files. This section describes the format of the XML file (*widget.xml*). The XML-file **must** be encoded with UTF-8 encoding.

A *widget.xml* file starts with these two rows:

```
<?xml version="1.0" encoding="UTF-8"?>
<widget spec_version=?2.1?>
```

Widget_Configuration_2.1

The widget element is the root element of the XML. It must have the attribute *spec_version* which specifies the version of the document. Currently 2.1 is the newest version.

Info

Info contains general metadata about the widget.

The following info fields are mandatory:

Name	Min	Max	Description
name	1	32	The name of the widget.
version	number.number		The version of the widget.
author	3	32	The name of the author. This does not need to be the uploader and publisher of the widget.
shortdescription	3	160	A short description of the widget.

The following info fields are optional:

Name	Min	Max	Description
clientversion	number.number		Client version needed to run the widget.
longdescription	3	10000	A longer description of the widget.
tags	3	200	Descriptive tags for the widget. For example, blog images.
help	A Help page for the widget that can be accessed through the widget homepage.		

help section The help element can contain a help text for several languages. A help page is contained in a 'text' element with the language code as an attribute:

```
<help default="en" showFirstTime="true">
  <text lang="fi">Ohjeet suomeksi.</text>
  <text lang="en">Widget instructions in English.</text>
</help>
```

The 'help' element has two attributes:

Name	Values	Description
default	Language id of a text element	This language will be used as the default help text of the widget if the language that has been set on the device does not match the Language id of any help page
showFirstTime	true/false	If true, the help will be shown on the first opening of the widget

The minimum length for a help text is 3 characters and the maximum is 10000 characters.

Example:

```

<info>
  <name>Blog reader</name>
  <version>1.2</version>
  <author>anttihei</author>
  <shortdescription>a short desc</shortdescription>
  <clientversion>0.97</clientversion>
  <longdescription>
    <![CDATA[This is a long description for the widget.]]>
  </longdescription>
  <tags>news blogs</tags>
  <help default="en">
    <text lang="en">Widget instructions here in English.</text>
  </help>
</info>

```

Services

A widget can call a server side service using the id associated with the service. A service definition must have a *type* and an *id* attribute. There is also an optional attribute *version* which identifies the version of the service. Different versions of the same service can have different input and output that has to be taken into account in the widget script. The ids of the services must be unique within a widget.

A service can have service parameters. Their values are defined by using references from normal parameters. Normal widget parameters can be linked to the service parameters by a *reference* element inside the service element. The value of the attribute *from* points to a parameter name such as *feedurlrss* (See [Parameters](#)). The value of *to* must be a valid service parameter of the service in question. Valid service parameters can be found in [Available content fetching services](#).

Filters can be associated with HTTP and Webfeed services. A filter is referenced by its id. A filter reference must have an id attribute.

Example:

```

<services>
  <service type="syndication" id="feed1" version="1">
    <reference from="feedurlrss" to="feedurl"/>
  </service>
  <service type="http" id="getter" version="1">
    <filter id="album_name"/>
  </service>
</services>

```

Parameters

Parameters are dynamic containers of settings related data that can be used in numerous different use cases. Parameter values are always kept on the WidSets server, and, for example, reloading a widget always defaults to these parameters.

In comparison, values that are kept in the client side storage are always scrapped when the widget is reloaded. Usable parameters need to be defined in the *widget.xml* and can then be used and changed in the code of the

Widget_Configuration_2.1

widget.

Widget parameters are also tightly integrated into the WidSets client UI and parameter values can be changed under each widgets settings page (Widget / Settings). Additionally, the widget developers have control over the editability, visibility, name, and type of parameters while creating their own widgets. Parameters also do not necessarily have to be sent to the mobile.

A widget can have a maximum amount of 100 parameters. There are six different types of parameters.

Parameter types:

Type	Description
string	A string of text.
number	A numeric value.
url	A URL of valid format.
password	A String of text that is not visible on the web site or mobile device.
auth	A parameter used to identify an authentication scheme.

A parameter must have an attribute name. Other attributes are optional, and their default values will be used.

Parameter attributes:

Name	Default	Min	Max	Description
name		2	32	The name of the parameter.
type	string			The parameter type.
description		0	255	A short description that is shown when the parameter is edited.
help		0	1000	A longer help text for parameter usage.
editable	true	{true, false}		Defines whether or not the parameter values are editable.
protected	false	{true, false}		Defines whether or not the parameter values are copied when the widget is duplicated, shared, or published.
visible	true	{true, false}		Defines whether or not the parameter is visible in the editing lists.
sendmobile	true	{true, false}		Defines whether or not the parameter is sent to the mobile widget.

These attributes (except name) can also be in elements of the same name:

```
<parameter name="param1">
  <type>string</type>
  <help>
    This text will contain the instructions for the parameter.
  </help>
  <value>This is a default value for the parameter</value>
```

```
</parameter>
```

The parameter value can be a simple value, or it can have options:

```
<value>This is a default value for the parameter</value>
```

The maximum length of a value is 10000 characters.

or

```
<value>
  <option value="sel_red" selected="true">Red</option>
  <option value="sel_green">Green</option>
  <option value="sel_blue">Blue</option>
</value>
```

An option value must have the attribute value. Its maximum length is 1000 characters. The option text maximum length is 200 characters.

Example:

```
<parameters>
  <parameter type="string"
    name="widgetname"
    description="WidSets Forum"
    editable="true"
    protected="true"/>
  <parameter type="url"
    name="feedurlrss"
    description="RSS Feed URL"
    editable="false">
    <value>
      <option value=" http://www.examplesite.com/feed1" selected="true">
        Recent Topics
      </option>
      <option value="http://www.examplesite.com/feed2">
        General News And Announcements
      </option>
      <option value="http://www.examplesite.com/feed3">
        Open Floor
      </option>
      <option value="http://www.examplesite.com/feed4">
        Mobile Related
      </option>
    </value>
  </parameter>
</parameters>
```

Variables

Parameter values can also be references to other parameters. A variable is in the format `${parameter_name}`. Also, option text can be referenced with an `.option.name` postfix: `${param_name.option.name}`.

Example:

Widget_Configuration_2.1

```
<parameter name="widgetname" editable="false" visible="true">
  <value>My ${color} Widget</value>
</parameter>
<parameter name="color">
  <value>
    <option value="red">Red</option>
    <option value="blue">Blue</option>
  </value>
</parameter>
```

Variables can be used also in option values:

```
<parameter name="selection" type="string">
  <value>
    <option value="val1">Selection 1</option>
    <option value="val2">Selection 2</option>
    <option value="${customval}">Custom Selection</option>
  </value>
</parameter>
<parameter name="customval" type="string" description="Custom value"/>
```

Resources

The widget resources are defined in the resources section.

Valid resource types:

Type	Attributes	Description
img	src	An image.
stylesheet	src	A stylesheet file.
code	src	A script source file.
binary	src	Any file treated as binary data.

The length of the attribute src must be in [1,64]. And the length of attribute id must be in [2,16].

Stylesheet and code can be embedded inside the element. The attribute src is then omitted. These embedded resources must be inside CDATA.

Example:

```
<resources>
  
  
  
  
  
  <code src="myscript.he">
  <stylesheet src="style.css"/>
</resources>
```

Layout

Each widget consists of views. In the case of feed reader widgets, there are four basic views. There are two minimized views: one for the normal minimized representation and another to tell the user that there are new items in the feed. Similarly, there are two maximized views: one for listing feed topics and another for presenting a single feed item.

Each view contains elements that define the layout and display characteristics of the view. For performance reasons, it is recommended to minimize the use of separate elements within a view. For example, it is not necessary to use the `decorate` element just to render the background, since the style of that particular view can take care of that.

Overlapping elements may also not work as expected. One good example is that using a scrollable element always renders the background, which is defined in that particular view. The component model where elements are tied is strictly hierarchical, and when a specific component is repainted, the decorations of the parent components are also applied.

The widget layout is constructed using WidSets' own element model. Most elements that you can use when building a widget are actually containers. For example, you can use the `` tag to display a picture.

This document mostly talks about elements when discussing the layout model, but keep in mind that the [img](#), [label](#), [text](#), [decorate](#) and [script](#) are actually elements, and that you can define positioning by giving these individual elements their `top`, `right`, `bottom` and `left` attributes.

Each element has six attributes that control its position and dimensions. These are *top*, *right*, *bottom*, *left*, *width* and *height*. All these attributes are relative to the parent container, not to the whole widget view. If you specify a width of 50%, this in effect means that the element will get a width that is half the width of the parent container. If you specify a left position of 5px, the position is calculated from the parent containers left edge by indenting 5 pixels.

The `webview` element is used for positioning [parameter](#) values and mobile [images](#) on the minimized view on the web Manager. For example, the name of the widget (selected by the user in Options) can be displayed also on the website.

The value of the attribute `style` is normal web CSS and is **not** related to the stylesheet for the widget.

Example:

```
<layout minimizedheight="10px+20sp+1em">
  <view id="viewIndex" left="90%" right="110px">
    
    <label top="10px+20sp" right="100%-5px" bottom="100%">
      ${widgetname}
    </label>
    <text>Multi-lined text component</text>
    <decorate class="myDecoration"/>
    <script id="myScriptAction"/>
  </view>

  <webview>
    <weblabel style="css style here">${widgetname}</weblabel>
    <webicon style="css style here" src="image.png"/>
  </webview>
```

```
</layout>
```

Dimensions

The dimensions of an element can be specified in a number of units.

The units

- **px**: absolute pixels (default).
- **em**: The height of the M-letter.
- **es**: The height of the small M-letter.
- **el**: The height of the large M-letter.
- **%**: Percentage of the outer box.
- **sp**: Scaled pixels. Resolution-dependent pixels.
On a large-resolution device, **1sp = 1px**.
- Expression of any of the above. For example, **50%+1em-10sp+1px**.

There are two ways an element can be positioned: absolute positioning or block positioning.

Elements

This section defines the elements that are available inside views. Currently available elements are [img](#), [label](#), [text](#), [decorate](#) and [script](#). See below for more precise explanations.

Label

The label tag paints a text label. A label does not wrap because it consists of only one line. If the dimensions are not big enough to contain the whole representation of the text, the text is truncated and three dots (...) are placed at the end of the text.

Example:

```
<label class="yourStyle">This is a label</label>
```

A good use case for labels is printing **parameter values to the UI**. For example, to print a widget name in a label, use the following action:

```
<label class="yourStyle">${widgetname}</label>
```

Or you can print **the name of an optionized parameter**:

```
<label class="yourStyle">${feedurlparam.option.name}</label>
```

Text

The text tag paints text. It can span multiple lines, and it wraps.

```
<text class="yourStyle">This is a multi-lined text component that wraps</text>
```

Script

The script tag introduces a scripted component inside the view.

```
<script id="miniView" class="yourStyle" top="0%" right="100%" bottom="100%" left="0%"/>
```

Decorate

The decorate tag decorates the given area with the given style.

```
<decorate class="yourStyle" top="0%" right="100%" bottom="100%" left="0%"/>
```

Img

The *img* tag refers to an image resource. Image resources need to be defined in the widget resources in *widget.xml*.

Defining the resources:

```
<resources>
  
  
  ...
</resources>
```

After defining the images in the appropriate place, you can use a symbolic reference to a particular widget image.

Example:

```

```

Images found in the WidSets client can also be referenced as images, but beware that they may change without warning.

See also

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

Widget_Configuration_2.1

- [Widget Configuration 2.0](#)
- [Stylesheet](#)