

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

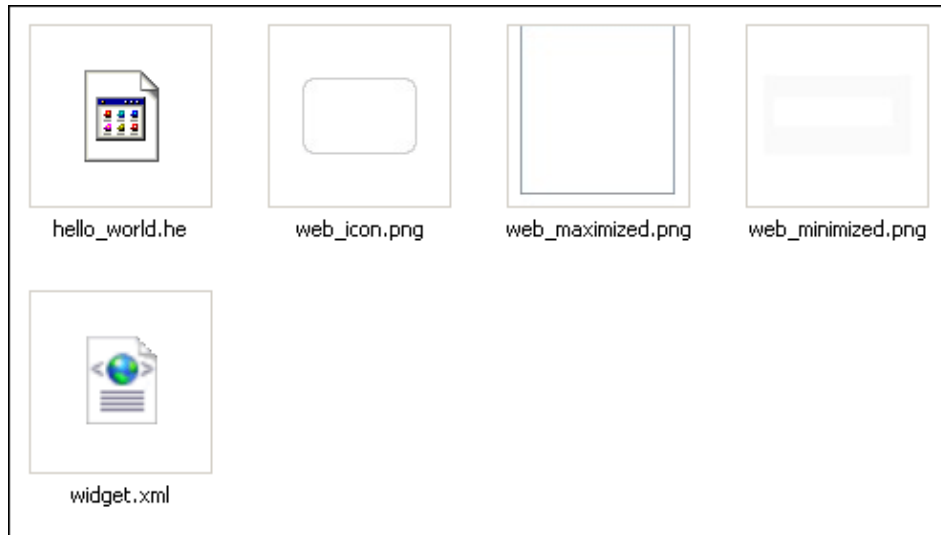
- [1 Introduction](#)
- [2 Project Files](#)
- [3 Source Code](#)
 - ◆ [3.1](#)
 - [widget.xml](#)
 - ◇ [3.1.1](#)
 - [info](#)
 - [tag](#)
 - ◇ [3.1.2](#)
 - [parameter](#)
 - [tag](#)
 - ◇ [3.1.3](#)
 - [resources](#)
 - [tag](#)
 - ◇ [3.1.4](#)
 - [layout](#)
 - [tag](#)
 - ◆ [3.2](#)
 - [hello_world.he](#)
 - ◇ [3.2.1](#)
 - [startWidget\(\)](#)
 - ◇ [3.2.2](#)
 - [openWidget\(\)](#)
 - ◇ [3.2.3](#)
 - [getSoftKey\(\)](#)
 - ◇ [3.2.4](#)
 - [actionPerformed\(\)](#)
 - [4 Preview](#)
 - [Images](#)
 - [5 See Also](#)

Introduction

Last episode, I have already . This page will describe you how the **hello_world** source code work.

Project Files

In **hello_world** example folder, there are 5 file contained as shown below.



hello_world.he and widget.xml are **Source Code files** while web_icon.png, web_maximized.png and web_minimized.png are **Preview Images files**. Let's see what they are used for.

Source Code

Each project, it must has at least 2 files **widget.xml** and **Helium source file**

widget.xml

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

<widget spec_version="2.1">
  <info>
    <name>Hello World</name>
    <version>1.0</version>
    <author>render</author>
    <clientversion>1.0</clientversion>
    <shortdescription>Very simple widget</shortdescription>
    <longdescription>Simplest possible widget saying hello to the world.</longdescription>
    <tags>test example hello world</tags>
  </info>

  <parameters>
    <parameter type="string" name="widgetname" description="Name of widget" editable="no">Hello W
  </parameters>

  <resources>
    <code src="hello_world.he"/>

  <stylesheet>
    bg {
      color-1: white;
      background: solid black;
      align: hcenter vcenter;
      border: 1 1 1 1;
      border-type: rectangle white;
    }
  </stylesheet>
</widget>
```

WidSets_for_Rookie_EP_3::_Understand_Hello_World

```
text {
    color-1: white;
    padding: 2 2 2 2;
}
</stylesheet>
</resources>

<layout minimizedheight="65sp">
    <view id="viewMini" class="bg">
        <label class="text">${widgetname}</label>
    </view>

    <view id="viewMaxi" class="bg">
        <script id="hello" class="text"/>
    </view>

    <webview>
        <weblabel class="top: 0px; left: 10px;" style="color: black;">${widgetname}</weblabel>
    </webview>
</layout>

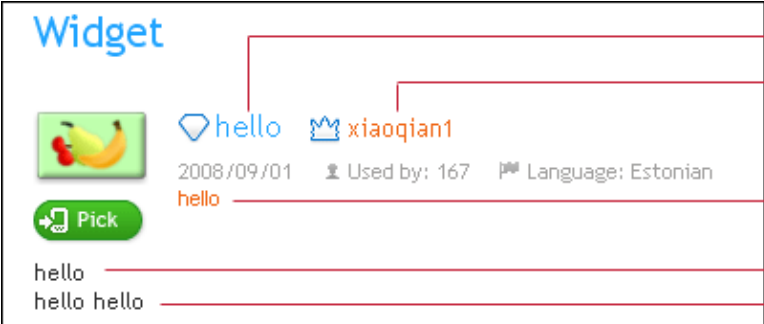
</widget>
```

XML is an manifest file used for describe widget overview. First, look at **info** tag,

info tag

```
<info>
    <name>Hello World</name>
    <version>1.0</version>
    <author>render</author>
    <clientversion>1.0</clientversion>
    <shortdescription>Very simple widget</shortdescription>
    <longdescription>Simplest possible widget saying hello to the world.</longdescription>
    <tags>test example hello world</tags>
</info>
```

These information will be shown when user browse through the [WidSets] site or through search feature in WidSets application.



The screenshot shows a widget listing interface. Red lines connect XML tags to specific UI elements:

- <name>** points to the word "Widget" at the top left.
- <author>** points to the author name "xiaoqian1" (with a crown icon).
- <tag>** points to the tag "hello" (with a diamond icon).
- <shortdescription>** points to the text "hello" below the widget icon.
- <longdescription>** points to the text "hello hello" below the "Pick" button.

parameter tag

Data in **parameter** tag are predefined variable used in another part of widget.xml

```
<parameters>
  <parameter type="string" name="widgetname" description="Name of widget" editable="no">
    Hello World
  </parameter>
</parameters>
```

resources tag

resources tag is used for define source code file (in **code** tag) and CSS stylesheet (in **stylesheet** tag) which will be used in another part of widget.xml. If you wanna include image file in your package, you can use **** tag to do that.

```
<resources>
  <code src="hello_world.he" />

  <stylesheet>
    bg {
      color-1: white;
      background: solid black;
      align: hcenter vcenter;
      border: 1 1 1 1;
      border-type: rectangle white;
    }

    text {
      color-1: white;
      padding: 2 2 2 2;
    }
  </stylesheet>
</resources>
```

layout tag

The last part, **layout** tag is used for defining widget layout. **viewMini** and **viewMaxi** are used in Helium source code (you will see in next part) while **webview** are used in Dashboard Manager in [<http://dev.widsets.com>]. **#{widgetname}** is predefined variable defined in parameter tag above. It will be replaced with **Hello World** in compile time.

```
<layout minimizedheight="65sp">
  <view id="viewMini" class="bg">
    <label class="text">#{widgetname}</label>
  </view>

  <view id="viewMaxi" class="bg">
    <script id="hello" class="text" />
  </view>

  <webview>
    <weblabel class="top: 0px; left: 10px;" style="color: black;">#{widgetname}</weblabel>
  </webview>
</layout>
```

hello_world.he

```

/*
 * Copyright (C) 2008 Nokia Corporation.
 *
 * Licensed under separate Nokia Corporation End-user Software
 * Agreement (the "License").
 *
 * You may not use this Software except in compliance with the License.
 * The Software is distributed under the License on "AS IS" basis,
 * without warranties. See the License for the specific language
 * governing rights and limitations under the License.
 */

class hello_world
{
    //It's nice to store command ids to static constants
    const int CMD_BACK = 1;

    //MenuItems are displayed over phone's soft buttons
    //Usually to go back, ok, open options menu etc
    MenuItem BACK = new MenuItem(CMD_BACK, "Back");

    //WidSets framework will call createElement() per
    //each script-element it finds from views being
    //created by createView()

    Component createElement(String viewName,
                            String elementName,
                            Style style,
                            Object context)
    {
        //return simple label with style defined in widget.xml
        //in this case "text"
        if (elementName.equals("hello")) {
            //in order to get the label align in the middle of the
            //view you'd need to contain it inside a Flow which
            //you'd return here
            return new Label(style, "Hello World");
        }
        return null;
    }

    void startWidget()
    {
        //instantiate minimized view in startup
        setMinimizedView(createView("viewMini", getStyle("bg")));
    }

    Shell openWidget()
    {
        //instantiate maximized view when user opens this widget
        Flow view = createView("viewMaxi", getStyle("bg"));
        return new Shell(view);
    }

    MenuItem getSoftKey(Shell shell, Component focused, int key)
    {

```

WidSets_for_Rookie_EP_3:_Understand_Hello_World

```
//return the key we want to display at position=SOFTKEY_BACK
//this usually is the Right Soft Button (RSB), for other key
//positions return null, as we don't want other keys
if (key == SOFTKEY_BACK) {
    return BACK;
}
return null;
}

void actionPerformed(Shell shell, Component source, int action)
{
    //when CMD_BACK event comes in, pop the current shell (this widget)
    if (action == CMD_BACK) {
        popShell(shell);
    }
}
}
```

startWidget()

First part I wanna show you is **startWidget** function.

```
void startWidget ()
{
    //instantiate minimized view in startup
    setMinimizedView(createView("viewMini", getStyle("bg")));
}
```

This function will be called when widget is shown in Dashboard.



So if you wanna change widget appearance in dashboard, you can change it in viewMini stylesheet directly. For example, I tried to change widget.xml like this.

```
<stylesheet>
...
  bgmini {
    color-1: white;
    background: solid red;
    align: hcenter vcenter;
    border: 1 1 1 1;
    border-type: rectangle white;
  }
...
</stylesheet>

<layout minimizedheight="65sp">
  <view id="viewMini" class="bgmini">
    <label class="text">${widgetname}</label>
  </view>
...

```

Here is the result.



Actually, after `startWidget()` has been called, **createElement** function will be called next using id defined in `widget.xml`. But in this case, `widget.xml` isn't define any script for `viewMini`, so nothing happened.

openWidget()

Next, let's see at **openWidget** function. This function will be called when user select widget.

```
Shell openWidget()
{
    //instantiate maximized view when user opens this widget
    Flow view = createView("viewMaxi", getStyle("bg"));
    return new Shell(view);
}
```

After `openWidget` called, `createElement` will be called next with the id defined in `widget.xml`

```
<view id="viewMaxi" class="bg">
  <script id="hello" class="text"/>
</view>
```

You will see that in `widget.xml` `script id="hello"` has been defined. So, `createElement` will be called with `elementName="hello"`.

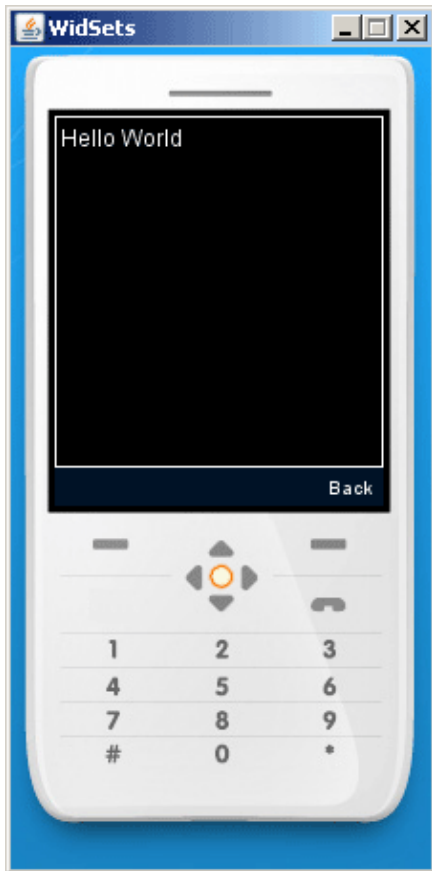
```
Component createElement(String viewName,
                        String elementName,
                        Style style,
                        Object context)
```

`openWidget()`

WidSets_for_Rookie_EP_3:_Understand_Hello_World

```
{
  //return simple label with style defined in widget.xml
  //in this case "text"
  if (elementName.equals("hello")) {
    //in order to get the label align in the middle of the
    //view you'd need to contain it inside a Flow which
    //you'd return here
    return new Label(style, "Hello World");
  }
  return null;
}
```

That's why **"Hello World"** text appear in the full screen widget.

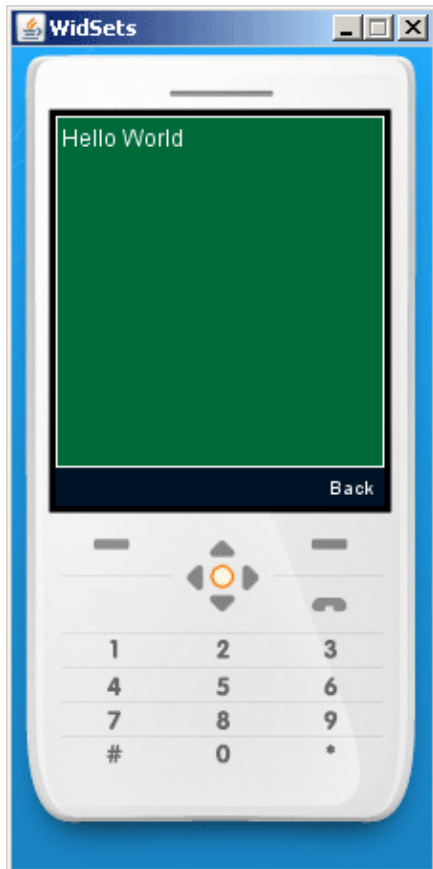


Now I tried to change appearance by modify stylesheet like this.

```
<stylesheet>
...
  bg {
    color-1: white;
    background: solid #006938;
    align: hcenter vcenter;
    border: 1 1 1 1;
    border-type: rectangle white;
  }
...
</stylesheet>
```

And here is the result.

openWidget()



getSoftKey()

getSoftKey function will be called by framework to define softkey appearance. First of all, let's define MenuItem object.

```
//It's nice to store command ids to static constants
const int CMD_BACK = 1;

//MenuItems are displayed over phone's soft buttons
//Usually to go back, ok, open options menu etc
MenuItem BACK = new MenuItem(CMD_BACK, "Back");
```

Next, let's see at **getSoftKey** function.

```
MenuItem getSoftKey(Shell shell, Component focused, int key)
{
    //return the key we want to display at position=SOFTKEY_BACK
    //this usually is the Right Soft Button (RSB), for other key
    //positions return null, as we don't want other keys
    if (key == SOFTKEY_BACK) {
        return BACK;
    }
    return null;
}
```

It will be called for each softkey, for example, SOFTKEY_OK (Left Softkey) and SOFTKEY_BACK (Right Softkey). In hello_world widget, only right softkey has been used. So this function will return MenuItem

BACK for key == SOFTKEY_BACK, other will be returned as null. And that's why "Back" text appear in the right softkey position.

actionPerformed()

Now, "Back" menu has been show at right softkey already. To handle it, actionPerformed has been used.

```
void actionPerformed(Shell shell, Component source, int action)
{
    //when CMD_BACK event comes in, pop the current shell (this widget)
    if (action == CMD_BACK) {
        popShell(shell);
    }
}
```

The function will be called by passing MenuItem value via **action** parameter. In this example, when user press Back, actionPerformed will be called with action == CMD_BACK. And then, popShell(shell) will be called to stop the fullscreen widget and return to dashboard.

Preview Images

You may already noticed that there are 3 image files in the hello_world example folder too. Let's see what they are.

- **web_icon.png** - 60x40 pixels image used for small preview
- **web_maximized.png** - 176x208 pixels image used for show screenshot in fullscreen mode
- **web_minimized.png** - 110x49 pixels image used for show screenshot in dashboard item mode

Warning: These files must be named like this and can't change to other name.



See Also

- [WidSets for Rookie EP 1 : First Step to WidSets SDK](#)
- [WidSets for Rookie EP 2 : First Compilation with WidSets SDK](#)
- **WidSets for Rookie EP 3 : Understand Hello World**
- [WidSets for Rookie EP 4 : Fasten WidSets Development](#)
- [WidSets for Rookie EP 5 : EditPlus Integration](#)
- [WidSets for Rookie EP 6 : Softkey Menu](#)
- [WidSets for Rookie EP 7 : Standard UI](#)
- [WidSets for Rookie EP 8 : Canvas](#)
- [WidSets for Rookie EP 9 : Timer](#)
- [WidSets for Rookie EP 10 : Key Handling](#)
- [WidSets for Intermediate EP 1 : HTTP Request](#)
- [WidSets for Intermediate EP 2 : HTTP with XML Filter](#)
- [WidSets for Advance EP 1 : Life Pictures Project](#)
- [WidSets SDK Tips : Emulator Language Changing](#)
- [WidSets SDK Tips : Emulator Skin Changing](#)
- [WidSets SDK Tips : Add Custom Emulator Skin](#)