

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 WidSets Development Tutorials](#)
- [2 Example of Writing a WidSet Tutorial Article using HelloWorld.he](#)
- [3 Introduction](#)
 - ◆ [3.1 Example of linking the WidSet Api to Hello.he](#)
 - ◆ [3.2 Summary](#)
- [4 See also](#)

WidSets Development Tutorials

The tutorial will guide you through the development of a simple arcade game for WidSets called "Paddle Game".

from Will Bramford at mobileradicals.com

Example of Writing a WidSet Tutorial Article using HelloWorld.he

Introduction

To create an article you will need to enter the title of your new page in the "Search" box on the top left of the page, then click "Go" this will give you title and ask you to click on "Create Page" link which will open the Wiki Editor. Now enter the following Wiki Article Header on the top line . This puts your article in the correct location. Please **DO NOT** use the FNWID. The WidSets banner is for protected documents that are locked, and if you use this your article will look like its LOCKED!!, and probably some people will not realise you have written this.

```
[[Category:WidSets]]
```

The entering of code use `< code >` keyword to start and the `< /code >` the end the block. If you want to add plain text just add two spaces to the beginning of the line, The `< br >` function will add a new line the index is created by using equals signs on each end of text.

```
=Introduction=
```

Each = will indicate a new level and repeat at the same level causes numbering and indent of the index.

This is an example of an annotated tutorial of the Hello world program, where the individual [functions](#) have been linked to the [WidSets API](#).

There is a very good example of a tutorial for a Paddle Game at www.mobileradicals.com

Example of linking the WidSet Api to Hello.he

The **createView()** function is used for creating a view for the widget. A view can be a minimized view or a maximized view depending on the widget's status.

Widget's UI components can be constructed within the **createElement()** function. This is a callback function and it gets called as a result every time the **createView()** function is called.

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

Component createElement (String viewName,
                        StringelementName,
                        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;
}
```

The **createView()** function is used for creating a view for the widget. A view can be a minimized view or a maximized view depending on the widget's status.

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

When the WidSets mobile client is opened, the **startWidget()** function is called for all widgets installed on the client's dashboard. In most cases, you should implement this function to create the widget's minimized view to be displayed on the WidSets dashboard.

```
Shell openWidget ()
```

WidSets_Development_Tutorials

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

The `openWidget()` function is called when the user selects a widget from dashboard. This is when the widget is also opened to the maximized mode. Most widgets implement the maximized mode. Within this function a `Shell` for using the views is typically created.

```
MenuItem getSoftKey(Shell shell, Component focused, int key)
{
    //return the key to be displayed at position=SOFKEY_BACK
    //this is usually the right softkey, for other key
    //positions return null

    if (key == SOFTKEY_BACK) {
        return BACK;
    }
    return null;
}
```

When the user presses a softkey of a mobile device (normally a softkey is associated to the widget's menu), `WidSets` will call the '`getSoftkey()`' to inform the widget which softkey was just clicked. You must implement this function to detect the softkey and tell `WidSets` what to do with the event. The '`getMenu()`' function is called when a softkey is designed to associate with an open menu.

```
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 '`actionPerformed()`' function is called when a key-pressed action on a softkey is detected, and the `keyAction()` function is called when the user clicks on an alphanumeric key (including the navigation key). These functions can be implemented to detect any actions done by the user and to process them accordingly. Remember that both '`actionPerformed()`' and `keyAction()` are system callback functions, and they should never be called directly.

Summary

This document is very highly linked and in a usual coding document may have a few links, but shows the value of linking, because the user does not have to keep searching API documentation for functions, and if the reader already understands the functions then the readability is increased and this is helpful all levels of readers.

Very highly linked documents can be difficult to update and broken links often occur, so always check the links when changes have been made. Look at this document in Wiki page Editing mode to get an idea of how to link to words.

See also

[formatting help](#)