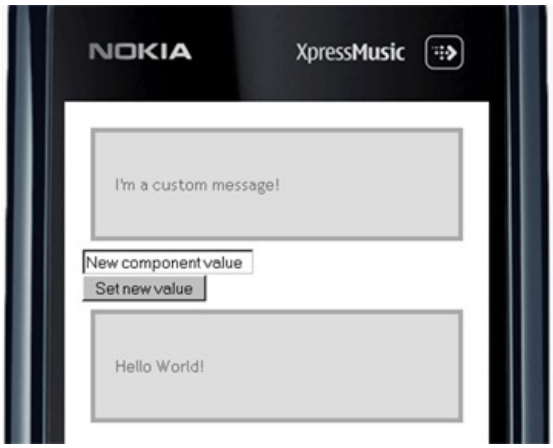
How to create a new Component for the Guarana UI Library

21 Feb 2010

This article shows all the steps involved in **creating a new Component from scratch for the Nokia WRT Guarana UI Library**.



The Component built is this article is also visible on the Guarana UI online Components Browser.

Introduction

The Guarana UI Library is a **jQuery-Based UI Library for Nokia WRT**, that provides a full set of Components ready to be used within Web Runtime widgets.

Guarana UI is structured into Components: **User Interface elements that fulfill a specific function**. Some examples include: Date, Rating and Progress Bar.

The Guarana UI structure allows to create new Components, by extending the Category:Web Apps base class. The Category:Web Apps base class represents a common base for all Guarana UI Components, providing basic and utility functions that manage event handling, vibration control and loading management of the Components.

Component files Structure

A Guarana UI Component is typically composed by:

- **1 JavaScript file**, placed in the "<GUARANA_BASE_PATH>/src/" folder
- **1 CSS file**, placed in the "<GUARANA_BASE_PATH>/themes/nokia/base/" folder
- optionally, **1 or more images** placed in the "<GUARANA_BASE_PATH>/themes/nokia/base/images/" folder

The Guarana UI Build processes

Before starting to create the new Component, it is necessary to **download and uncompress the Full Guarana UI Package**, available here: Guarana UI Full Package.

The Full Package ships with all the source files of the Guarana UI Library, and includes the **build files necessary to create custom Library versions**. To have a detailed overview of the Guarana UI folders and build process, check out the Guarana UI Overview.

Note: In this article, <GUARANA_BASE_PATH> refers to the complete path where the Guarana UI Full Package was uncompressed

Setup the Component source files

The following sections describe **how to perform the setup steps for a new Guarana UI Component**, here named TestComponent. The involved steps are:

- **creating the Component skeleton** in the Guarana UI Full Package
- **build and move the lib and themes folder** to an empty Web Runtime widget

Create an empty Web Runtime project

Before starting, **create an empty Web Runtime project**: it will be used to write and test the actual Component's code. You can do

(C) Copyright Nokia 2014. All rights reserved.

this manually, or by using the available Web Runtime IDE's Plugins.

Setup the Component skeleton structure

Now let's start, by creating the **empty source files for the new TestComponent**. These will be:

- **testcomponent.js**: place this empty file in the **"<GUARANA_BASE_PATH>/src" folder**
- **testcomponent.css**: place this empty file in the **"<GUARANA_BASE_PATH>/themes/nokia/base/" folder**

Now, it is necessary to **tell the Guarana UI Library that these files exist**. To do this, **open the "<GUARANA_BASE_PATH>/src/defaults.js" file**, and add this JavaScript code, just after all the other Components, as **properties of the window.NOKIA_DEFAULTS.modules object**:

```
/*
* Test Component
*/
'testcomponent': {
 getPath: function() { return NOKIA_PATH_JAVASCRIPT + 'testcomponent.js'; },
 requires: ['widget', 'testcomponent-css']
},

'testcomponent-css': {
 getPath: function() { return NOKIA_PATH_STYLE_ROOT + 'testcomponent.css'; },
 type: 'css',
 requires: ['base-css']
}
```

These 2 properties allow Guarana UI to know that new resources exist, and specify the **dependencies** of each of them. Most of the Components usually require only the specified dependencies, but it's possible to change them accordingly to different needs.

Before proceeding, **perform the same modifications to the "<GUARANA_BASE_PATH>/build/build/js/defaults.js" file**.

Build Guarana UI and move resources

Now that the skeleton files are ready, it is possible to **build Guarana UI by using the standard build process**. Just go into the "<GUARANA_BASE_PATH>/build/" folder and, from command-line, call:

```
ant
```

Once built, **move the "<GUARANA_BASE_PATH>/build/Guarana/lib/" and "<GUARANA_BASE_PATH>/build/Guarana/themes/" to the empty Web Runtime widget created before**.

Before proceeding, **check that the Component empty files were actually built**. If so, they should be located at these paths:

- "<GUARANA_BASE_PATH>/build/Guarana/lib/testcomponent.js"
- "<GUARANA_BASE_PATH>/build/Guarana/themes/nokia/base/testcomponent.css"

Writing the Component code

The following sections show how to write the actual Component's code.

The Component JavaScript skeleton

The very-basic Component is composed by just one function, that takes care of **initializing the Component** itself:

```
(function() {
 Nokia.TestComponent = Nokia.Widget.extend({
  initialize: function(options) {

   // initialize the Component
```


(C) Copyright Nokia 2014. All rights reserved.

```
  }
 });
})();
```

Some other functions and properties are also usually defined, and they are:

- the **_create() function**, that takes care of creating the **Component's DOM structure** and of defining all the **event handlers**
- the **Styles property**, that holds the **names of CSS classes** used by the Component

So, a more complete Component will be the following one:

```
(function() {
 Nokia.TestComponent = Nokia.Widget.extend({
  initialize: function(options) {

   // initialize the Component

  },

  _create: function() {

   // create the Component DOM structure and define events

  },

  Styles: {

   // define the main CSS styles classes used by the Component

  }
 });
})();
```

Define the Component DOM structure

The TestComponent built in this article has to show a message, and so it is possible to use the **following simple DOM structure**:

```
<div class="test-component">
 <p class="test-component-message">
  Hello World!
 </p>
</div>
```

The Component's CSS classes

Since the Component uses **2 CSS classes**, it is possible to use the Component's **Styles property** to store their names, as shown below:

```
Styles: {
 testComponent: 'test-component',
 message: 'test-component-message'
}
```

Those classes will be defined in the Component's CSS file, **testcomponent.css**. Some simple rules are the following:

(C) Copyright Nokia 2014. All rights reserved.

```
.test-component {
 margin: .5em;
 padding: .5em;
 background: #dddddd;
 border: 4px solid #aaaaaa;
}
.test-component-message {
 color: #666666;
}
```

Initializing the Component

The initialization is actually the **most important part of the Component**, as it has to:

- **initialize the Component's properties**, by using the **passed options together with the default ones**. The following code shows how to define some default options (in this case, a default message), and how to add to these options the ones passed to the Component:

```
var defaults = {
 message: "Hello World!"
};

instance.options = jQuery.extend(defaults, options);
```

- **get a reference to the Component's placeholder DOM element**:

```
instance.element = jQuery( instance.options.element );
```

- call the **Widget super-class initialization function**:

```
instance._super.apply(instance, [instance.options]);
```

- **register the component**. This is done with a call to the "registerData()" function:

```
instance.registerData("testcomponent");
```

- **create the DOM structure**. This is usually done from a separate function (usually the "_create" one), so the initialize() function just calls that:

```
instance._create();
```

- **fire the 'create' callback**, to notify the finish of the initialization process:

```
instance.fireCallback('create');
```

The complete initialize() function is the following one:

```
initialize: function(options) {
 var instance = this;

 var defaults = {
  message: "Hello World!"
```


(C) Copyright Nokia 2014. All rights reserved.

```
  };

  instance.options = jQuery.extend(defaults, options);

  instance.element = jQuery( instance.options.element );

  instance._super.apply(instance, [instance.options]);

  instance.registerData("testcomponent");

  instance._create();

  instance.fireCallback('create');
 }
```

Create the DOM structure

Let's take back the Component's DOM structure: it is now necessary to create it via JavaScript, defining the necessary event handlers, and using the Styles property to assign the right CSS class to the various elements. The following code shows how this can be performed:

```
_create: function() {
 var instance = this;
 var options = instance.options;
 var element = instance.element;
 var Styles = instance.Styles;

 element.each(function(i, el) {
  Nokia.util.addClass(el, Styles.testComponent);
 });

 instance.hello = Nokia.dom.parseHTML('<p class="' + Styles.message + '">' +
this.options.message + '</p>');

 element.append(instance.hello);

}
```

In the above code, it is possible to see how to use the Component's "options" property to retrieve the message to display.

Add some interactivity to the Component

In order to add some interactivity, let's define **a function that allows to change the displayed message**. This function is visible below:

```
setValue: function(html)
{
 var instance = this;

 instance.hello.html(html);
}
```

Using the component

The component is now **ready for use**, and to actually create a TestComponent within a Web Runtime widget the following steps

(C) Copyright Nokia 2014. All rights reserved.

have to be performed:

- **Include the Guarana UI CSS dependencies** (as explained here: GuaranaUI-QuickStart):

```
<!-- Themeroller CSS -->
<link rel="stylesheet" href="themes/themeroller/default-theme/Themeroller.css"
type="text/css" media="screen">
<!-- Specific Theme/Resolution CSS -->
<link rel="stylesheet" href="themes/nokia/ext-theme/default/360x640/custom.css"
type="text/css" media="screen">
```

- Include the **JavaScript dependencies**:

```
<!-- jQuery file -->
<script src="/lib/jquery/jQuery.js" type="text/javascript" charset="utf-8"></script>
<!-- Guarana file -->
<script src="/lib/Guarana.js" type="text/javascript" charset="utf-8"></script>
```

- Define a **placeholder DOM element** in the widget's HTML code:

```
<div id="testcomponent01"></div>
```

- Call the **"Nokia.use" function** to tell Guarana UI that you need to use the new component. Details about the **Nokia Loader** are available here: Nokia Loader. Before calling the Nokia Loader, it is necessary to **define the 2 Guarana UI PATH variables**, to specify where its folders are located:

```
<script>
NOKIA_PATH_JAVASCRIPT = 'lib/';

NOKIA_PATH_STYLE_ROOT = 'themes/nokia/base/';

Nokia.use('testcomponent', init);
</script>
```

- When the Component resources will be available, the init() function is called. Within the init() function, it is so possible to **create an instance of the newly created TestComponent**:

```
function init()
{
    var testComponent1 = new Nokia.TestComponent({
        element: '#testcomponent01',
        message: "I'm a custom message!",
        create: function() {
            alert('test component created');
        }
    });
}
```

If all goes well, an alert message with the "test component created" message should be displayed, just after the Component creation.

Finalize the Component

Now that the Component is ready, it is possible to **move its resources back to the Guarana UI Full Package** folders,

(C) Copyright Nokia 2014. All rights reserved.

overwriting the empty files created at the beginning.

Note: if the **Component also uses images** (e.g.: referenced by the Component's CSS file), it is necessary to move them also to the Guarana UI folders. In case of images, **the folder to use is the "<GUARANA_BASE_PATH>/themes/nokia/base/images/"**.

Once the Component's files are copied back to the Guarana UI folders, it is possible to launch a new build process, that will **optimize and compress the Component's code**. Done that, everything is ready to use and distribute the new Component.

Links and Resources

- Guarana UI: a jQuery-Based UI Library for Nokia WRT
- GuaranaUI-Documentation
- GuaranaUI-Overview
- Guarana UI online Components Browser

(C) Copyright Nokia 2014. All rights reserved.