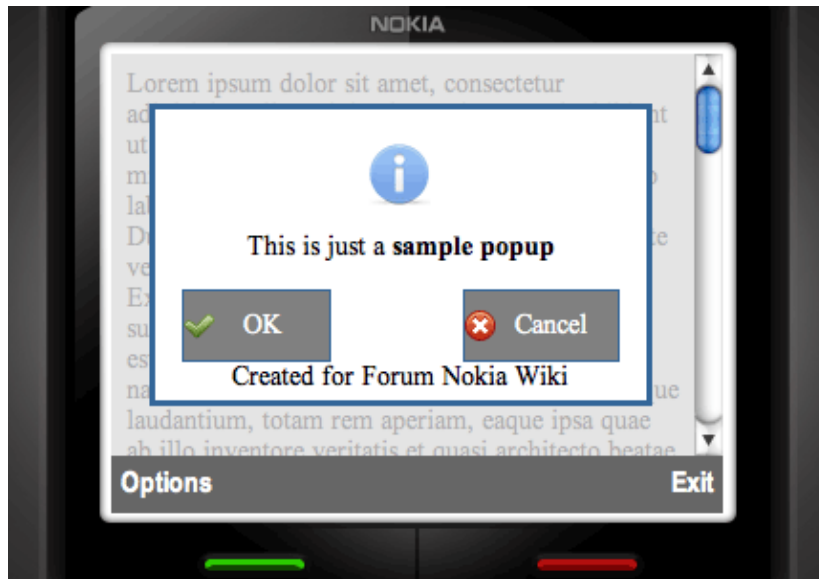




This article explains **how to build and use a modal popup component for Web Runtime widgets.**



Modal popup windows or messages are useful to **display important information to the user**, blocking the rest of the Widget interface until the popup is disposed, typically by pressing a button.

Contents

- [1 Popup component: how to use it](#)
- [2 Popup component implementation](#)
 - ◆ [2.1 Popup constructor](#)
 - ◆ [2.2 Building and showing the Popup](#)
 - ◆ [2.3 Popup buttons' click handlers](#)
 - ◆ [2.4 Hiding the component](#)
- [3 Downloads](#)

Popup component: how to use it

The Popup component presented in this article can be used by following these steps:

- Include the Popup.js JavaScript file in your code (source code available here: [Media:Wrt Popup component.zip](#))

```
<script language="javascript" type="text/javascript" src="Popup.js"></script>
```

- Include a **semi-opaque image to be used to opacize the widget's content** when the popup is displayed. An example image, used in this article, is the following:

Since the **image will be stretched both horizontally and vertically**, the image size is not a problem.

- Before creating your first Popup, **define CSS rules to customize the component appearance**. The default CSS classes and ids used by the Popup component are the following (but can be easily customized by modifying the code, see implementation for details):
 - ◆ **popup** CSS class for the main Popup DOM element
 - ◆ **popup_text** CSS class for the Popup content
 - ◆ **popup_buttons** CSS class for the Popup buttons' container DIV
 - ◆ **popup_button_ok** ID for the OK button
 - ◆ **popup_button_cancel** ID for the CANCEL button
- Now, all is ready to display the first Popup window. To do it, just **instantiate a new Popup object, and then show it**:

```
var popup = new Popup(
  "This is just a <strong>sample popup</strong>",
  "OK",
  null,
  "Cancel",
  null);

popup.show();
```

Arguments for the Popup constructor are the following:

- the **'popup content** (any valid HTML content)
- the **text for the OK button**
- the **handler function** for the OK button (can be null)
- the **text for the CANCEL button (optional**: if null, the cancel button will not be shown)
- the **handler function** for the CANCEL button (**optional**, can be null)

Sample CSS rules, used in this article, are the following:

```
.popup {
border: 3px solid #336699;
padding: 4px;
background: white;
text-align:center;
}

.popup_buttons {
overflow: auto;
}

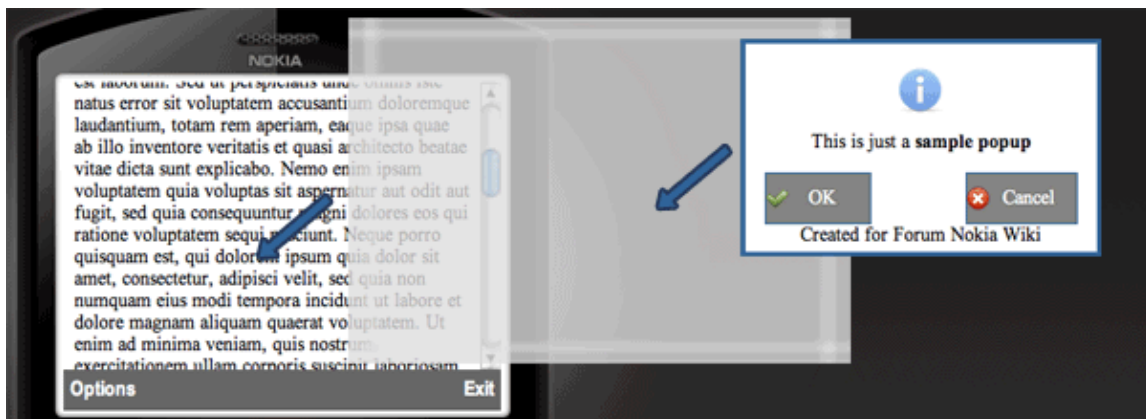
#popup_button_ok, #popup_button_cancel {
border: 1px solid #336699;
background-color: gray;
color:white;
min-width: 50px;
text-align:center;
padding: 10px;
}

#popup_button_ok:hover, #popup_button_cancel:hover {
background-color: #6699cc;
```

```
}  
  
#popup_button_ok {  
float: left;  
margin-left: 10px;  
background-image: url("images/accept.png");  
background-position: left center;  
padding-left: 16px;  
background-repeat: no-repeat;  
}  
  
#popup_button_cancel {  
float: right;  
margin-right: 10px;  
background-image: url("images/cancel.png");  
background-position: left center;  
padding-left: 20px;  
background-repeat: no-repeat;  
}
```

Popup component implementation

The Popup component is structured as shown by the following picture.



So, once the WRT widget needs to show a Popup instance, the component will perform these steps:

- **append the semi-opaque image over the Widget's content**, and stretch it to cover its whole size
- **build the popop DOM structure**
- **append the popup DOM element just over the cover image**, centering it both horizontally and vertically

Since the image will cover the whole Widget user interface, user interactions will be forbidden until the popup, and so the image cover, will be disposed.

Now, let's see the component implementation.

Popup constructor

This Popup component built in this article has customizable features:

- customizable popup content
- customizable OK and CANCEL buttons' text
- customizable handler functions on the OK and CANCEL buttons' click events

The Popup constructor allows to set all these properties through its arguments:

```
function Popup(popupHtml, okButtonText, okButtonHandler, cancelButtonText, cancelButtonHandler)
{
  /* HTML content to be shown inside the popup */
  this.popupHtml = popupHtml;

  /* text for the ok button */
  this.okButtonText = okButtonText;

  /* text for the cancel button */
  this.cancelButtonText = cancelButtonText;

  /* handler function for the ok button */
  this.okButtonHandler = okButtonHandler;

  /* handler function for the cancel button */
  this.cancelButtonHandler = cancelButtonHandler;

  /* page cover DOM element */
  this.coverElement = null;

  /* popup DOM element */
  this.popupElement = null;
}
```

The last 2 defined properties will hold references to the image cover and popup DOM elements, respectively.

Building and showing the Popup

Before building the popup DOM structure, some properties are defined in order to allow easier styling and customization of Popup visual appearance.

```
/* CSS class applied to the main popup element */
Popup.MAIN_CSS_CLASS = 'popup';

/* CSS class applied to popup content */
Popup.TEXT_CSS_CLASS = 'popup_text';

/* CSS class applied to popup buttons' container */
Popup.BUTTONS_CSS_CLASS = 'popup_buttons';

/* path of the image used to opacize the widget */
Popup.COVER_IMAGE_SRC = 'images/page_cover.png';

/* element ID of the OK button */
Popup.OK_BUTTON_ID = 'popup_button_ok';
```

Popup_JavaScript_component_for_Web_Runtime

```
/* element ID of the CANCEL button */
Popup.CANCEL_BUTTON_ID = 'popup_button_cancel';

/* minimum margin of the popup from screen boundaries */
Popup.MARGIN = 20;

/* z-index to be used for the popup DOM element */
Popup.zIndex = 10000;
```

The Popup DOM structure is built on the fly when the Widget needs to display it. So, all the job is done by the **show()** method:

```
Popup.prototype.show = function(beforeShowHandler)
{
  /* calculate the width and height to be covered */
  var bodyHeight = Math.max(document.body.offsetHeight, window.innerHeight);
  var bodyWidth = document.body.offsetWidth;

  /* build the cover that will opacize the widget's content */
  var coverElement = document.createElement('img');
  coverElement.src = Popup.COVER_IMAGE_SRC;
  coverElement.position = 'absolute';
  coverElement.width = '100%';
  coverElement.height = bodyHeight + 'px';
  coverElement.top = '0px';
  coverElement.left = '0px';
  coverElement.zIndex = Popup.zIndex - 1;

  this.coverElement = coverElement;

  /* append the cover element to the document's body */
  document.body.appendChild(coverElement);

  /* now, build the popup DOM element */
  var popupElement = document.createElement('div');
  popupElement.position = 'absolute';
  popupElement.width = (bodyWidth - 2 * Popup.MARGIN) + 'px';
  popupElement.left = (document.body.scrollLeft + Popup.MARGIN) + 'px';
  popupElement.zIndex = Popup.zIndex;
  popupElement.className = Popup.MAIN_CSS_CLASS;

  this.popupElement = popupElement;

  /* build the DOM element that contains the popup content */
  var textElement = document.createElement('div');
  textElement.className = Popup.TEXT_CSS_CLASS;
  textElement.innerHTML = this.popupHtml;
  popupElement.appendChild(textElement);

  /* add the ok and cancel buttons */
  var buttonsContainer = document.createElement('div');
  buttonsContainer.className = Popup.BUTTONS_CSS_CLASS;
  popupElement.appendChild(buttonsContainer);

  var okButton = document.createElement('div');
  okButton.id = Popup.OK_BUTTON_ID;
  okButton.innerHTML = this.okButtonText;
  buttonsContainer.appendChild(okButton);

  var self = this;
```

Popup_JavaScript_component_for_Web_Runtime

```
        okButton.onclick = function()
    {
        okButtonPressed();
    }

    if(this.cancelButtonText != null)
    {
        var cancelButton = document.createElement('div');
            id = cancelButtonID;
            innerHTML = this.cancelButtonText;
            appendChild(cancelButton);

            onclick = cancelButtonFunction()
    {
        cancelButtonPressed();
    }
    }

    if(beforeShowHandler)
        beforeShowHandler();

    /* finally, append the popup element to the document's body */
    document.body.appendChild(popupElement);

    /* let's center the popup */
    var popupTop = Math.max(
        (document.body.scrollTop + (window.innerHeight - popupElement.offsetHeight) / 2),
        document.body.scrollTop);

    popupElement.style.top = popupTop + 'px';
}
```

The **show()** method accepts an optional function handler as argument: if passed, it will be called just before the Popup DOM element is actually shown, in order to allow further customization of the component itself.

The steps performed by the **show()** method are:

- calculating the whole widget width (**bodyWidth**) and height (**bodyHeight**), in pixels
- building the cover image (**coverElement**) and appending over the widget's content
- creating the popup DOM element (**popupElement**) and then populating it with its content:
 - ◆ the popup HTML content (**textElement**)
 - ◆ and the popup OK and CANCEL buttons (**okButton** and **cancelButton**)
 - ◆ onclick handler functions are also defined for the OK and CANCEL button, in order to respond to user interaction
- before showing the component, the **beforeShowHandler**, if defined, is called
- finally, the component is shown and its top coordinate is adjusted, to center it vertically

Popup buttons' click handlers

The following methods are used to **manage user interaction for the OK and CANCEL buttons**, respectively. If custom handlers were defined through the component's constructor, they'll be called as well.

```
/**
 * Function called when the popup CANCEL button is pressed
 */
```

Popup_JavaScript_component_for_Web_Runtime

```
Popup.prototype.cancelButtonPressed = function()
{
this.hide();

if(this.cancelButtonHandler != null)
this.cancelButtonHandler();
}
/**
 * Function called when the popup OK button is pressed
 */
Popup.prototype.okButtonPressed = function()
{
this.hide();

if(this.okButtonHandler != null)
this.okButtonHandler();
}
```

Hiding the component

Once the user has seen the popup, and wants to dispose it (through one of the OK and CANCEL button), the **hide()** method will be called to actually **dispose the component**:

```
/**
 * Function to hide the popup
 */
Popup.prototype.hide = function()
{
this.popupElement.parentNode.removeChild(this.popupElement);
this.coverElement.parentNode.removeChild(this.coverElement);

this.popupElement = null;
this.coverElement = null;
}
```

Downloads

The following files are available for download:

- A sample Widget using the Popup component: [Media:PopupWidget.zip](#)
- Popup component JavaScript code: [Media:Wrt_Popup_component.zip](#)