

This article shows **how to create an image gallery by using JavaScript and Platform Services** in a Web Runtime widget. Since it uses Platform Services, this code works with **Web Runtime version 1.1**.



Contents

- [1 ImageGallery component: how to use it](#)
- [2 ImageGallery implementation](#)
 - ◆ [2.1 Constructor](#)
 - ◆ [2.2 Building DOM structure](#)
 - ◆ [2.3 Showing and hiding the image loader](#)
 - ◆ [2.4 Loading images from device gallery with Platform Services](#)
 - ◇ [2.4.1 The GetList\(\) callback handler](#)
 - ◆ [2.5 Displaying gallery images](#)
 - ◇ [2.5.1 Image element onload event](#)
 - ◆ [2.6 Navigating through the gallery images](#)
 - ◆ [2.7 Selecting the current image](#)
- [3 Considerations](#)
- [4 Downloads](#)

ImageGallery component: how to use it

In order to use the ImageGallery component in a Web Runtime widget, the following steps are necessary:

- **Import the ImageGallery.js JavaScript file** into your code (you can download it from this link: [Media:Wrt_imagegallery_component.zip](#)):

Web_Runtime_image_gallery_JavaScript_component

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

- **Define an HTML element** to be used as a container for the image gallery:

```
<body>
[... ]
<div id="gallery_holder">

</div>
[... ]
</body>
```

- add to your project these **4 images**:
 - ◆ 1 for the 'next image' button
 - ◆ 1 for the 'previous image' button
 - ◆ 1 for the 'select image' button
 - ◆ and 1 to be used as the image loader

The paths of these images are defined in the implementation part of this article, and can be customized as well. Sample images, used in this article, are shown below, and can be downloaded from this link:

[Media:Wrt_imagegallery_sampleimages.zip?](#)



- **define a handler function** to be called **when the user select an image**. This handler will be called with the photo path as argument. A sample implementation, that uniquely alerts the image path, is the following:

```
function myImageHandler(imagePath)
{
  alert(imagePath);
}
```

- Finally, to create and show the image gallery, **create an ImageGallery instance** with these parameters:
 - ◆ the **parent HTML element**, defined in your HTML code
 - ◆ the **gallery width** (in pixels)
 - ◆ the **gallery height** (in pixels)

Then, **set the previously defined image handler**, by setting the **imagePickHandler** property, and then call the **load()** method on the ImageGallery instance:

```
var gallery = new ImageGallery(document.getElementById('gallery_holder'), 300, 400);
gallery.imagePickHandler = myImageHandler;
gallery.load('FileDate', false);
```

The **load()** method accepts 2 arguments:

- **the sort key**, that must be one of the keys defined on [this Forum Nokia Library page](#) (for the **criteria.Sort.Key** property)

- a **boolean value**, defining if the **sort order** must be ascending (true) or descending (false)

ImageGallery implementation

Constructor

The ImageGallery constructor just defines some properties, and then call an initialization method, **init()**, that will be defined below.

```
function ImageGallery(parentElement, galleryWidth, galleryHeight)
{
  /* max image size */
  this.maxImageHeight = 0;
  this.maxImageWidth = 0;

  /* DOM elements */
  this.imageElement = null;
  this.imageLoader = null;

  /* library images */
  this.libraryImages = null;
  this.currentPhotoIndex = -1;

  /* image pick handler */
  this.imagePickHandler = null;

  this.init(parentElement, galleryWidth, galleryHeight);
}
```

Building DOM structure

Before proceeding, it's useful to define some properties that will be used for the DOM structure of this component. So, let's define:

```
ImageGallery.ICON_HEIGHT = 64;
ImageGallery.PREV_IMAGE_SRC = 'images/arrow_left.png';
ImageGallery.NEXT_IMAGE_SRC = 'images/arrow_right.png';
ImageGallery.PICK_IMAGE_SRC = 'images/pick_image.png';
ImageGallery.IMAGE_LOADER = 'images/ajax-loader.gif';
```

Done this, it's time to build the DOM structure of the ImageGallery component. The following **init()** method will do this:

```
ImageGallery.prototype.init = function(parentElement, galleryWidth, galleryHeight)
{
  var self = this;

  /* maximum size of gallery images */
  this.maxImageHeight = galleryHeight - ImageGallery.ICON_HEIGHT;
  this.maxImageWidth = galleryWidth;

  /* main gallery element */
  var el = document.createElement('div');
```

Web_Runtime_image_gallery_JavaScript_component

```
style.height = galleryHeight + 'px';
style.width = galleryWidth + 'px';
className = 'image_gallery';
parent.appendChild(el);

/* image element */
var imgContainer = document.createElement('div');
imgContainer.style.textAlign = 'center';
imgContainer.style.overflow = 'hidden';
imgContainer.style.width = this.maxImageWidth + 'px';
imgContainer.style.height = this.maxImageHeight + 'px';
parent.appendChild(imgContainer);

/* image loader, to be shown while loading a new image */
var loader = document.createElement('img');
loader.src = ImageGallery.IMAGE_LOADER;
loader.style.display = 'none';
imgContainer.appendChild(loader);
this.imageLoader = loader;

/* image element, will contain the current shown image */
var img = document.createElement('img');
img.style.margin = 'auto';
imgContainer.appendChild(img);
this.imageElement = img;

/* buttonsContainer will contain the buttons used to perform gallery actions */
var buttonsContainer = document.createElement('div');
buttonsContainer.style.textAlign = 'center';
buttonsContainer.className = 'image_gallery_buttons';
parent.appendChild(buttonsContainer);

/* button to go to the previous gallery image */
var prevImageButton = document.createElement('img');
prevImageButton.src = ImageGallery.PREV_IMAGE_SRC;
buttonsContainer.appendChild(prevImageButton);

/* button to go to the pick the current gallery image */
var pickImageButton = document.createElement('img');
pickImageButton.src = ImageGallery.PICK_IMAGE_SRC;
buttonsContainer.appendChild(pickImageButton);

/* button to go to the next gallery image */
var nextImageButton = document.createElement('img');
nextImageButton.src = ImageGallery.NEXT_IMAGE_SRC;
buttonsContainer.appendChild(nextImageButton);
}
```

Showing and hiding the image loader

As seen in the DOM structure, an image loader will be shown while the next (or previous) image is loading from the device gallery. So, let's define 2 methods that will be used to show and hide the image loader:

```
ImageGallery.prototype.showLoader = function()
{
this.imageLoader.style.display = '';
}
ImageGallery.prototype.hideLoader = function()
{
```

```
this.imageLoader.style.display = 'none';  
}
```

Loading images from device gallery with Platform Services

WRT 1.1 offers access to device media through the Media Management Service API. The following **load()** method uses this API to retrieve the images stored in the device's gallery:

```
ImageGallery.prototype.load = function(sortBy, ascending)  
{  
  if(sortBy == undefined || sortBy == null)  
    sortBy = 'FileName';  
  
  var sortType = ascending ? 'Ascending' : 'Descending';  
  
  this.showLoader();  
  
  var so = device.getServiceObject("Service.MediaManagement", "IDataSource");  
  
  var criteria =  
  {  
    'Type': 'FileInfo',  
    'Filter':  
    {  
      'FileType': 'Image'  
    }  
  };  
  
  var self = this;  
  
  var result = so.IDataSource.GetList(  
    , criteria  
  function(transId, eventCode, result)  
  {  
    libraryImagesCallback(transId, eventCode, result);  
  }  
  );  
  
  if(result.ErrorCode != 0)  
  {  
    this.showError(result.ErrorMessage);  
  }  
}
```

The above code, after having **initialized the Service Object**, and created the **criteria object** to be used to retrieve the **Image** files, call the **GetList()** method in asynchronous mode, using the **libraryImagesCallback()** instance method as callback function.

The GetList() callback handler

The **libraryImagesCallback()** will perform these operations:

- iterate through all the images returned by the **GetList()** method
- store the image paths in the **libraryImages** instance variable

Web_Runtime_image_gallery_JavaScript_component

```
ImageGallery.prototype.libraryImagesCallback = function(transId, eventCode, result)
{
  if(eventCode != 4 && result.ErrorCode == 0)
  {
    this.libraryImages = new Array();

    var iterator = result.ReturnValue;

    var item = null;

    while((item = iterator.getNext()) != undefined)
    {
      var imagePath = item.FileNameAndPath.replace(/\\/g, "/");

      = 'file:///'+imagePath;

      this.libraryImages.push(imagePath);
    }
    if(this.libraryImages.length > 0)
    {
      this.showLibraryImage(0);
    }
    else
    {
      this.showError('No images found in device Gallery.');
```

Error management, for the purposes of this article, is performed by the **showError()** method, that will uniquely alert the error message passed as argument:

```
ImageGallery.prototype.showError = function(txt)
{
  alert(txt);
}
```

Displaying gallery images

If the **GetList()** method of **MediaManagement Service API** returns at least an image, then the first image is shown, by calling the **showLibraryImage()** method. This method:

- hides the current displayed image
- shows the image loader, by calling the **showLoader()** method defined above
- sets the current image index
- resets the image element width and height, by using the *auto* value
- finally, sets the *src* attribute of the image element to the path of the current image

```
ImageGallery.prototype.showLibraryImage = function(imageIndex)
{
  this.imageElement.style.display = 'none';

  this.showLoader();
```

The **GetList()** callback handler

Web_Runtime_image_gallery_JavaScript_component

```
this.currentPhotoIndex = imageIndex;

this.imageElement.style.width = 'auto';
this.imageElement.style.height = 'auto';

this.imageElement.src = this.libraryImages[this.currentPhotoIndex];
}
```

Image element onload event

When the current image has been loaded, it's necessary to show it (because the image element was hidden by the **showLibraryImage()** method), and to hide the image loader. Also, we want to adjust the image size to fit the available size reserved to the gallery. To do this, it's necessary to be notified when the image has been completely loaded, and this can be done by using the **onload** event. So, let's take back the **init()** method and define this event handler:

```
ImageGallery.prototype.init = function(parentElement, galleryWidth, galleryHeight)
{
  [...]

  onload = function()
  {
    imageLoaded.call(this);
  };
}
```

The **imageLoaded()** is the instance method that will handle the operations defined above, and is implemented as follows:

```
ImageGallery.prototype.imageLoaded = function()
{
  this.hideLoader();

  this.imageElement.style.display = '';

  var imageWidth = this.imageElement.offsetWidth;
  var imageHeight = this.imageElement.offsetHeight;

  var widthProp = imageWidth / this.maxImageWidth;
  var heightProp = imageHeight / this.maxImageHeight;

  var resizeProp = Math.max(widthProp, heightProp);

  if(resizeProp > 1)
  {
    this.imageElement.style.width = (imageWidth / resizeProp) + 'px';
    this.imageElement.style.height = (imageHeight / resizeProp) + 'px';
  }

}
```

This method:

- hides the image loader
- displays the image element

- check if the new loaded image is greater than the available size and, if so, scales it down maintaining the image proportions

Navigating through the gallery images

Now that the first image is displayed, the user has to be able to navigate to the next ones. To do this, we have to add some behavior to the **nextImageButton** and **prevImageButton** defined by the **init()** method:

```
ImageGallery.prototype.init = function(parentElement, galleryWidth, galleryHeight)
{
  [...]

  prevImageButton = function()
  {
    viewLibraryImage(-1);
  };

  nextImageButton = function()
  {
    viewLibraryImage(1);
  };
}
```

Both methods will call the same instance method, **viewLibraryImage()**, that accepts as argument the index delta of the next photo to be shown. The **viewLibraryImage()** method can be implemented as follows:

```
ImageGallery.prototype.viewLibraryImage = function(indexDelta)
{
  var newIndex = this.currentPhotoIndex + indexDelta;

  if(newIndex >= 0 && newIndex < this.libraryImages.length)
  {
    this.showLibraryImage(newIndex);
  }
}
```

Selecting the current image

Finally, the user has to be able to select the current image, and your widget has to be notified of this, in order to perform further operations on it. So, we have previously defined the **imagePickHandler** property, that holds a reference to a custom handler function that can be freely defined in your code. To call this function, let's add a behavior to the **pickImageButton** defined in the **init()** method:

```
ImageGallery.prototype.init = function(parentElement, galleryWidth, galleryHeight)
{
  [...]

  pickImageButton = function()
  {
    chooseLibraryImage();
  };
}
```

So, on the click event, the **pickImageButton** will call the **chooseLibraryImage()** defined as follows:

Web_Runtime_image_gallery_JavaScript_component

```
ImageGallery.prototype.chooseLibraryImage = function()
{
if(this.imagePickHandler != null && this.currentPhotoIndex >= 0)
{
this.imagePickHandler(this.libraryImages[this.currentPhotoIndex]);
}
}
```

Considerations

- It would be nice to have an image gallery with more images' thumbs shown at once. But, since WRT memory is limited, an images stored in device's gallery can be quite big, the best approach, for both memory and performance issues, is to show one image at a time.
- As a further improvement to this component, it would be nice to implement touch gestures to move from an image to the next/previous.

Downloads

The following resources, related to this article, can be downloaded:

- A **sample Widget using the ImageGallery component**: [Media:Wrt_imagegallery_widget.zip](#)
- The **ImageGallery component JavaScript code**: [Media:Wrt_imagegallery_component.zip](#)