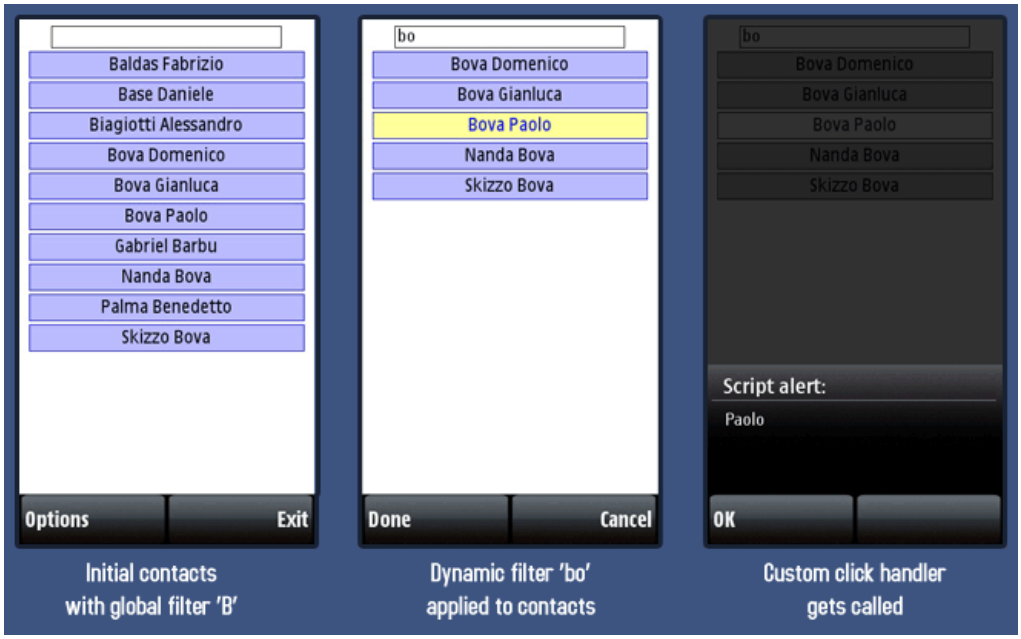
Contacts component for Symbian Web Runtime

This articles shows a Contacts component for Web Runtime widgets, with the following features:

- customizable user interface and visualized contacts' information
- customizable error and click handlers
- support for dynamic filtering of contacts

Introduction

The Contacts component is a customizable user interface component that allows listing, visualization, filtering and handling of contacts retrieved from the device phonebook.



Contacts component: how to use it

In order to use the the Contacts component, these steps are necessary:

JavaScript component library

First thing to do, is to include the `ContactsComponent` JavaScript file (available here: Media:Wrt ContactsComponent.zip) in the widget's code:

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

Defining an HTML node for the component

Before actually creating the Contacts component, it is necessary to define a HTML element that will contain the component's data. An example is given by the following DIV element:

```
<html>
<!-- init function is called as soon as the widget starts -->
 <body onLoad="javascript:init();">
  [...]
<input type="text" name="contacts_filter" id="contacts_filter"
onkeypress="javascript:init();" />
  <div id="contacts_view">

  </div>
```


(C) Copyright Nokia 2014. All rights reserved.

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

ContactsComponent instantiation

In order to create a new Contacts component, the `ContactsComponent` constructor must be called with the following arguments:

- a DOM element where contacts will be appended (e.g.: a DIV element)
- a template string (see section below), to specify the contact's data to be visualized
- a handler function to be called on click events on contact elements

A sample instantiation of a `ContactsComponent` is visible in the following code snippet:

```
var contacts = new ContactsComponent(
 document.getElementById('contacts_view'),
 '{LastName} {FirstName}',
 handleContact
);
```

Contact template string

Contacts' data visualization is customizable through a template string, that specifies the exact information to show for each contact. Template strings can include custom HTML code, in order to allow further customization. Sample template strings are the following:

- "{LastName} {FirstName}": to display last and first name of each contact
- "{LastName} {FirstName}, {MobilePhoneGen}" to display last and first name, and the general mobile phone in empathized style.

For a complete list of allowed template keys, check out this Nokia Developer Library page:

- Supported contact keys

The handler function

The handler function is called when the user clicks on a contact element. The component will pass to the handler the contact data as argument. The data format of the contact information is the one returned by the Contacts Service API, and is detailed in this Nokia Developer Library page:

- Contact data format

A sample handler function, that alerts the chosen contact's first name, is the following:

```
function handleContact(contact)
{
 alert(contact.FirstName.Value);
}
```

Setting the error handler

When an error occurs in the Contacts component, a custom handler function can be called to appropriately show the error information to the user. In order to set a custom error handler, the `setErrorHandler()` method must be called with the handler function passed as argument. The error handler, when called, receives as argument the error text message. A sample error handler, that alert the error message, is the following:

```
function handleError(errorMessage)
{
 alert(errorMessage);
}
```


(C) Copyright Nokia 2014. All rights reserved.

And in order to set this handler to the Contacts component created before, the following code will work:

```
contacts.setErrorHandler(handleError);
```

Initializing the component

Once created, the component must be initialized, so that it actually loads and shows the contacts' information, and this is done by the `initialize()` method. The `initialize()` method accepts an optional string argument: if defined, this argument will be used to filter the data loaded from the device phonebook. If no argument is passed to the initialize() method, all the contacts will be loaded and shown, as in the code snippet below:

```
contacts.initialize();
```

Dynamic filtering of contacts

Once initialized, all the loaded contacts are shown to the user. In order to allow the user to dynamically filter the contacts, it is possible to set an input field as filter for the displayed contacts. Once it is set as filter, the component automatically uses the input field value as filter for the displayed contacts, matching its value with a customizable set of fields. Default searched fields are the contact's first and last name. So, to set an input field as filter for the contacts component:

- define an INPUT element in widget's code:

```
<input type="text" name="contacts_filter" id="contacts_filter"
onkeypress="javascript:init();" />
<!-- Onkyepress even is added to filter the search list in the contact for pattern
matching. -->
```

- set the field as filter for the Contacts component, through the `setFilterInput()` field:

```
contacts.setFilterInput(document.getElementById('contacts_filter'));
```

- in order to customize the fields where the contact searches for the filter text, call the `setFilterFields()` with the field names as argument:

```
contacts.setFilterFields(new Array('FirstName', 'MobilePhoneHome'));
```

When applying the dynamic filter, the global filter, passed as argument to the `initialize()` method is still active, so both the filters are actually applied to the contact list.

Complete usage code

The complete code needed to use and customized the Contacts component is summarized by the following snippet:

```
var contacts = new ContactsComponent(
 document.getElementById('contacts_view'),
 '{LastName} {FirstName} <div class="contact_detail">{MobilePhoneGen}</div>',
 handleContact
);

contacts.setErrorHandler(handleError);

contacts.setFilterInput(document.getElementById('contacts_filter'));
```


(C) Copyright Nokia 2014. All rights reserved.

```
contacts.setFilterFields(new Array('FirstName', 'LastName', 'MobilePhoneGen'));

contacts.initialize();

function handleError(errorMessage)
{
 alert(errorMessage);
}
function handleContact(contact)
{
 alert(contact.FirstName.Value);
}
```

Contacts component implementation

ContactsComponent constructor

The constructor, as seen in the previous section, accept three arguments:

- the DOM element that will contain the contacts
- the template string
- the contacts' click handler

So, the constructor stores these 3 arguments in three instance variables, and defines the other following properties:

```
function ContactsComponent(contactsElement, templateString, clickHandler)
{
 /* contact item template */
 this.contactTemplate = templateString;

 /* contact items DOM container */
 this.contactsElement = contactsElement;

 /* handler function for click event */
 this.clickHandler = clickHandler;

 /* contacts list */
 this.contacts = null;

 /* filtered contacts list */
 this.textFilter = null;

 /* fields used to filter contacts */
 this.filterFields = new Array('FirstName', 'LastName');

 /* input filter field */
 this.filterInput = null;

 /* error handler function */
 this.errorHandler = null;
}
```

Error management

In order to appropriately manage errors, two functions are defined:

- `setErrorHandler()`: to allow customization of the error handler function

(C) Copyright Nokia 2014. All rights reserved.

- `notifyError()`: that is called by the component when some errors occur, and that calls the custom error handler

```
ContactsComponent.prototype.setErrorHandler = function(handler)
{
 this.errorHandler = handler;
}
ContactsComponent.prototype.notifyError = function(errorMessage)
{
 if(this.errorHandler)
 {
  this.errorHandler(errorMessage);
 }
}
```

Loading contacts data

The `initialize()` method takes care of all the Contacts Service API related calls, in order to load the contacts from the device phonebook.

```
ContactsComponent.prototype.initialize = function(globalFilter)
{
 if(device)
 {
  var contactsService = device.getServiceObject("Service.Contact", "IDataSource");

  var criteria =
  {
   'Type': 'Contact'
  };

  if(globalFilter)
  {
   criteria.Filter =
   {
    'SearchVal': globalFilter
   }
  };

  var self = this;

  var result = contactsService.IDataSource.GetList(
   criteria,
   function(transId, eventCode, result)
   {
    self.retrieveContactsHandler(transId, eventCode, result);
   }
  );

  if(result.ErrorCode != 0)
  {
   this.notifyError(result.ErrorMessage);
  }
 }
 else
 {
```


(C) Copyright Nokia 2014. All rights reserved.

```
  this.notifyError("Contacts component not supported");
 }
}
```

If some error occurs while trying to load the contacts' data, the notifyError() method is called with an appropriate error message. If the `GetList()` method of Contacts Service API is correctly called, it'll call the `retrieveContactsHandler()` function defined as follows:

```
ContactsComponent.prototype.retrieveContactsHandler = function(transId, eventCode,
result)
{
 if(eventCode != 4)
 {
  if(result.ErrorCode != 0)
  {
   this.notifyError(result.ErrorMessage);
  }
  else
  {
   var contact;

   var iterator = result.ReturnValue;

   this.contacts = new Array();

   while((contact = iterator.getNext()) != undefined)
   {
    this.contacts.push(contact);
   }
   this.showContacts();
  }
 }
 else
 {
  this.notifyError("There was an error while retrieving contacts.");
 }
}
```

The handler, in case of errors, also calls the `notifyError()` method, otherwise it stores in the contacts property all the contacts returned by the Contacts Service API.

Once the contacts' data is stored, the `showContacts()` method is called in order to actually show the contacts.

Showing the contacts data

When the component needs to show the contacts' data, it has to perform these steps:

- loop all the loaded contacts
- check, if a dynamic filter is defined, if the contact matches that filter
- if it matches, the component formats the contact label by using the template string. If multiple values are defined for a single property, the values are concatenated with separating commas
- then, an onclick event handler is defined, in order to allow the contact element to be clickable by the user
- finally, the contact data appended to the DOM container

```
ContactsComponent.prototype.showContacts = function()
{
 while(this.contactsElement.firstChild)
```


(C) Copyright Nokia 2014. All rights reserved.

```
  {
   this.contactsElement.removeChild(this.contactsElement.firstChild);
  }

  var self = this;

  for(var i = 0; i < this.contacts.length; i++)
  {
   if(this.textFilter == null || this.contactMatchesFilter(this.contacts[i]))
   {
    var contactLabel = this.contactTemplate;

    for(key in this.contacts[i])
    {
     var value = this.contacts[i][key].Value;

     var next = this.contacts[i][key].Next;

     while(next)
     {
      value += ', ' + next.Value;

      next = next.Next;
     }

     contactLabel = contactLabel.replace('{' + key + '}', value);
    }
    contactLabel = contactLabel.replace(/\{.+?\}/, '');

    var contactElement = document.createElement('div');

    contactElement.contactIndex = i;

    contactElement.className = ContactsComponent.CONTACT_CSS_CLASS;

    contactElement.innerHTML = contactLabel;

    contactElement.onclick = function()
    {
     if(self.clickHandler)
     {
      self.clickHandler(self.contacts[this.contactIndex]);
     }
    }

    this.contactsElement.appendChild(contactElement);
   }
  }
 }
```

The `ContactsComponent.CONTACT_CSS_CLASS` property contains the CSS class name that is applied to all the contact elements, so allowing further customization:

```
ContactsComponent.CONTACT_CSS_CLASS = 'contact';
```


(C) Copyright Nokia 2014. All rights reserved.

Dynamic filtering of contacts

When a text input field is set as filter for the contacts component, it is automatically used by the component itself to filter the loaded contacts. The following methods allow to appropriately manage and use the filter input field:

- setFilterFields(): defines the field names to be used to match the current filter value

```
ContactsComponent.prototype.setFilterFields = function(fieldsArray)
{
 this.filterFields = fieldsArray;
}
```

- `setFilterInput()`: sets the text input field, passed as argument, as dynamic filter for the component. It does so by attaching appropriate events to the field, so that the typed text is automatically used to filter the contacts, without the need for the user to press any buttons or perform other user interactions.

```
ContactsComponent.prototype.setFilterFields = function(fieldsArray)
{
 this.filterFields = fieldsArray;
}
ContactsComponent.prototype.setFilterInput = function(inputField)
{
 var self = this;

 if(this.filterInput != null)
 {
  this.filterInput.onchange = undefined;
 }
 this.filterInput = inputField;

 var inputHandler = function()
 {
  self.setFilterText(this.value);
 }

 this.filterInput.onkeyup = inputHandler;
 this.filterInput.onkeydown = inputHandler;
 this.filterInput.onchange = inputHandler;
}
```

- setFilterText(): called by the input field event handlers, performs the actual filtering of contacts, by setting the new filter property value, and by calling the `showContacts()` method to refresh the component's content:

```
ContactsComponent.prototype.setFilterText = function(filterText)
{
 if(this.textFilter != filterText)
 {
  this.textFilter = filterText;

  this.showContacts();
 }
}
```

- contactMatchesFilter(): check if the contact passed as argument matches with the current filter value. This method works by checking the contact properties defined in the filterFields property, and if multiple values are defined for a single property, all

(C) Copyright Nokia 2014. All rights reserved.

the single values are checked.

```
ContactsComponent.prototype.contactMatchesFilter = function(contact)
{
 var regExp = new RegExp(this.textFilter, "i");

 for(var i = 0; i < this.filterFields.length; i++)
 {
  var current = contact[this.filterFields[i]];

  while(current)
  {
   if(regExp.test(current.Value))
   {
    return true;
   }

   current = current.Next;
  }
 }
 return false;
}
```

Downloads

These related resources are available for download:

- A sample widget that uses the Contacts component: Media:ContactsComponentWidget.zip
- The Contacts component JavaScript source code: Media:Wrt ContactsComponent.zip

(C) Copyright Nokia 2014. All rights reserved.