Forerunner Mobilizer RDL Extensions

Introduction

The Forerunner Mobilizer RDL extension enable new functionality in the report definition when using the Forerunner HTML5 render. These extensions are currently settable via the RDL Extension dialog in Mobilizer or by saving a Report Server Property names "`ForerunnerRDLExt`" via the SSRS API. The RDL extension format is a JSON object where the name of the RDL element that is being extended is a member of the object and that elements extension settings are in an object value of the member.

Report Item Extension

You can give any report item an html ID or Class. These are using to find the item via JavaScript or to style the element.

```
{
 "Textbox1": {
  "ID": "Textbox1",
  "Class": "MyClass"
 }
}
```

Rectangle Extensions

Rectangles can be extended to allow for fixed sized content, represent a Form, become an IFrame or embed HTML.

Fixed sizing forces the rectangle to take up a specific amount of space and if its content is larger to scroll within the fixed size. You can fix the width, height or both. When a dimension is fixed then the render will display a scroll bar on that dimension. The height and width may be specified in the following units: px, in, mm, em, pt or cm. If no unit is specified it will be assumed to be px.

Examples

You can specify a fixed width or height by setting the FixedWidth or FixedHeight property.

```
{
 "Rectangle1": {
  "FixedWidth": "3.7in",
  "FixedHeight": "4in"
 }
}
```

To turn a rectangle into an IFrame set the IFrameSrc property. Optioinaly set the IFrameSeamless property to remove the iFrame styling.

```
{
 "Rectangle1": {
  "IFrameSrc": "http://msn.com",
  "IFrameSeamless": true
```

```
 }
}
```

To turn a rectangle into an HTML Frame set the FormAction, FormName and FormMethod properties. This is useful if you want to easy submit a form of input elements

```
{
 "Rectangle1": {
  " FormAction": "http://msn.com",
  " FormName": "Form1",
  " FormMethod ": "POST"
 }
}
```

To embed custom HTML in the rectangle set the CustomHTML property.

```
{
 "Rectangle1": {
  "CustomHTML": "<div>Hello World</div>"
 }
}
```

Textbox Extensions

Textboxs may be extended to become any type of HTML input element. This allows the user to modify the values in the report and with used with a rectangle form, EasySubmit or JavaScript action you can post back these changes to your web service or other script in your application.

Textboxes can now also have justified text.

Examples

To make a texbox have justified text.

```
{
 "Testbox1": {
  "Align": "justify"
 }
}
```

To make a texbox an input element set the Input Properties.

- InputType: any valid HTML input type or textarea for multiline input
- InputName: the name you want to use for the property on a submit
- InputSubmit: can be “always”, “never” or “changed”. Used to determine if the property should be always, never or conditionally included in a EasySubmit or when using “auto” when calling the getImputs function

```
{
 "CustomerID": {
  "InputType": "text",
  "InputSubmit": "always",
  "InputName": "CustID",
```

```
  "InputReadOnly": true
 },
 "LastName": {
  "InputType": "text",
  "InputName": "LastName",
  "InputSubmit": "changed"
 },
 "FirstName": {
  "InputType": "text",
  "InputName": "FirstName",
  "InputSubmit": "changed"
 },
 "Phone": {
  "InputType": "text",
  "InputName": "Phone",
  "InputSubmit": "changed"
 },
 "Address": {
  "InputType": "textarea",
  "InputName": "Address",
  "InputSubmit": "changed"
 },
 "Call": {
  "InputType": "checkbox",
  "InputName": "Call",
  "InputSubmit": "changed"
 }
}
```

To simulate a form in a tablix row without writing any code you can use the EasySubmit properties.

- EasySubmitType: GET or a POST
- EasySubmitURL: the URL to submit to
- EasySubmitDatatype: text or json, text will be comma delimited name=value pairs, jso will be a json object
- EasySubmitSuccess: string for success message
- EasySuccessFail: string for fail message

```
{
"SaveCust": {
  "InputType": "button",
  "InputName": "saveCust",
  "InputSubmit": "never",
  "EasySubmitType": "POST",
  "EasySubmitURL": "/api/saveCust",
  "EasySubmitDatatype": "json",
```

```
  "EasySubmitSuccess": "Saved customer",
  "EasySuccessFail": "Failed to Save customer"
 }
}
```

Tablix Extensions

A Tablix can be extended to customize the responsive UI rules when re-flowing the layout of a Tablix when response UI is enabled. By default the column headers are assumed to be in the first row of the report (row 1) and that columns will be hidden starting from the right. These rules can be customized to allow the headers to be anywhere in the Tablix and for the columns to be hidden in a specific order. You can also specify the background color of the area that will display the hidden columns when expanded.

Examples

To specific the background color for hidden column area in a Tablix or change the header row you set the ColHeaderRow or BackgroundColor properties:

```
{
 "Tablix1": {
  "ColHeaderRow": 1,
  "BackgroundColor": "#F2F2F2"
 }
}
```

To hide columns in a specific order, you specify the priority of the columns. Any column not specified will be assumed to lowest priority and will be removed from the far right to left. Column numbers start at 1.

```
{
 "Tablix1": {
  "Columns": [
   {
    "Col": 1,
    "Pri": 1
   },
   {
    "Col": 2,
    "Pri": 1
   },
   {
    "Col": 4,
    "Pri": 2
   },
   {
    "Col": 6,
    "Pri": 3
```

```
    }
   ]
  }
}
```

To specify column headers anywhere in the report and have them change based on the grouping you need to specify the names of the header cells in the ColumnHeaders array property. The headers for the column will change for every instance of the cells in the ColumnHeaders array. This allows you to have multiple levels of headers and to have dynamic header values. Any Column that does not have a header defined will default to the ColumnHeaderRow value.

```
{
 "Tablix1": {
  "ColumnHeaders": [
   "GroupColHeader1",
   "GroupColHeader2",
   "GroupColHeader3",
   "DetailColHeader1",
   "DetailColHeader2",
   "DetailColHeader3"
  ]
 }
}
```

JavaScript Action Extension

You can add an array of JavaScript actions to a Textbox, Tablix, Rectangle or image element. This action will be added but not replace any RDL action. It is best not to have an RDL action on the same element as a JavaScript Action. The script will have access to a parameter ***e*** which is the event parameter for the event. JQuery is also loaded. JavaScript actions can be embedded in each element or shared across elements. The data property of ***e*** with have the following properties:

- reportViewer: a pointer to the reportViewer HTML element and widget
- element: the html element of the textbox or image
- getImputs: a function that will return an array of objects for all the input elements in the same row
- easySubmit: a function that will perform a submit similar to a form submit

The getImputs and easySubmit functions are useful if you want to handle a button click to submit input elements in the same row of a tablix.

```
getImputs: function(element,filter)
element: the html element in a tablix row that you want to get the other html imput elements on
filter: either “auto” (default) or “all”.  All returns all inputs, auto uses the InputSubmit property to determine which fields to return.
returns: an array of data objects {name, value,origionalValue,type,submitType} where type is the type of input element, and submitType is the InputSubmit property specified for the field

easySubmit: function(inputs,type,url,datatype, done,fail)
inputs: an array of dataobjects to post, returned from getImputs
```

type: GET or POST

url: the url to post to

datatype: text or json

done: the function to call when done

fail: the function to call on fail

Examples

To create a function that navigates to the next page of the report, set the JavaScriptAction property of the NextPage RDL element.

```
{
 "NextPage": {
  "JavaScriptActions": [
   {
    "On": "click",
    "Code": "var page = e.data.reportViewer.reportViewer('getCurPage');\n
e.data.reportViewer.reportViewer('navToPage',page+1);"
   }
  ]
 }
}
```

To highlight a cell when you mouse over it using a shared JavaScript action:

```
 {"HighlightTextbox": {
  "JavaScriptActions": [
   {
    "On": "click",
    "Code": "$('#bing').attr('src','http://www.bing.com/search?q='+$(e.data.element).text());"
   },
   {
"SharedAction": "ChangeColor"     }
  ]}
 ,
 "SharedActions": {
  "ChangeColor": {
   "JavaScriptActions": [
    {
     "On": "mouseenter",
     "Code": "$(e.data.element).css('background-color','red');"
    },
    {
     "On": "mouseleave",
     "Code": "$(e.data.element).css('background-color','white');"       }
```

```
    ]
   }
  }
}
```