

Editing Data with Microsoft SQL Server Reporting Services

By Boris Cherkasskiy
09/24/09

Overview

[Microsoft SQL Server Reporting Services](#) (SSRS) is a tool that is used to develop Web-based reports; it integrates into MS Internet Information Server (IIS) and uses familiar MS Visual Studio as the report designer. A simple report in SSRS could be generated in minutes. SSRS can create advanced reports that allow the viewer to drill-down into deeper data such as sub-reports and graphs. It can be integrated into [MS SharePoint Portal](#). SSRS is an integral and versatile part of MS SQL server included in all editions of MS SQL server, including the free "Express" edition.

In some situations the end user may desire the ability to update and edit a report separate from the initial data entry and collection. For example one of DMC's clients wished to add additional data to comment fields and add missing that operators initially failed to input. The obvious, straight-forward solution would be to create a custom asp.net web page to replicate the report with editing capabilities. This is probably good solution for simple report, but re-creating a complex, deep report using asp.net could be time-consuming. The alternative solution DMC decided upon was to create a custom popup window (asp.net web page) with the few values needed to be edited and call this popup from SSRS report.

There are several approaches to create popup window:

- Using hidden div element with editor popup and show it on the screen when needed. This approach required adding custom code to the report or wrapping the report into another page.
- Using the JavaScript function `window.showModalDialog`. This function is Internet Explorer specific and isn't supported by other browsers. Also this function does not allow access to the parent window from the popup.
- Using the JavaScript function `window.open`

DMC decided upon the JavaScript function `window.open` option because it is universal, supported by many browsers and allows access to the parent window from the popup (this allows to most of the JavaScript to reside in the popup window). Using a hidden div on the page is also good solution, but this approach would require adding JavaScript functions to the report.

Some additional considerations need to be reviewed before implementing the popup editor solution:

- This solution is good if only a few items/values need to be edited in a complex report; If multiple values are to be edited in a simple report, the time spent to create custom popup could be comparable with the time needed to recreate the whole report in asp.net.
- The popup needs to be an asp.net web page and has to be deployed on the same server/domain with SSRS
- The value to modify has to be stored in the database and the popup page needs access to this database
- The value to be modified must be identifiable; a unique identifier must be passed (the PK for the specific row in the table for example) to the popup in order to find the correct row record/values

Following is an overview of the sequence of events that occur when a user uses the report editing popup:

1. User clicks on the report control element to be edited and the popup web page opens in a new window using URL link action and JavaScript
2. Row to edit identifier (row PK or similar) is passed to the popup in a query string
3. Popup opens in the middle of the screen
4. Popup opens modally (disables report page to prevent user to navigate out of the page, open another popup, etc.)
5. Popup creates/modifies data in the database

6. Popup refreshes report, after the user finishes editing and data is submitted to database
7. Popup re-enables report to allow user interaction with it

Below are step by step instructions and code snippets to achieve this functionality.

Call editor popup from SSRS and pass table PK

To open editor web page in new window use the Action properties of the SSRS control.

In "jump to URL" JavaScript is needed to open the page in a new window. The PK value is added as a query string parameter to identify the row that needs to be edited; of course the PK value is optional and depends on your application.

```
= "javascript:void window.open('http://myserver.org/ReportEditor/UserEditPopup.aspx?PK=" & Fields!PK.Value & "', '_blank', 'status=0, location=0, menubar=0, toolbar=0, directories=0, resizable=0, scrollbars=0, width=100, height=100');"
```

The JavaScript function `window.showModalDialog` could be used instead of `window.open`, but `showModalDialog` will work in IE only and doesn't allow access to the parent page from the child page (popup window in our case). Width and height parameters are not important as the popup window size is configured from the popup itself.

Resize and center popup, disable report from the popup

Because we don't set the popup size from the report, we use the JavaScript function to set the popup window size and position.

```
function WindowSetSize() {  
    // Seems like DocumentElement has proper size in IE  
    var docElem = document.documentElement;  
  
    // Scroll width/height is size of the whole document;  
    // Client width/height is browser window size  
    // Scroll width should be set by body style width;  
    // Scroll height calculated automatically by page size  
    var iWidth = docElem.scrollWidth - docElem.clientWidth;  
    var iHeight = docElem.scrollHeight - docElem.clientHeight;  
  
    // Resize window to be exact size of the document  
    window.resizeBy(iWidth, iHeight);  
  
    // Move window to the center of the screen  
    window.moveTo((screen.width-docElem.clientWidth) / 2, (screen.height-docElem.clientHeight) / 2);  
}
```

The following JavaScript function is used in the popup window to disable (gray-out) the report. The function creates a grey div element in the parent window with 50% transparency and places this div on the other elements. Please note that in order to access parent window from the popup both of them should be on the same domain.

```
function ParentWindowGrayOut(){
  // Be sure parent window exists
  if (window.opener) {
    // Parent window exists; try to get gray div element from parent window
    var divElem = window.opener.document.getElementById('GrayDiv');

    // Verify gray element is exists
    if (!divElem) {
      // Gray div does not exists, create it
      divElem = window.opener.document.createElement('div');
      divElem.id = 'GrayDiv'; // Set element name

      // Get body element and add new div to it
      var bodyElem = window.opener.document.getElementsByTagName("body")[0];
      bodyElem.appendChild(divElem);
    }

    // Apply style to this new div
    divElem.style.position = 'absolute';
    divElem.style.top = '0px';
    divElem.style.left = '0px';
    // Clip the content of the original document
    // Shouldn't show anything outside the new div
    divElem.style.overflow = 'hidden';
    // Set filter to be 50% transparent
    divElem.style.filter = 'alpha(opacity=50)';
    // This should be high enough to be on the top of every other elements
    divElem.style.zIndex = 100;
    divElem.style.backgroundColor = '#000000';

    // Get parent window size
    var docElem = window.opener.document.documentElement;
    divElem.style.width = docElem.scrollWidth;
    divElem.style.height = docElem.scrollHeight;
    divElem.style.display = 'block';
  }
}
```

We have to call both JavaScript functions on the page load event.

Modify data in the database

The popup window is just a standard aspx (or php if you prefer) page, so you have full asp.net toolbox for your disposal to enter/validate/modify data in database.

Close popup window, refresh report data and re-enable report

When the user is done editing data and all data is committed to the database the popup window needs to be closed from code behind. To do so use the RegisterStartupScript method, (below is an example in VB).

```
ClientScript.RegisterStartupScript(Me.GetType(), "CloseScript01", "window.close();", True)
```

The report needs to be enabled in the parent window in any case, even if the popup is closed by the close window ("X") button. JavaScript exposes the "window.onbeforeunload" event exactly for this purpose. When this event fires the parent window must be re-enabled using the JavaScript function "ParentWindowUnGray". This function removes the div element created by the "ParentWindowGrayOut" function. It does not actually delete div element, but sets its style to "none".

```
function ParentWindowUnGray() {  
    // Be sure parent window exists  
    if (window.opener) {  
        // Parent window exists; try to get gray div element from parent window  
        var divElem = window.opener.document.getElementById('GrayDiv');  
  
        // Verify gray element is exists; If it doesn't, we have nothing to ungray  
        if (divElem) {  
            // Gary div was created before, don't display it  
            divElem.style.display = 'none';  
        }  
    }  
}
```

The report must be refreshed. I found at least three methods for doing this: two for pure Reporting Services and another one for a report embedded into [DotNetNuke](#) using the [SQLRS](#) module. We should try all methods.

```
if (window.opener) {  
    // Seems like there are two methods to refresh report, try both  
    if (window.opener.window.ActionHandlerReportViewerControl != null) {  
        window.opener.window.ActionHandlerReportViewerControl('Refresh', null);  
    } else if (window.opener.document.getElementById('ReportViewerControl') != null) {  
        window.opener.document.getElementById('ReportViewerControl').ClientController.ActionHandler('Refresh', null)  
    } else {  
        // Seems like DNN Report Viewer 2005 creates div with id "x_ViewRS2005_ReportViewer_x"  
        // If such div exists we have to call "ClientController.ActionHandler('Refresh', null)"  
        var AllDiv = window.opener.document.getElementsByTagName('div')  
        for (var i = 0; i < AllDiv.length; i++) {  
            if (AllDiv[i].id.indexOf("ViewRS2005_ReportViewer") >= 0) {  
                if (AllDiv[i].ClientController) {  
                    AllDiv[i].ClientController.ActionHandler('Refresh', null);  
                }  
            }  
        }  
    }  
}
```

Other considerations

The above code is enough to have basic popup functionality. It could be extended to have better visual effects such as popup window animation, prevention from the popup loosing focus, etc. In order to prevent the popup window from loosing focus, the `window.onblur` event should be handled to set focus back to the popup. `window.onbeforeunload` event fires in case of the post-back, which causes the report to refresh/flicker. In order to handle this situation properly we set a JavaScript global variable in the onclick event of any controls causing a post back and evaluate this variable before enabling/refreshing the report.