GemBox Support Center

ThreadAbortException in ASP.NET application

Mario - GemBox - 2017-02-09 - 0 Comments - in GemBox.Spreadsheet

When using GemBox.Spreadsheet in ASP.NET application, a common task that one can require is downloading of spreadsheet files. To do this, GemBox.Spreadsheet provides overload Save methods that can be used for direct streaming of spreadsheet files to a client's browser.

You can find examples of this in the Save method documentation or in the article about working with spreadsheet file streams.

Now consider the following scenario:

C# code

```
public partial class _Default : System.Web.UI.Page
{
    protected void btnDowload_Click(object sender, EventArgs e)
    {
        ExcelFile workbook = new ExcelFile();
        ExcelWorksheet worksheet = workbook.Worksheets.Add("Sheet1");

        worksheet.Cells[0, 0].Value = "Hello World!";

        try
        {
            workbook.Save(this.Response, "Output.xlsx");
        }
        catch (Exception ex)
        {
            string typeName = ex.GetType().Name;
        }
    }
}
```

VB.NET code

```
Public Class _Default
    Inherits Page

    Protected Sub btnDowload_Click(sender As Object, e As EventArgs) Handles btnDowload.
Click
        Dim workbook As New ExcelFile()
        Dim worksheet As ExcelWorksheet = workbook.Worksheets.Add("Sheet1")

        worksheet.Cells(0, 0).Value = "Hello World!"

        Try
            workbook.Save(Me.Response, "Output.xlsx")
        Catch ex As Exception
            Dim typeName As String = ex.GetType().Name
        End Try
```

```
    End Sub
End Class
```

If we place a breakpoint inside a catch block and run this "btnDowload_Click" method, we'll notice that the ThreadAbortException is thrown at execution, even though we have successfully downloaded an "Output.xlsx" file.

The reason for this is that those Save overload methods that are used to stream a file to a client's browser will call a Response.End at the end of their execution. The following code is somewhat similar to what that Save method does:

C# code

```
var response = this.Response;
var fileName = "Output.xlsx";
var options = SaveOptions.XlsxDefault;

response.Clear();
response.ContentType = options.ContentType;
response.AddHeader("Content-Disposition", "attachment; filename=" + fileName);

// Unlike with other formats, with XLSX we cannot write directly to Response.OutputStream.
// This is because MS Packaging API does not allow this, so we use a temporary MemoryStre
am.
var ms = new System.IO.MemoryStream();
workbook.Save(ms, options);
ms.WriteTo(response.OutputStream);

response.End();
```

VB.NET code

```
Dim response = Me.Response
Dim fileName = "Output.xlsx"
Dim options = SaveOptions.XlsxDefault

response.Clear()
response.ContentType = options.ContentType
response.AddHeader("Content-Disposition", "attachment; filename=" + fileName)

' Unlike with other formats, with XLSX we cannot write directly to Response.OutputStream.
' This is because MS Packaging API does not allow this, so we use a temporary MemoryStrea
m.
Dim ms = New System.IO.MemoryStream()
workbook.Save(ms, options)
ms.WriteTo(response.OutputStream)

response.End()
```

The last executed method is a Response.End that is used to immediately terminate the response after a file download. **By nature, it throws a ThreadAbortException**; you can read about this in the Resonse.End method's remarks.

So what we can do, is either ignore the catching of a ThreadAbortException type of exception when calling a GemBox.Spreadsheet's Save method, or use an alternative approach to stream the file. For example:

C# code

```
var response = this.Response;
var fileName = "Output.xlsx";
var options = SaveOptions.XlsxDefault;

response.Clear();
response.ContentType = options.ContentType;
response.AddHeader("Content-Disposition", "attachment; filename=" + fileName);

var ms = new System.IO.MemoryStream();
workbook.Save(ms, options);
ms.WriteTo(response.OutputStream);

response.Flush();
response.SuppressContent = true;
HttpContext.Current.ApplicationInstance.CompleteRequest();
```

VB.NET code

```
Dim response = Me.Response
Dim fileName = "Output.xlsx"
Dim options = SaveOptions.XlsxDefault

response.Clear()
response.ContentType = options.ContentType
response.AddHeader("Content-Disposition", "attachment; filename=" + fileName)

Dim ms = New System.IO.MemoryStream()
workbook.Save(ms, options)
ms.WriteTo(response.OutputStream)

response.Flush()
response.SuppressContent = True
HttpContext.Current.ApplicationInstance.CompleteRequest()
```

However, note that this alternative approach does not work in ASP.NET applications that have a Medium Level Trust, which is why GemBox.Spreadsheet does not use it.