NetAdvantage Reporting 2013.1 (build 1058) Service Release Notes – November 2013

Use Reporting, the industry's first WPF and Silverlight-based design-time and rendering reporting tool, to create elegant and easy-to-design reports engineered to help you deliver information to your users in the shortest time possible —without the need for legacy code.

Bug Fixes

Component	Product Impact	Description
Reporting (Viewer)	Bug Fix	**Server-side Export didn't work when System fonts were not accessible** When exporting a report from the server, now you can define IFontLoader's to load fonts when the system fonts are not available. When exporting reports with an IServerExporter, you can now configure fonts as embedded fonts. For more information refer to *Font Loaders* in the Changes/Addition section.
Reporting (Viewer)	Bug Fix	**Using Reporting and Infragistics Silverlight controls in the same application issue fixed** You can now use the Reporting SL viewer and other Infragistics Silverlight controls in the same application without issue.
Reporting (Viewer)	Bug Fix	**Print button in the HTML5 viewer was not displayed in IE11** When viewing a report in HTML5 using IE11 with the Adobe Reader extension installed, now the print button is displayed as expected.

Changes/Additions for 2013 Volume 1

Font Loader

If you are exporting a report server-side using a server without access to system fonts or without the required fonts installed, you need to dynamically load those fonts into the server.

The way to do this is implementing the IFontLoader interface.

IFontLoader interface

The interface is defined under Infragistics.Reports namespace in the InfragisticsWPF4.Reports.v13.1 assembly and has only one method defined with the following signature:

```
bool TryGetExternalEmbeddedFonts(string fontFamily, ref
FontWeight fontWeight, ref FontStyle fontStyle, out byte[]
data);
```

The method implementation should return *true* or *false* depending if the requested font could be resolved or not, and requested font bytes should be returned in the *data* parameter of the method.

Here there is a practical example:

In this solution we have the *Arial.tff* file as an embedded resource in the web project where the FontLoader is implemented.

```
[FontLoader]
public class MyFontLoader : IFontLoader
{
   public bool TryGetExternalEmbeddedFonts(string fontFamily, ref
FontWeight fontWeight, ref FontStyle fontStyle, out byte[] data)
  {
    const string ResourceName = "Fonts.Arial.ttf";

    if (fontFamily == "Arial")
    {
        var assembly = Assembly.GetExecutingAssembly();
        using (var stream =
assembly.GetManifestResourceStream(ResourceName))
        {
            var bytes = GetBytes(stream);
            data = bytes;
        }
```

```
                return true;
            }

            data = new byte[0];
            return false;
        }

        private static byte[] GetBytes(Stream stream)
        {
            using (var memoryStream = new MemoryStream())
            {
                stream.CopyTo(memoryStream);
                return memoryStream.ToArray();
            }
        }
    }
```

In order to enable this class to be discovered by our MEF composer you should follow this two steps:

1) Configure the assembly where the loader is contained as a runtime assembly. A section in the configuration file with this tags should be added:

```
<infragistics.reports>
     <runtimeAssemblies>
         <add assembly="ProjectAssemblyName" />
    </runtimeAssemblies>
 </infragistics.reports>
```

2) Add a *[FontLoader]* attribute at the top of the class. This attribute indicates to our MEF composer that this is an implementation of an IFontLoader.

   You need to add a reference to the System.ComponentModel.Composition assembly.