ALLAN THRAEN | 14 years ago | PDF |

ARTICLE RATINGS WITH DYNAMIC DATA STORE

A classic need I've come across a bunch of times is people wanting to store various counters like user ratings or view counters in EPiServer CMS. Up till CMS 5 they've been left with a few choices:

- Install EPiServer Community it has both features. Not a bad idea, but kinda overkill if all you need is a simply 5-star rating mechanism.
- Have a property on a page to keep count. I wouldn't really consider this option at all, due to performance implications in all the frequent publishing – but I have seen people attempt this solution
- Use a custom database – or database table to keep track of the data
- Use the old 'secret' data store – or pretend that the data is an xform – even though it really isn't.

I even did a blog post some time ago where I made a sample implementation of a view counter wrapped in a disguised custom table.

Well. In EPiServer CMS 6 Dynamic Data Store comes to the rescue. And it's really awesome…

So, naturally, as soon as I could get my hand on a CTP with the Dynamic Data Store I jumped right into building a simple rating mechanism.

I implemented it as a couple of extension methods, that can extend either a PageData object – or just the PageBase, making them easy to use on an existing asp.net form or user control.

I created a class to hold my data. Essentially my data will be 2 integers – A sum of all ratings and a count of how many has rated.

```
public class Rating : IDynamicData
{
    public int Sum { get; set; }
    public int Count { get; set; }

    #region IDynamicData Members

    public EPiServer.Data.Identity Id     {get; set;}

    #endregion
}
```

The data store I am going to use will be really basic – just a list of the Rating objects as described above. To really speed it up, I decided to "cheat" a little and ensure that the Identity of each rating item corresponds to the Guid of the page it's on. That way I don't have to search through unindexed data to every time I need to list or rate a page, but rather just load it from a Guid Id. To show a rate I create the data store (with a setting to make sure we use an existing store instead of creating a new, if one does exist).

```
public static int Rate(this PageData pd)
{
    DynamicDataStore<Rating> ds = DynamicDataStore<Rating>.CreateStore(false);
    Rating rt = ds.Load(Identity.NewIdentity(pd.PageGuid));
    if (rt == null) return 0;
    return rt.Sum / rt.Count;

}
```

For storing a rate, we check if it's the first time the page is rated – and in that case create a new Rating object.

```
public static int Rate(this PageData pd, int score)
{
    DynamicDataStore<Rating> ds = DynamicDataStore<Rating>.CreateStore(false);
    Rating rt = ds.Load(Identity.NewIdentity(pd.PageGuid));
    if (rt == null)
    {
        rt = new Rating();
        rt.Id = Identity.NewIdentity(pd.PageGuid);
        rt.Sum = score;
        rt.Count = 1;
        ds.Save(rt);
    }
```

And we simply save the object back to the DDS.

On top of that I simply add a couple of overloads:

```
public static int Rate(this TemplatePage p)
{
    return Rate(p.CurrentPage);
}

public static int Rate(this TemplatePage p, int score)
{
    return Rate(p.CurrentPage, score);
}
```

And we are ready to rumble :-)

To test it, I made a user control that I put on various page types, which is so simple in code, yet so ugly in design that I'm not even going to share the code for it. But it worked! And total code-time = 15 min.

RECENT POSTS

Phone: +45 26 13 66 96
CVR: 39680688