Enhanced String Handling II

Introduction

In my previous article **Enhanced String Handling**, http://www.codeproject.com/KB/string/EnhancedStringHandling.aspx, I ended the article with a section: **Where do we go from here**, in where I pointed out a limitation: "The specifications as they stand now will not allow for a delimiter as part of the value. So for example: **{identifier::value containing an open or close brace}** will not pass the brace matching check. This is not of theoretical interest only. For example if we would like to write a **{Decrypt::encrypted value}** decryption construct—if we cannot guarantee that the encrypted value has neither an open nor close braces then we cannot write such a **ProcessDecrypt** class. This problem is not necessarily confined to the case of a single charactered delimiter because the multi charactered delimiters are transformed into single charactered delimiters."

A few days after I published the article it occurred to me that the limitation was in my mind only.

The perceived problem

When we attempt to write a regular expression that will match on the form: ***{Crypt::encrypted-text}*** we are in trouble if the encrypted-text contains an open delimiter ("***{***"), a close delimiter ("***}***") or a separator string ("***::***"). Therefore, I concluded, prematurely, that we needed more powerful machinery to handle such situations.

Fixing the limitation

Up to now, throughout the article: **Enhanced String Handling**, we handled problems of this kind by substituting the offending string with a new string. In this case we can use this very same technique by providing the encrypted text as unicode numbers converted to strings.

So if we are to encrypt the string: "**01234 -- The quick brown fox jumped over the lazy dogs -- 56789!**" we will first encrypt it into a string or characters. But instead of including the encrypted characters in a string as is (in the ***{Crypt::encrypted string}*** construct) we will include the unicode character representation of the encrypted string, coma separated. So the string: "**01234 -- The quick brown fox jumped over the lazy dogs -- 56789!**" encrypted will be represented as: "**225,14,132,43,189,68,220,227,84,28,69,216,140,97,101,85,254,11,229,238,148,191,73,177,235,233,193,176,45,187,218,44,92,107,175,168,56,90,14,24,201,219,251,161,82,146,221,133,249,49,111,196,239,55,164,209,93,126,144,158,212,39,101,29,197,221,62,174,210,137,124,134**"

The **ProcessCrypt()**, therefore, will process a construct representing the above string, encrypted, like so: ***{Crypt::225,14,132,..}***. Therefore, we solve the issue of an encrypted string that can contain any character including a delimiter or a separator. See CypherTest(), TestMethod. (The sample code in ProcessCrypt() is not meant to be the most secure encryption/decryption code, it is meant to illustrate the point—that we can overcome a delimiter or a separator in the string value to be processed.)

```
    [TestMethod]
    public void CypherTest()
    {
        string org = "01234 -- The quick brown fox jumped over the lazy dogs -- 56789!";
        string encripted = string.Format("Text: {{Crypt::{0}}}", ProcessCrypt.Encrypt(org));

        var context = new List<IProcessEvaluate>();
        context.Add(new ProcessCrypt());
        var eval = new EnhancedStringEval(context);
        string decript = eval.EvaluateString(encripted);

        Assert.AreEqual("Text: " + org, decript);
    }
```

While ProcessCrypt is as follows:

```
  public sealed class ProcessCrypt : IProcessEvaluate
  {
    static ProcessCrypt()
    {
        var reo = RegexOptions.Singleline | RegexOptions.IgnoreCase;
        _reCrypt = new Regex(@"{\s*Crypt\s*::(?<cipher>[0-9,]*)}", reo);
    }

    public ProcessCrypt() { }

    #region IProcessEvaluate Members

    /// <summary>
    ///
    /// </summary>
    /// <param name="src"></param>
    /// <param name="ea"></param>
    public void Evaluate(object src, EnhancedStringEventArgs ea)
    {
        // Task 1:
        string encrypted = ea.EhancedPairElem.Value;

        // ea.IsHandled == false, by default.
        Match m = _reCrypt.Match(encrypted);
        if (!m.Success) return;

        string deciphered = _reCrypt.Replace(encrypted, CipherReplace);
        if (deciphered == encrypted) return;

        // Task 2
        ea.IsHandled = true;

        // Task 3
        ea.EhancedPairElem.Value = deciphered;
    }

    #endregion

    /// <summary>
    ///
    /// </summary>
    /// <param name="m"></param>
    /// <returns></returns>
    private string CipherReplace(Match m)
    {
        string encrypted = m.Groups["cipher"].Value;
        return Decrypt(encrypted);
    }

    private const string _criptSplitter = ",";
    private static Regex _reCrypt;
    ...
  }
```

Enjoy,

Avi