

1 The first rich text editor (RTE) for Silverlight 2

Many people are waiting for a way to type rich text. Even if I think that Microsoft will bring out their own one, nobody knows how long to wait for and which features are included. My RTE ships with various common features and an extensive documentation. Please note that the whole thing is still in BETA state and there are some weird bugs on XP and different ones on Vista. It seems as if they are not caused by my component at all, because Visual Studio would notify me. So wait for the final version of Silverlight 2, until you use this component in any production environment! It is of course possible that some things are still undocumented or documented ones will not behave as expected. Don't hesitate to report such incoherencies.

An incomplete feature list:

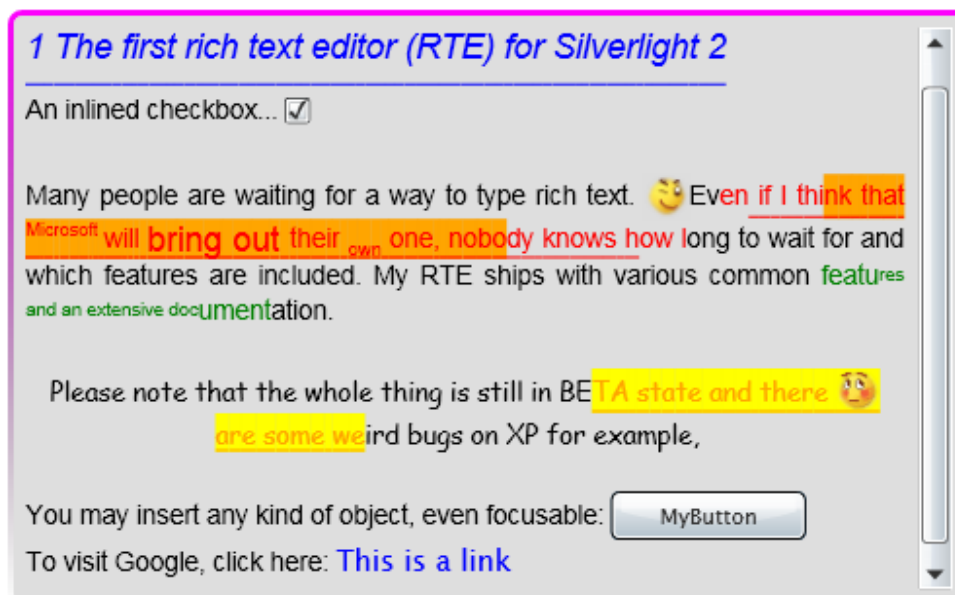
- Copy/Paste formatted text between RichTextBoxes and copy/paste from/to clipboard of unformatted but macro-enabled text. This means in windows clipboard even things like emoticons will be kept.
- You may insert line breaks, unordered lists and blockquotes.
- You may use various keyboard selection features like "End/Home/PageUp/PageDown/Left/Up/Right/Down", "Ctrl"+"A/End/Home", "Ctrl+Shift"+"End/Home/Left/Right", "Shift"+"End/Home/PageUp/PageDown/Left/Up/Right/Down" and so forth...
- Supports direct Unicode character input using "Ctrl"+[NumPad].
- All silverlight font formatting is supported and even some more like SUP/SUB formatting.
- You may define macros and a proper object class that should replace matching text, like emoticons...
- In contrast to many other rich text editors, this one is fully real-time. That means no preview is required because the editor allows editing all things directly.
- If you only use macros and IRichTextObject to extend the control, you will automatically get support for secure content serialization of all control elements. Content serialization also supports to reload content and edit it again.
- Secure content serialization gets rid of any potential security leak when storing user typed formatted text on a server and presenting it to visitors, because it is fully verifiable.
- You may restrict font formatting to a well defined custom subset. This allows you to ensure that all user typed input matches your needs or website design. (this feature is currently not implemented, but only prototyped)
- Snapshots allow convenient access to formatted content and also Find&Replace with regular expressions for example...

Because there is a complete API documentation available at "RichText\Docu.chm", this introduction concentrates on introducing the component. If you really want to understand it, you will have to read the documentation and work with the demo at "RichText\Demo\Page.xaml.cs".

Visit <http://www.codeplex.com/richtextedit> for source code and latest bug fixes. If you want to report any bug or improvements, please write me a mail to LooneyLynn@gmx.biz. I will then add the issue to the CodePlex issue tracker. You should also check if your bug isn't already there...

1.1 A first look

The following screenshot shows some formatting and the use of macros (as emoticons are). It also uses a custom designed border and you can see that the inner content automatically gets realigned to fit into the outer border. You may also customize selection back- and foreground as well as cursor color.



There obviously are two uncommon features. The first thing is that you may insert any object derived from FrameworkElement into the text flow. The second thing is that such objects, as macros, are capable of assigning the formatting information, which will allow emoticons, for example, to also be underlined and background colored as the surrounding text is.

As you can see, there is a little problem with Silverlight's interpolation feature. It leads to small separators which are visible after each char when font background is unequal to control background. For obscurity, this seems to be an issue of Vista, because under XP it is working as expected!

If someone has an idea how to work around this issue, please tell me!

1.2 Clipboard support

But it goes further. If you select such custom objects, for example the button, you will be able to copy/paste it with Ctrl+c/v or internal clipboard methods including all text formatting. If you select and copy macros to clipboard, they will be translated back to their text representation allowing to

copy/paste macros between application boundaries, where in contract formatted text operations are restricted to the same application only. Clipboard operations are also supported, if the browser does not support it or the user has denied access. But then of course all operations are restricted to the current application.

This way you are able to thread custom objects as letters and also remove/overwrite them as usual text with backspace for example.

1.3 Future features

Of course this control is not perfect. The following features will be implemented within the next months:

- A XML scripting language to initialize rich text content in Expression Blend for example. This also eases management of multiple languages, because content and design are separated.
- A way to wrap text around controls like known from Microsoft Office. Currently they are only inserted like a letter which limits capabilities of inserting images or any other kind of rich non-text content. But this feature is really far away from being simple, so don't wait for it.
- Non-editable XAML output which requires the XamlWriter that is currently not available.
- Features that other developers may submit.

2 A tour through the demo

To understand how to use this component I really recommend stepping through the demo file. It utilizes nearly all features and shows some tricks to realize hyperlinks or Find&Replace with regular expressions using the abstract RTE interface. The following chapters refer to "RichText\Demo\Page.xaml.cs".

2.1 Initialization

As RichTextEdit is derived from UserControl, you may use and initialize it as any other framework element.

```
RichEdit = new RichTextEdit();
RichEdit.AutoFocus = true;
RichEdit.InsertString(...);
RichEdit.OnSelectionChanged += new NotificationHandler(RichEdit_OnSelectionChanged);
RichEdit.OnContentChanged += new NotificationHandler(RichEdit_OnContentChanged);
```

```
MSNEmoticons.Apply(RichEdit);  
RichEdit.RegisterObject(new SerializableButton(null));
```

The above code initializes the content with a test string and makes the control ready to receive two special events. This allows your code to get notified if either the selection or content has changed. Also the MSNEmoticons-Extension is applied which will replace common emoticons like “:-)” with a matching MSN-Icon. The last line is something special and will be covered later. All lines except the object creation are optional!

2.2 Updating a ToolBar

```
void RichEdit_OnSelectionChanged(object sender)  
{  
    IsUpdating = true;  
  
    ...  
}
```

This event will update the whole Demo GUI according to the formatting at current selection. This method is very important for any useful ToolBar-Editor pair and you should try to understand what is going on there... The most important thing to mention is that all formatting properties like “RichTextEdit.FontAttributes” are set according to the current selection. That means you just have to read them out within the “OnSelectionChanged”-Handler and update your ToolBar properly.

2.3 Find&Replace

To find text, we need to utilize the RichTextEdit.Snapshot. This allows us to directly operate on string content what is not common when dealing with rich text:

```
private MatchCollection REGEX_Matches;  
private Int32 REGEX_Index = 0;  
private RichTextEdit.Snapshot REGEX_Snapshot;  
  
private void BTN_Find_Click(object sender, RoutedEventArgs e)  
{  
    REGEX_Snapshot = RichEdit.QueryText();  
  
    Regex Exp = new Regex(EDIT_Find.Text, RegexOptions.IgnoreCase |
```

```
        RegexOptions.Multiline | RegexOptions.ECMAScript);

    REGEX_Matches = Exp.Matches(REGEX_Snapshot.Text);
    REGEX_Index = 0;

    BTN_Replace.IsEnabled = true;
    BTN_FindNext.IsEnabled = true;
    BTN_FindNext_Click(null, null);
}
```

To realize the “Find Next” method, we just loop through all matches...

```
private void BTN_FindNext_Click(object sender, RoutedEventArgs e)
{
    if ((REGEX_Matches == null) || (REGEX_Matches.Count == 0))
    {
        BTN_FindNext.IsEnabled = false;
        return;
    }

    if (REGEX_Matches.Count <= REGEX_Index)
        REGEX_Index = 0;

    // select match
    Match m = REGEX_Matches[REGEX_Index++];

    REGEX_Snapshot.Select(CursorPosition.End, m.Index, m.Length);
}
```

As you can see a snapshot also allows us to select rich text based on string offsets.

If you also want to replace rich text, thing will get a little bit more complicated. Firstly we need to remove the text referred by the current match entry. Then we insert the replacement and select it.

```
private void BTN_Replace_Click(object sender, RoutedEventArgs e)
{
    if ((REGEX_Matches == null) || (REGEX_Matches.Count == 0))
        return;

    if (REGEX_Matches.Count <= REGEX_Index)
        REGEX_Index = 1;

    // replace selection
    Match m = REGEX_Matches[REGEX_Index - 1];
    Int32 iStart = m.Index;
    Int32 iLen = m.Length;

    REGEX_Snapshot.Remove(ref iStart, ref iLen);
    REGEX_Snapshot.Select(CursorPosition.Start, iStart, 0);
}
```

```
REGEX_Snapshot.InsertString(EDIT_Replace.Text);
REGEX_Snapshot.Select(CursorPosition.Start,
REGEX_Snapshot.SelectionStart - EDIT_Replace.Text.Length,
EDIT_Replace.Text.Length);
}
```

Even if this might look strange, it is a very consistent way to realize Find&Replace. Imagine there are custom objects between the letters that you wouldn't know about. Such circumstances are handled within a snapshot and you don't have to care about.

3 Secure content serialization

After a user has typed and formatted his text, you need to save it. My control provides a secure way to do this:

```
private void BTN_Serialize_Click(object sender, RoutedEventArgs e)
{
    MemoryStream Buffer = new MemoryStream();
    RichTextEdit.Snapshot Snapshot = RichEdit.QuerySelectionText();

    Snapshot.Serialize(false, Buffer);

    // convert to base64
    LABEL_Binary.Text = Convert.ToBase64String(Buffer.GetBuffer(), 0,
        (int)Buffer.Length);
    BTN_Deserialize.IsEnabled = true;
}
```

As you can see, again the snapshot is involved; you need a snapshot of what you want to serialize. If your web server does not support binary serialization, just encode it to base64 as shown above. All formatting, all macros and all custom rich text objects will be included in such a serialization stream.

Deserialization works similarly:

```
private void BTN_Deserialize_Click(object sender, RoutedEventArgs e)
{
    MemoryStream Buffer = new
        MemoryStream(Convert.FromBase64String(LABEL_Binary.Text));

    RichEdit.InsertDeserialization(false, Buffer);
}
```

3.1 Why is this secure?

It is secure because it is 100% verifiable. Secure does NOT mean that it is encrypted; you still have to use SSL for encrypted content transmission. Verifiability prevents you from a whole range of common attacks because it is simply not possible to store harmful serialization content on your server or invoke harmful operations when visiting your site.

In future versions, serialization will be improved to be compressed which will heavily reduce final stream size and though reduce your storage costs.

4 Custom rich text objects

Even if you may insert normal framework elements (FE), I don't recommend it. A normal FE is not included in any kind of serialization and though not in clipboard operations. To allow FEs to be serialized, you have to create a class which implements the "IRichTextObject"-Interface. The following shows a rich text object wrapper around a simple button:

```
class SerializableButton : UserControl, IRichTextObject
{
    private String m_Caption;
    private Button m_Instance;

    private SerializableButton() : base() { }

    public SerializableButton(String InCaption)
    {
        m_Caption = InCaption;
    }

    public Int16 GetTypeID()
    {
        return 0x100;
    }
    public Boolean IsFocusable
    {
        get
        {
            return true;
        }
    }

    public void Serialize(FrameworkElement InElement, BinaryWriter InTarget)
    {
        SerializableButton Button = (SerializableButton)InElement;
    }
}
```

```
InTarget.Write((String)Button.m_Instance.Content);
}

public FrameworkElement Deserialize(
    Boolean InIgnoreWarnings,
    BinaryReader InSource)
{
    return CreateInstance(InSource.ReadString());
}

public FrameworkElement CreateInstance()
{
    return CreateInstance(m_Caption);
}

private FrameworkElement CreateInstance(String InCaption)
{
    if (InCaption == null)
        throw new InvalidOperationException();

    SerializableButton Result = new SerializableButton();

    Result.m_Instance = new Button();
    Result.m_Instance.Content = InCaption;
    Result.Content = Result.m_Instance;
    Result.Width = 100;
    Result.Height = 25;

    return Result;
}
}
```

The above code combines a framework element and the interface. In general this is the easiest way, because during serialization and deserialization you will only get a reference to the FE and it's your duty to find the proper rich text object interface.

The class implementing the interface should have a constructor that takes all parameters that are required to instantiate it. In our case we only need a caption for the underlying button. Those are also the parameters that should be serialized, because this way you only have to deserialize and pass them to the constructor to deserialize the whole rich text object.

The instance creator is something special. The rich text object (RTO) can be threaded as a wrapper around a normal framework element. It provides a way to keep all required data through serialization, to later recreate a technically equal instance. An RTO instance should always provide technically equal framework elements through "CreateInstance" but NEVER return any object twice. It should just ensure that all returned objects referring to the same RTO constructor parameters will behave and look the same.

Before all that you have to register the RTO with your editor as shown in initialization:

```
RichEdit = new RichTextEdit();  
...  
RichEdit.RegisterObject(new SerializableButton(null));
```

You don't need to pass any parameters, because this RTO instance should just provide the "Serialize", "Deserialize" and "GetTypeID" methods which can be threaded as static, but static methods are not supported with interfaces...

This technology is something hard to explain but it is very powerful if once understood. Please look at the demo and try to find out how it works.