Hallo Bonn-to-code,

Während des Vortrags über **Windows Forms Application Settings** kamen zwei Fragen auf, die ich hier noch einmal kurz erläutern möchte:

- IPersistComponentSettings – sinnvoll/notwendig?

Bei dem gezeigten Beispiel über die persistente Speicherung von UserControl-Einstellungen wurde im Eventhandler `ColorControl_Disposed()` einzig die Methode `this.Settings.Save()` aufgerufen. Die Speicherung der Einstellungen in Abhängigkeit der Eigenschaft `SettingsKey` wurde von dem `LocalFileSettingsProvider` automatisch vorgenommen, da dieser durch die Implementierung des Interfaces `IPersistComponentSettings` davon ausgehen konnte, dass die Eigenschaft SettingsKey implementiert war. Selbiges gilt für die Eigenschaft `SaveSettings`.

Insofern spielt die Implementierung des Interfaces eine wichtige Rolle und sollte bei der Umsetzung eigener Settingsprovider entsprechend beachtet werden. (Man könnte versucht sein, eine solche Individualisierung über das Context-Objekt durchzuführen ;)

Übrigens: In dem Fall (`SettingsKey == string.Empty && SaveSettings`) werden die Einstellungen des UserControls zwar gespeichert, jedoch ohne die explizite Zuordnung zu dem entsprechenden Control.

- Serialisierung von komplexen Datentypen

Zudem kam die Frage auf, wie sich die Application Settings Infrastruktur mit komplexen Datentypen und deren Serialisierung verhält, hier weitere Infos:

********** snip **********

Quelle: Client Settings FAQ - http://blogs.msdn.com/rprabhu/articles/433979.aspx

A: There are two primary mechanisms that ApplicationSettingsBase uses to serialize settings:
(1) If a TypeConverter exists that can convert to and from string, we use it.
(2) If not, we fallback to the XmlSerializer. While most common types can be serialized in one of these ways, there are some types that may not. In such cases, you have a few different options:

- Implement a TypeConverter for the type that can convert to and from string. The implementation can use a suitable serialization mechanism like one of the formatters/serializers that ship in the Framework or any custom mechanism you wish. You can then specify this converter on the type itself or on the property in your settings class.
- Specify a particular SettingsSerializeAs enum value using a SettingsSerializeAsAttribute. For example, if you wish to serialize a setting in binary format, simply specify SettingsSerializeAs.Binary.

********** snip **********

Und noch ein Beispiel:
Eine einfache Klasse zur Speicherung von Dokument-Einstellungen (Ohne TypeConverter für die Konvertierung in ein `string`–Objekt):

```
public class DocumentSettingsProperty
{
    #region Fields

    private string _printer;
    private DocumentMarginsProperty _documentMargins;

    public string Printer
    {
        get { return _printer; }
        set { _printer = value; }
    }

    public DocumentSettings.DocumentMarginsProperty Margins
    {
        get { return _documentMargins; }
        set { _documentMargins = value; }
    }
...
public class DocumentMarginsProperty
{
   int _marginTop, _marginRight, _marginBottom, _marginLeft;
...
```

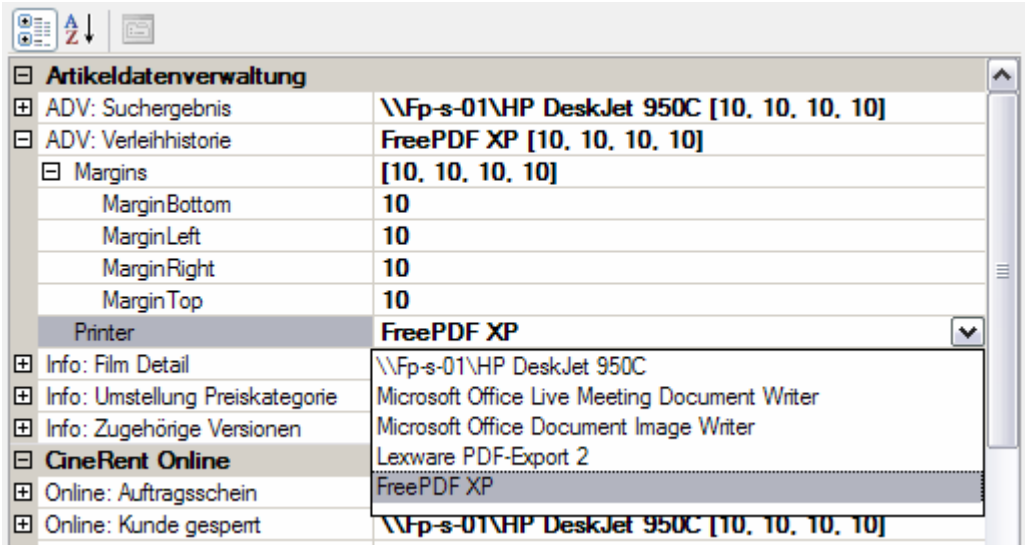
Und das dazugehörige Settings Property in der Settings-Klasse:

```
...
[UserScopedSetting]
[Category("Artikeldatenverwaltung")]
[Description("Artikel-Verleihhistorien werden auf dem hier konfigurierten Drucker
                                                                    ausgedruckt.")]
[DisplayName("ADV: Verleihhistorie")]
[TypeConverterAttribute(typeof(DocumentSettingsTypeConverter))]
public DocumentSettingsProperty AdvRentHistory
{
    get { return GetDocumentSettingsProperty("AdvRentHistory"); }
    set { this["AdvRentHistory"] = value; }
}
...
```

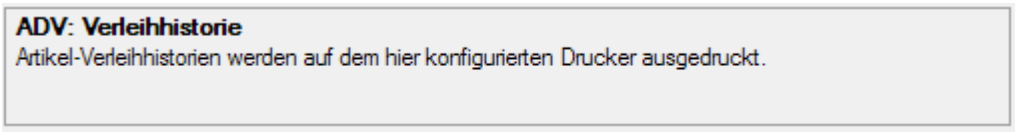
Ohne weiteres Zutun persistent gespeichert in der Datenbank als:

```
<?xml version="1.0" encoding="utf-16"?>
<DocumentSettingsProperty xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Printer>FreePDF XP</Printer>
  <Margins>
    <MarginTop>10</MarginTop>
    <MarginRight>10</MarginRight>
    <MarginBottom>10</MarginBottom>
    <MarginLeft>10</MarginLeft>
  </Margins>
</DocumentSettingsProperty>
```

Über das PropertyGrid dargestellt als:



...



Die individualisierte Anzeige wird über Verwendung der Klassen `ExpandableObjectConverter`, `StringConverter`, `TypeConverterAttribute`, `UITypeEditor` erreicht. Soweit ich weiß ist dies Inhalt einer der kommenden Vorträge, es bleibt also spannend...

Viele Grüße,
Moritz Pfennig