

Settings (Beckhoff.App.Settings.BASettings.dll)

Installation

In der Library „Beckhoff.App.Settings.BASettings.dll“ im Ordner „References“ befindet sich die Implementierung sowohl des Interfaces „IBASettings“ als auch die Oberfläche zum Ändern von Einträgen.

Um zur Implementierung von „IBASettings“ die hier beschriebenen Settings zu benutzen, muss eine entsprechende Konfigurationsdatei im Ordner „System\locConfig“ erstellt werden, die folgendermaßen aussieht: (Bsp. „Settings2.config.xml“).

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection,
      Microsoft.Practices.Unity.Configuration" />
  </configSections>
  <unity>
    <alias alias="string" type="System.String, mscorlib" />
    <alias alias="singleton" type="Microsoft.Practices.Unity.ContainerControlledLifetimeManager,
      Microsoft.Practices.Unity" />
    <alias alias="external" type="Microsoft.Practices.Unity.ExternallyControlledLifetimeManager,
      Microsoft.Practices.Unity" />
    <container>
      <!-- Beckhoff Application Settings -->
      <type type="Beckhoff.App.Core.Interfaces.IBASettings, Beckhoff.App.Core.Interfaces"
        mapTo="Beckhoff.App.Settings.BASettings, Beckhoff.App.Settings.BASettings">
        <lifetime type="singleton">
        </lifetime>
        <constructor>
          <param name="fileName" parameterType="string" value="System\GeneralSettings.xml"/>
          <param name="container"></param>
          <param name="eventAgg"></param>
          <param name="logger"></param>
        </constructor>
        </type>
      </container>
    </unity>
  </configuration>
```

Weiterhin sollten „alte“ Settings Konfigurationsdateien im selben Ordner gelöscht werden. (Settings.config.xml)

BECKHOFF

Der Aufruf des Userinterfaces kann mit Hilfe des Menumanagers auf eine beliebige Taste konfiguriert werden.

Dazu sollte das Form „Beckhoff.App.Settings.FormSettings2“ aus der DLL „References\Beckhoff.App.Settings.BASettings.dll“ eingebunden und die Instanz aufgerufen werden.

<input type="checkbox"/> FormTest	TcApplication.FormTest	Plugins\Beckhoff.App.Nc.dll
<input type="checkbox"/> FormPlcStatus	TcApplication.FormPlcStatus	Plugins\Beckhoff.App.Essential.dll
<input type="checkbox"/> FormSettings2	Beckhoff.App.Settings.FormSettings2	References\Beckhoff.App.Settings.BASettings.dll

Nach dem Aufruf des FormSettings2 werden die aktuellen Einstellungen in einer Baumstruktur dargestellt.

Die Auswahl eines Elements im Baum führt zur Anzeige der dazugehörigen Einträge im rechten Fenster.


The screenshot shows the Beckhoff FormSettings2 configuration interface. On the left, a tree view lists various settings categories, with 'General' currently selected. The main area on the right displays the configuration for the 'AppFrame' application. It includes several settings with their current values and data types:

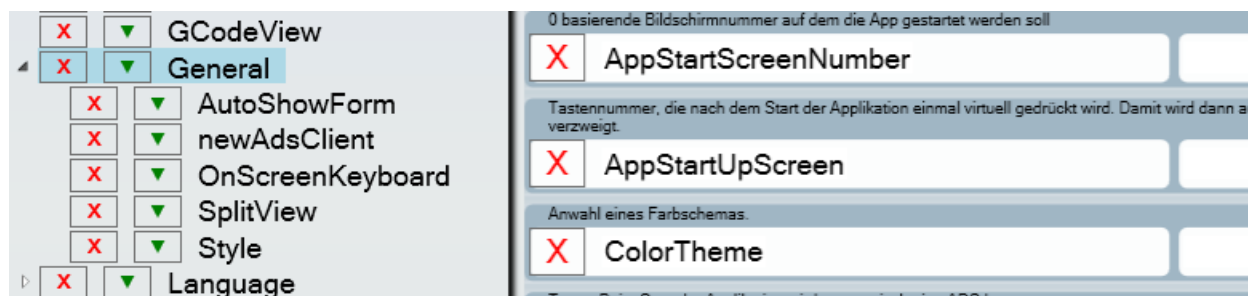
- AppFrame**: Boolean, checked (True = Applikationsrahmen mit Icons für das Minimieren, Maximieren und Schließen wird angezeigt).
- AppKey**: String, 0.
- AppStartMax**: Boolean, unchecked (True = Applikation wird maximiert gestartet).
- AppStartMin**: Boolean, unchecked (True = Applikation wird minimiert gestartet).
- AppStartScreenNumber**: Integer, 0 (0 basierende Bildschirmnummer auf dem die App gestartet werden soll).
- AppStartUpScreen**: Integer, 0 (Tastenummer, die nach dem Start der Applikation einmal virtuell gedrückt wird. Damit wird dann automatisch in das entsprechende Form verzweigt).
- ColorTheme**: Integer, 1 (Anwahl eines Farbschemas).
- CreateAdsServerInstance**: Boolean, checked (True = Beim Start der Applikation wird automatisch eine ADS Instanz erzeugt).
- EnableVisualStyles**: Boolean, checked (Der ab WinXp verfügbare Visuell Style kann hiermit aktiviert werden).
- Fkey2SelectedColor**: Color, LightBlue.
- Fkey2SelectedMode**: Boolean, unchecked.
- Fkey2StartupSelectedKey**: Integer, 1.
- FKeysBottomVisible**: Boolean, checked.


At the bottom, there is a toolbar with buttons for 'Manage', 'Import', and 'back', along with a status bar showing 'Administrator' and 'Level: Admin'.

Nach jeder Änderung eines Eintrags, werden die kompletten Settings automatisch gespeichert. Es wird dabei einmal alle 24 Stunden ein Backup der Settings erstellt, das an der Dateiendung „.backup“ zu erkennen ist.

Das XML Speicherformat ist abwärts kompatibel zu der „alten“ Settings Version, das heißt eine XML Datei der alten Settings kann problemlos von der aktuellen Version gelesen werden kann. Nach der Übernahme einer „alten“ XML ist diese wahrscheinlich mit den älteren Settings nicht mehr nutzbar.

Mit Hilfe des Buttons „Manage“ kommt man in einen Modus, in dem Einträge sowohl im Baum links als auch im rechten Bereich gelöscht werden können.  (Klick auf das rote Kreuz)



Ein Klick auf das  Icon exportiert das entsprechende Element in eine XML Datei.

Die exportierte Datei kann dann mit Hilfe der Funktion „ImportSettings“ (normalerweise auf Taste F2) importiert werden.

Beim Import werden schon bestehende Elemente überschrieben, nicht vorhandene Elemente werden erzeugt.



Default Typen, die von den Settings unterstützt werden

Es werden folgende Typen im Standard von den Settings unterstützt:

- String
- Boolean
- Byte
- Int16, Int32, Int64
- UInt16, UInt32, UInt64
- Decimal, Double, Single
- Color (System.Drawing.Color)
- Font (System.Drawing.Font)
- BAFileName
- BADirectoryName
- BAPlcVariableName
- BAStringListSelected

Weiter unten wird beschrieben, wie benutzerdefinierte Typen erstellt und genutzt werden können.

Erstellung benutzerspezifischer Editoren und Konverter

Die Settings können beliebig um benutzerspezifische Typen erweitert werden. Dazu ist es nötig, für jeden Typ einen Editor (WPF UserControl) und einen Konverter zu erstellen, der zwischen dem Typ und einem String konvertieren kann.

Aktuell sind folgende Typen implementiert:

```
BADirectoryName
BAFileName
Boolean
Byte
Color (aus System.Drawing)
Decimal
Double
Font
Int16, Int32, Int64, UInt16, UInt32, UInt64
String
```

Um einen eignen Typ zu implementieren, muss das Interface IBASettingsEntryVM implementiert und über Unity (z.B. einer Konfigurationsdatei in System\locConfig) bekannt gemacht werden. Zusätzlich zur Definition des Interface muss die Implementierung des IBASettingsEntryVM einen Konstruktor mit einem String Parameter mit Namen „ValueString“ besitzen, der aus einem String einen entsprechenden Typ erstellt.

```
namespace Beckhoff.App.Core.Interfaces.Settings
{
    /// <summary>
    /// Interface for an settings entry
    /// </summary>
    public interface IBASettingsEntryVM : INotifyPropertyChanged
    {
        /// <summary>
        /// Gets the string representation of the type.
        /// </summary>
        /// <value>
        string TypeString { get; }

        /// <summary>
        /// Gets the value as the defined type
        /// </summary>
        /// <value>
        object Val { get; set; }

        /// <summary>
        /// Gets the type which is handled by the setting
        /// </summary>
        Type Type { get; }

        /// <summary>
        /// Converts the type to its string representation.
        /// </summary>
        /// <param name="obj">The object.</param>
        /// <returns></returns>
        string ConvertToString();

        /// <summary>
        /// Gets the user control which is displayed in settings entry line
        /// </summary>
        /// <value>
        /// The WPF user control.
        /// </value>
        UserControl uControl { get; }
    }
}
```



Hier als Beispiel die Implementierung des Typs „Boolean“:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Controls;
using Beckhoff.App.Core.Interfaces.Settings;
using GalaSoft.MvvmLight;

namespace Beckhoff.App.Settings.UI.UserControlsType
{
    public class BASettingsTypeBoolean : ViewModelBase, IBASettingsEntryVM
    {
        private string _valueString;
        private Boolean _val;
        ucControlBoolean control;

        public BASettingsTypeBoolean(string ValueString)
        {
            if (!string.IsNullOrEmpty(ValueString))
            {
                valueString = ValueString;
                _val = Convert.ToBoolean(ValueString);
            }
        }

        public string TypeString
        {
            get { return "Boolean"; }
        }

        public UserControl uControl
        {
            get { return control ?? (control = new ucControlBoolean { DataContext = this }); }
        }

        public object Val
        {
            get { return _val; }
            set
            {
                if (value is Boolean)
                {
                    _val = (Boolean)value;
                    RaisePropertyChanged(() => Val);
                }
            }
        }

        public string ConvertToString()
        {
            return _val.ToString();
        }

        public Type Type
        {
            get { return typeof (Boolean); }
        }
    }
}
```

Die Zuordnung zum Typ Boolean erfolgt nun in der Unity Definitionen mit Hilfe eines Namens. Der Name muss dabei folgendermaßen aufgebaut sein: BASettings-„Typname“. In diesem konkreten Beispiel also: „BASettings-Boolean“

Die Unity Konfigurationsdatei im XML Format für den Typ Boolean sieht dann folgendermaßen aus:

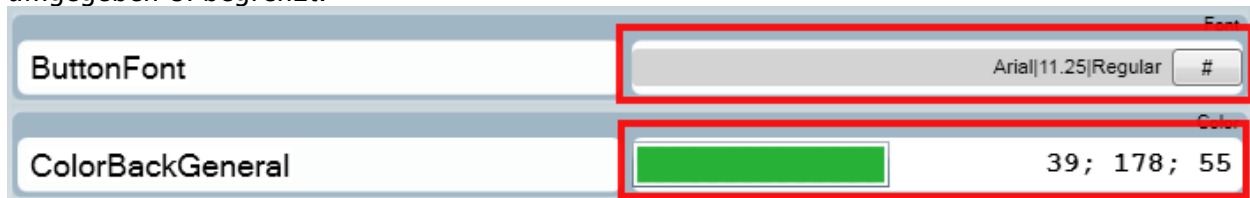
```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection,
Microsoft.Practices.Unity.Configuration" />
  </configSections>
  <unity>
    <alias alias="string" type="System.String, mscorlib" />
    <alias alias="singleton" type="Microsoft.Practices.Unity.ContainerControlledLifetimeManager,
Microsoft.Practices.Unity" />
    <alias alias="external" type="Microsoft.Practices.Unity.ExternallyControlledLifetimeManager,
Microsoft.Practices.Unity" />
    <container>

      <!-- Beckhoff Application Settings -->
      <type type="Beckhoff.App.Core.Interfaces.Settings.IBASettingsEntryVM, Beckhoff.App.Core.Interfaces"
        name="BASettings-Boolean" mapTo="Beckhoff.App.Settings.UI.UserControlsType.BASettingsTypeBoolean,
        Beckhoff.App.Settings.BASettings">

        </type>
      </container>
    </unity>
  </configuration>
```

In der Implementierung wird ein UserControl benötigt, dass über das Property „uControl“ abgerufen wird.

Dieses WPF UserControl wird im UI der Settings automatisch erzeugt und an der markierten Position eingesetzt. Die Höhe des Controls ist dabei nahezu beliebig, die Breite wird durch das umgebene UI begrenzt.



Das UserControl muss das Property „Val“ ändern und der Setter von „Val“ muss ein „PropertyChanged“ Event feuern, damit die Integration funktioniert.

Die Methode „ConvertToString“ erstellt eine String Darstellung des Types, die in der XML Datei der Settings gespeichert wird.

Beim Laden der Datei wird dieser String im Konstruktor wieder an die Klasse übergeben, die dann eine Konvertierung zurück vornehmen muss.



Eventaggregator

Mit Hilfe des EventAggregators (IBAEventAggregator) kann man sich auf ein BASettingsEntryChanges Event registrieren.

Bei jeder Änderung wird das entsprechende Event über den Eventaggregator verbreitet.

Bsp:

```
var ev = _eventAgg.GetEvent<BASettingsEntryChangedEvent>();
ev.Subscribe(SettingsEntryChanged, BATHreadOption.UIThread);
```

Die Methode SettingsEntryChanged wird nun bei jeder Änderung mit dem Parameter „baSettingsEntryData“ vom Typ BASettingsEntryData aufgerufen.

```
private void SettingsEntryChanged(BASettingsEntryData baSettingsEntryData)
```

BASettingsEntryData hat drei Properties, die den geänderten Eintrag beschreiben.

```
public string Section
public string Entry
public object Obj
```