**Reference**

Add reference to System.Data.DataSetExtensions.dll

**Settings.settings**

**ADD 2 Settings**

Name Type Scope Value

THSourceLanguage string user English

THTargetLanguage string user French



**Forms\MainView.cs**

    #region Docking Content

    private DeserializeDockContent mDeserializeDockContent;

    private PluginTreeContent pluginTreeContent = new PluginTreeContent();

    private SubrecordListContent subrecordListContent = new SubrecordListContent();

    private RichTextContent selectedTextContent = new RichTextContent();

    //Add for TranslatorHelper

    private TranslatorHelper translatorHelperContent = new TranslatorHelper();

    #endregion

    public MainView()

    {

      //Add for TranslatorHelper

      this.WindowState = FormWindowState.Maximized;

      AppDomain.CurrentDomain.UnhandledException += new UnhandledExceptionEventHandler(Program.CurrentDomain\_UnhandledException);

      Application.ThreadException += new System.Threading.ThreadExceptionEventHandler(Program.Application\_ThreadException);

      //End for TranslatorHelper

      if (!RecordStructure.Loaded)

      {

        try

        {

          RecordStructure.Load();

        }

        catch (Exception ex)

        {

          MessageBox.Show(Resources.CannotParseRecordStructure + ex.Message, Resources.WarningText);

        }

      }

      InitializeComponent();

      InitializeToolStripFind();

      InitializeToolStripTranslatorHelper(); //Add for TranslatorHelper

      InitializeDockingWindows();

      RegisterMessageFilter();

      uTF8ModeToolStripMenuItem.Checked = Properties.Settings.Default.UseUTF8; //Add for TranslatorHelper

      PluginTree.SelectionChanged += (o, e) => RebuildSelection();

    private void LayoutDockingWindows(bool force)

    {

      try

      {

        if (!force

            && IsVisible(pluginTreeContent)

            && IsVisible(subrecordListContent)

            && IsVisible(selectedTextContent)

            && IsVisible(translatorHelperContent)) //Add for TranslatorHelper

          return;

        dockPanel.SuspendLayout(true);

        if (force)

        {

          pluginTreeContent.DockPanel = null;

          subrecordListContent.DockPanel = null;

          selectedTextContent.DockPanel = null;

          translatorHelperContent = null;

        }

        if (!IsVisible(pluginTreeContent) || force)

        {

          pluginTreeContent.Show(dockPanel, DockState.DockLeft);

          dockPanel.Width = Math.Max(dockPanel.Width, pluginTreeContent.MinimumSize.Width);

        }

        if (!IsVisible(subrecordListContent) || force)

          subrecordListContent.Show(this.pluginTreeContent.Pane, DockAlignment.Bottom, 0.5);

        if (!IsVisible(selectedTextContent) || force)

          selectedTextContent.Show(dockPanel, DockState.Document);

        //Add for TranslatorHelper

        if (!IsVisible(translatorHelperContent) || force)

          translatorHelperContent.Show(dockPanel, DockState.Document);

        //End for TranslatorHelper

      }

      catch

      {

      }

      finally

      {

        dockPanel.ResumeLayout(true, true);

      }

    }

    private void ShowDockingWindows()

    {

      this.selectedTextContent.RtfInfo.LinkClicked += this.rtfInfo\_LinkClicked;

      this.selectedTextContent.RtfInfo.PreviewKeyDown += this.tbInfo\_PreviewKeyDown;

      this.pluginTreeContent.CloseButtonVisible = false;

      this.subrecordListContent.CloseButtonVisible = false;

      this.selectedTextContent.MdiParent = this;

      this.selectedTextContent.CloseButtonVisible = false;

      this.selectedTextContent.CloseButton = false;

      this.selectedTextContent.HideOnClose = true;

      //Add for TranslatorHelper

      this.translatorHelperContent.MdiParent = this;

      this.translatorHelperContent.CloseButtonVisible = false;

      this.translatorHelperContent.CloseButton = false;

      this.translatorHelperContent.HideOnClose = true;

      //End for TranslatorHelper

      LayoutDockingWindows(force: false);

    }

    private IDockContent GetContentFromPersistString(string persistString)

    {

      if (persistString == typeof(PluginTreeContent).ToString())

        return pluginTreeContent;

      if (persistString == typeof(SubrecordListContent).ToString())

        return subrecordListContent;

      if (persistString == typeof(RichTextContent).ToString())

        return selectedTextContent;

            //Add for TranslatorHelper

            if (persistString == typeof(TranslatorHelper).ToString())

              return translatorHelperContent;

            //End for TranslatorHelper

      return null;

    }

        private void InitializeLanguage()

        {

            languageToolBarItems.Add("English", englishToolStripMenuItem);

            languageToolBarItems.Add("Czech", czechToolStripMenuItem);

            languageToolBarItems.Add("French", frenchToolStripMenuItem);

            languageToolBarItems.Add("German", germanToolStripMenuItem);

            languageToolBarItems.Add("Italian", italianToolStripMenuItem);

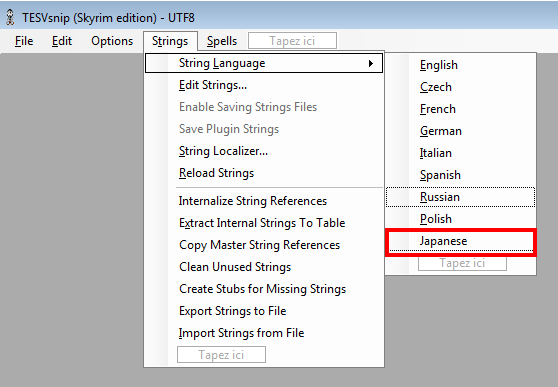
            languageToolBarItems.Add("Spanish", spanishToolStripMenuItem);

            languageToolBarItems.Add("Russian", russianToolStripMenuItem);

            languageToolBarItems.Add("Polish", polishToolStripMenuItem);

            languageToolBarItems.Add("Japanese", japaneseToolStripMenuItem);

        }



**Program.cs**

  internal static class Encoding

  {//add utf-8

    private static readonly System.Text.Encoding s\_CP1252Encoding = System.Text.Encoding.GetEncoding(1252);

    private static readonly System.Text.Encoding s\_UTF8Encoding = System.Text.Encoding.GetEncoding("utf-8");

    internal static System.Text.Encoding CP1252;

    private static readonly Dictionary<string, FontLangInfo> defLangMap =

        new Dictionary<string, FontLangInfo>(StringComparer.InvariantCultureIgnoreCase);

    static Encoding()

    {

      defLangMap.Add("English", new FontLangInfo(1252, 1033, 0));

      defLangMap.Add("Czech", new FontLangInfo(1252, 1029, 238));

      defLangMap.Add("French", new FontLangInfo(1252, 1036, 0));

      defLangMap.Add("German", new FontLangInfo(1252, 1031, 0));

      defLangMap.Add("Italian", new FontLangInfo(1252, 1040, 0));

      defLangMap.Add("Spanish", new FontLangInfo(1252, 1034, 0));

      defLangMap.Add("Russian", new FontLangInfo(1251, 1049, 204));

      defLangMap.Add("Polish", new FontLangInfo(1250, 1045, 0));

      defLangMap.Add("Japanese", new FontLangInfo(932, 1041, 128)); //128 => i'm not sure but i find on the web SHIFTJIS\_CHARSET = 128

      CP1252 = Properties.Settings.Default.UseUTF8 ? s\_UTF8Encoding : s\_CP1252Encoding;

      //http://www.tek-tips.com/viewthread.cfm?qid=712495

      //Private Const DEFAULT\_CHARSET = 1

      //Private Const SYMBOL\_CHARSET = 2

      //Private Const SHIFTJIS\_CHARSET = 128

      //Private Const HANGEUL\_CHARSET = 129

      //Private Const CHINESEBIG5\_CHARSET = 136

      //Private Const CHINESESIMPLIFIED\_CHARSET = 134

    }

**Model\Plugin.cs**

**ADD**

Line 45

    //Add for TranslatorHelper

    public string GetFileName()

    {

      return FileName;

    }

    //End for TranslatorHelper

Line 348

    public void ReloadStrings()

    {

      if (string.IsNullOrEmpty(StringsFolder) || string.IsNullOrEmpty(FileName) ||

          !Directory.Exists(StringsFolder))

        return;

    }

**CHANGE:**

    public void ReloadStrings()

    {

in

    public void ReloadStrings(string FileName, string Folder="")

    {

      if (String.IsNullOrWhiteSpace(StringsFolder)) StringsFolder = Folder;