XAML CS Files

Directory

[MainWindow](#MainWindow)

[CreatEditQA](#CreatEditQA)

[Home](#Home.xaml.cs)

[KeyWordControl](#KeyWordControlXamlCs)

[Link\_Note](#LinkNoteXamlCs)

[SubjectTree](#SubjectTree)

[TestReview](#TestReview)

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

MainWindow

using System.IO;

using System.Windows;

using System.Windows.Controls;

using NewLSP.StaticHelperClasses;

namespace NewLSP

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void miHome\_Click(object sender, RoutedEventArgs e)

{

SetActiveUserControl(ucHome);

}

private void miSubjectTree\_Click(object sender, RoutedEventArgs e)

{

if(SubjectStaticMembers.DisplayList.Count ==0)

{

MessageBox.Show("You cannot open this tab until you select a Subject Folder");

return;

}

if (QAStaticMembers.DictionaryChanged == true)

{

if (MessageBox.Show("There are unsaved changes to the QADictionary. Do you want to ignore them?", "Confirmation", MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

// Close the window

}

else

{

return;

}

}

SetActiveUserControl(ucSubjectTree);

}

private void miTest\_Click(object sender, RoutedEventArgs e)

{

if (SubjectStaticMembers.DataNode == null)

{

MessageBox.Show("You must select a DataNode before opening this window");

return;

}

if (QAStaticMembers.DictionaryChanged == true)

{

if (MessageBox.Show("There are unsaved changes to the QADictionary. Do you want to ignore them?", "Confirmation", MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

// Close the window

}

else

{

return;

}

}

//Check to make sure the QAFile exists and populate the dictionary and numbers list

if (QAStaticMembers.DoesQAFileExist())

{

TestReviewStaticMembers.InitializeData();

//StartNewQATestReview();

}

else

{

MessageBox.Show("There is no QAFile for this subject");

}

SetActiveUserControl(ucTestReview);

MessageBox.Show("You must select a Test/Review action first. If you want to randomize the questions select QuestionOrder -> Randomize first" );

}

public void SetActiveUserControl(UserControl control)

{

// Make the visibility of all user controls collapsed

ucHome.Visibility = Visibility.Collapsed;

ucSubjectTree.Visibility = Visibility.Collapsed;

ucCreatEditQA.Visibility = Visibility.Collapsed;

ucTestReview.Visibility = Visibility.Collapsed;

//ucInstructions.Visibility = Visibility.Collapsed;

ucLinkNote.Visibility = Visibility.Collapsed;

// Make the selected control visible

control.Visibility = Visibility.Visible;

}

/// <summary>

/// Called when the user clicks the QAPages tab.

/// If there is not data node a message is shown and control returns

/// Else SetActiveUserContro is called to open the ucCreateEditQA control

/// and a message to choose the edit Mode before proceeding is shown

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void CreateOrEdit\_Click(object sender, RoutedEventArgs e)

{

if (SubjectStaticMembers.DataNode == null)

{

MessageBox.Show("You must select a DataNode before opening this window");

return;

}

SetActiveUserControl(ucCreatEditQA);

MessageBox.Show("You must Choose the Edit Mode before Proceeding");

}

private void miCreateQA\_Click(object sender, RoutedEventArgs e)

{

if (SubjectStaticMembers.DataNode == null)

{

MessageBox.Show("You must select a DataNode before opening this window");

return;

}

}

private void miLinkNotes\_Click(object sender, RoutedEventArgs e)

{

bool HasHyperlink = false;

bool HadDataNodeReferenceFile = false;

SetActiveUserControl(ucLinkNote);

if (SubjectStaticMembers.DataNode != null)

{

//1. Test to see if this node has hyperlinks

// a. Create the filepath to the DataNodes HyperlinkFile

string DataNodesHyperlinkPath = CommonStaticMembers.HomeFolderPath + "Hyperlinks\\" + SubjectStaticMembers.DataNode.ID.ToString() + ".txt";

// b. Test to see if a hyperlink file exists

if (File.Exists(DataNodesHyperlinkPath))

{

HasHyperlink = true;

// Added- 20211020 The following was activated to try and populate the list of hyperlinks this works

LinkNoteStaticMembers.SetHyperlinkStringsList();

// End todo of activation 20211020

}

// c. Test to see if this data node has a DataNodeReference file and if so set HadDataNodeReferenceFile to true

string DataNodesReferenceFilePath = CommonStaticMembers.DataNodesNoteReferencesFilesPath;

if (File.Exists(DataNodesReferenceFilePath))

{

HadDataNodeReferenceFile = true;

}

if (HasHyperlink && !HadDataNodeReferenceFile)

{

MessageBox.Show("This node has a hyperlink file call Files -> Open Hyperlink");

}

else if(!HasHyperlink && HadDataNodeReferenceFile)

{

MessageBox.Show("This node has a DataNodeReference file call Files -> Open Notes");

}

else if (HasHyperlink && HadDataNodeReferenceFile)

{

MessageBox.Show("This node has a Notes file call Files -> Open Notes \r\n" +

"and a hyperlink file call Files -> Open Hyperlink")

}

}

}

private void miCloseApplication\_Click(object sender, RoutedEventArgs e)

{

// Save the KeyWords Dictionary file

KeyWordsStaticMembers.SaveDictionary();

Application curApp = Application.Current;

curApp.Shutdown();

}

}

}

MainWindow Methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access/Static | Return | CalledBy | Purpose |
| **MainWindow.xaml.cs** | | | | |
| [miHome\_Click](XAML%20CS%20Files.docx#miHome_Click) | private | void | miHome | NOTE: Comment Sets the active user control to ucHome |
| [miSubjectTree\_Click](XAML%20CS%20Files.docx#miSubjectTree_Click) | private | void | miSubjectTree | NOTE: Comment After making sure a Subject folder has been chosen it sets the active user control to ucSubjectTree |
| [CreateOrEdit\_Click](XAML%20CS%20Files.docx#CreateOrEdit_Click) | private | void | CreateOrEdit | Called when the user clicks the QAPages tab. If there is not data node a message is shown and control returns Else SetActiveUserContro is called to open the ucCreateEditQA control and a message to choose the edit Mode before proceeding is shown |
| [miTest\_Click](XAML%20CS%20Files.docx#miTest_Click) | private | void | miTest | NOTE: Comment After making sure a DataNode has been selected and that a QA file exists, it sets the active user control to ucTestReview |
| [miLinkNotes\_Click](XAML%20CS%20Files.docx#miLinkNotes_Click) | private | void | miLinkNotes | After making sure that a DataNode has been selected and that a Hyperlink file exists it this method opens a MessageBox informing the user that a Hyperlink exists and invites him to open it |
| [miCloseApplication\_Click](XAML%20CS%20Files.docx#miCloseApplication_Click) | private | void | miCloseApplication | NOTE: Comment Saves the KeyWord Dictionary and closes the application |
| [SetActiveUserControl](XAML%20CS%20Files.docx#SetActiveUserControl) | public | void | MainWindow.xaml.cs [miHome\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miHome_Click)  MainWindow.xaml.cs  [miSubjectTree\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miSubjectTree_Click)  MainWindow.xaml.cs  [CreateOrEdit\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#CreateOrEdit_Click)  MainWindow.xaml.cs  [miTest\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miTest_Click)  MainWindow.xaml.cs  [miLinkNotes\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miLinkNotes_Click) | NOTE: Comment When called changes the Visibility property of all User Controls so that only the desired UC is visible |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CreatEditQA

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using Microsoft.Win32;

using NewLSP.StaticHelperClasses;

using NewLSP.DataModels;

using System;

namespace NewLSP.UserControls

{ /// <summary>

/// Interaction logic for CreatEditQA.xaml

/// </summary>

public partial class CreatEditQA : UserControl

{

public CreatEditQA()

{

InitializeComponent();

}

#region Fields

private Dictionary<string, string> qaDictionary = new Dictionary<string, string>();

private QADataModel QADataModelObject = new QADataModel();

//--------------------------Booleans---------------------------------------//

// Edit mode is append to a new or existing file

private bool appendToFile = false;

// Edit mode is edit selected qa Pairs

private bool editSelectedQAPairs = false;

// Edit mode is edit All files Seriatem

private bool editAllSeriatem = false;

//--------------------------String---------------------------------------//

private string QuestionJpgUrl;

private string AnswerJpgUrl;

private string QuestionMp3Url;

private string AnswerMp3Url;

//private string currentQAPairStr = "";

#endregion Fields

#region Menu Items Click Methods

#region Files Menu

#region SaveFile

private void SaveFile\_Click(object sender, RoutedEventArgs e)

{

QAStaticMembers.SaveQADictionary();

MessageBox.Show("This QA File Saved.");

}

#endregion SaveFile

#region Append to File

private void Append\_Click(object sender, RoutedEventArgs e)

{

// Test to see if the dictionary is empty and if fill it and get the current item cound

if(QAStaticMembers.QADictionary.Count == 0)

{

TestReviewStaticMembers.InitializeData();

QAStaticMembers.CurrentQANumberInt = QAStaticMembers.QADictionary.Count;

tbkCurrentQuestionNumber.Text = QAStaticMembers.CurrentQANumberInt.ToString();

}

}

#endregion Append to File

#endregion Files Menu

#region Edit Menu

#region Begin New QAFile

/// <summary>

/// Called when the user clicks the Begin a New File Menu item

/// It then uses a Yes/No MessageBox to ask if the user wants to override it.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void NewFile\_Click(object sender, RoutedEventArgs e)

{

//Check to insure that the file doesn't exist and then create it

if (QAStaticMembers.DoesQAFileExist())

{

MessageBoxResult result = MessageBox.Show("This QA Node already has a QAFile, do you want to override it", "Yes", MessageBoxButton.YesNo);

switch(result)

{

case MessageBoxResult.Yes:

QAStaticMembers.CreateNewQAFile();

break;

case MessageBoxResult.No:

MessageBox.Show("Then Select append");

break;

}

QADataModelObject = new QADataModel();

}

else

{

QAStaticMembers.CreateNewQAFile();

}

int CurrentQANumberInt = QAStaticMembers.CurrentQANumberInt;

tbkCurrentQuestionNumber.Text = CurrentQANumberInt.ToString();

}

#endregion Begin New QAFile

#endregion Edit Menu

private void miQuestionJpgUrl\_Click(object sender, RoutedEventArgs e)

{

QuestionJpgUrl = ReturnJpgUrl();

}

private void miAnswerJpgUrl\_Click(object sender, RoutedEventArgs e)

{

AnswerJpgUrl = ReturnJpgUrl();

}

private void miQuestionMp3Url\_Click(object sender, RoutedEventArgs e)

{

QuestionMp3Url = ReturnMp3Url();

}

private void miAnswerMp3Url\_Click(object sender, RoutedEventArgs e)

{

AnswerMp3Url = ReturnMp3Url();

}

#endregion Menu Items Click Methods

private void btnGetNextQA\_Click(object sender, RoutedEventArgs e)

{

AddThisQAPairToTheDictionary();

//send the current quesntion number string to tbkCurrentQuestionNumber

tbkCurrentQuestionNumber.Text = QAStaticMembers.CurrentQANumberInt.ToString();

// Clear the controls data and data string

tbxQuestion.Text = "";

tbxAnswer.Text = "";

QuestionJpgUrl = "";

QuestionMp3Url = "";

AnswerJpgUrl = "";

AnswerMp3Url = "";

}

#region Private Methods

#region Open JPG FIles

#endregion Open JPG FIles

private static string ReturnJpgUrl()

{

string JpgUrlPath = "";

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Jpg files (\*.jpg)|\*.jpg";

if (openFileDialog.ShowDialog() == true)

{

JpgUrlPath = openFileDialog.FileName;

}

return JpgUrlPath;

}

private static string ReturnMp3Url()

{

string Mp3UrlPath = "";

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Mp3 files (\*.mp3)|\*.mp3";

if (openFileDialog.ShowDialog() == true)

{

Mp3UrlPath = openFileDialog.FileName;

}

return Mp3UrlPath;

}

#region AddThisQAPairToTheDictionary

private void AddThisQAPairToTheDictionary()

{

// Eliminate any blank lines and replace line feeds with a tilda

string thisQuestion = tbxQuestion.Text;

thisQuestion = thisQuestion.Replace("\r\n", "~");

string [] thisQuestionArray = thisQuestion.Split('~');

thisQuestion = "";

foreach(string line in thisQuestionArray)

{

if(line != "")

{

thisQuestion = thisQuestion +line+ "~";

}

}

// Prepare the first question

// delete the terminal tilds

try

{

thisQuestion = thisQuestion.Substring(0, thisQuestion.Length - 1);

}

catch (Exception ex)

{

MessageBox.Show("You cannot process and empty question!");

return;

}

string thisAnswer = tbxAnswer.Text;

thisAnswer = thisAnswer.Replace("\r\n", "~");

string[] thisAnswerArray = thisAnswer.Split('~');

thisAnswer = "";

foreach (string line in thisAnswerArray)

{

if (line != "")

{

thisAnswer = thisAnswer + line + "~";

}

}

// delete the terminal tilds

thisAnswer = thisAnswer.Substring(0, thisAnswer.Length - 1);

QADataModelObject.Question = thisQuestion;

QADataModelObject.Answer = thisAnswer;

QADataModelObject.QuestionJpgUrl = QuestionJpgUrl;

QADataModelObject.QuestionMp3Url = QuestionMp3Url;

QADataModelObject.AnswerJpgUrl = AnswerJpgUrl;

QADataModelObject.AnswerMp3Url = AnswerMp3Url;

// Save the current QADataModelObject

QAStaticMembers.AddQAObjectToDictionary(QAStaticMembers.CurrentQANumberInt.ToString(), QADataModelObject);

// Clear this object and create a new one

QADataModelObject = null;

QADataModelObject = new QADataModel();

// Increment the current question number

QAStaticMembers.CurrentQANumberInt++;

}

#endregion AddThisQAPairToTheDictionary

#endregion Private Methods

}// End Class

}// End namespace

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access/Static | Return | CalledBy | Purpose |
| **CreatEditQA.xaml.cs** | | | | |
| [SaveFile\_Click](#SaveFile_ClickCreatEditQa)( | private | void | CreateEdit.xaml.cs SaveFile | NOTE: Comment QADataModelObject Saves the QADictionary |
| [NewFile\_Click](XAML%20CS%20Files.docx#NewFile_Click)( | private | void | CreateEdit.xaml.cs  NewFile | NOTE: Comment Called when the user clicks the Begin a New File Menu item It then uses a Yes/No MessageBox to ask if the user wants to override it. |
| [Append\_Click](XAML%20CS%20Files.docx#Append_Click)( | private | void | CreateEdit.xaml.cs  Append | NOTE: Comment Check to see if the QAStaticMembers.QADictionary.Count == 0 and if so set the tbkCurrentQuestionNumber.Text to the Count  NOTE: Investigate??? |
| [miQuestionJpgUrl\_Click](XAML%20CS%20Files.docx#miQuestionJpgUrl_Click)( | private | void | CreateEdit.xaml.cs  miQuestionJpgUrl | NOTE: Comment Sets the local variable QuestionJpgUrl to the value returned by the private static method ReturnJpgUrl() |
| [miAnswerJpgUrl\_Click](XAML%20CS%20Files.docx#miAnswerJpgUrl_Click)( | private | void | CreateEdit.xaml.cs  miAnswerJpgUrl | NOTE: Comment Sets the local variable AnswerJpgUrl to the value returned by the private static method ReturnJpgUrl() |
| [miQuestionMp3Url\_Click](XAML%20CS%20Files.docx#miQuestionMp3Url_Click)( | private | void | CreateEdit.xaml.cs  miQuestionMp3Url | NOTE: Comment Sets the local variable QuestionMp3Url to the value returned by the private static method ReturnMp3Url(); |
| [miQuestionMp3Url\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miQuestionMp3Url_Click)( | private | void | CreateEdit.xaml.cs  miAnswerMp3Url | NOTE: Comment Sets the local variable AnswerMp3Url to the value returned by the private static method ReturnMp3Url(); |
| [btnGetNextQA\_Click](XAML%20CS%20Files.docx#btnGetNextQA_Click)( | private | void | CreateEdit.xaml.cs  btnGetNextQA | NOTE: Comment Calls AddThisQAPairToTheDictionary() to add the adjusted question and answer strings as well as all media links to the QADictionary |
| [ReturnJpgUrl](#ReturnJpgUrl)( | private static | string | CreateEdit.xaml.cs  [miQuestionJpgUrl\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miQuestionJpgUrl_Click)(  CreateEdit.xaml.cs  [miAnswerJpgUrl\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miAnswerJpgUrl_Click)( | NOTE: Comment Calls the OpenFileDialog() with .jpg extension filter to select a jpg URL and use it to set the local variable JpgUrlPath |
| [ReturnMp3Url(](#ReturnMp3Url) | private static | string | CreateEdit.xaml.cs  [miQuestionMp3Url\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miQuestionMp3Url_Click)(  CreateEdit.xaml.cs  [miQuestionMp3Url\_Click](file:///D:\Users\Owner\OneDrive\Documents\2021CSharpProjects\Dated%20Backup%20Files%20for%20NewLSP\_Maps%20and%20Links\XAML%20CS%20Files.docx#miQuestionMp3Url_Click)( | NOTE: Comment Calls the OpenFileDialog() with .mp3 extension filter to select a jpg URL and use it to set the local variable Mp3UrlPath |
| [AddThisQAPairToTheDictionary](#AddThisQAPairToTheDictionaryDefinition)() | private | void |  | NOTE: Comment QADataModelObject Adjusts the Question and Answer strings by replacing \n\r with ~ and creates a QADataModelObject which will hold the question and answer strings as well as all media links |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**[Home.xaml.cs](http://Home.xaml.cs)**

using Microsoft.WindowsAPICodePack.Dialogs;

using NewLSP.StaticHelperClasses;

using System.Windows;

using System.Windows.Controls;

using System.IO;

using NewLSP.DataModels;

using System.Collections.Generic;

using System;

namespace NewLSP.UserControls

{

/// <summary>

/// Interaction logic for Home.xaml

/// </summary>

public partial class Home : UserControl

{

public Home()

{

InitializeComponent();

}

/// <summary>

/// This is the first method called when the user begins the application

/// It forces the user to either create a Subjects folder and a References folder

/// of to select such folders that have been previously created

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnOpenSubjectFolder\_Click(object sender, RoutedEventArgs e)

{

//Create a lable so that the user can return here if needed

SubjectFolder: MessageBox.Show("Create or Open an existing -- S U B J E C T -- folder");

// Get or create the Name of the Subject folder

CommonOpenFileDialog dialog = new CommonOpenFileDialog();

dialog.IsFolderPicker = true;

string FolderPath = "";

if (dialog.ShowDialog() == CommonFileDialogResult.Ok)

{

FolderPath = dialog.FileName + '\\';

CommonStaticMembers.HomeFolderPath = FolderPath;

}

// Get the number of '\\'s in FolderPath

var NumberOfSlashes = StringHelper.ReturnNumberOfDeliniters(FolderPath, '\\');

// Get the Subjects Name from the item a position NumberOfSlashes -1

var FolderName = StringHelper.ReturnItemAtPos(FolderPath, '\\', NumberOfSlashes - 1);

lblTitle.Content = "This is the Subjects Tree for " + FolderName;

//Use the FolderName to name the Subject, and its main data files

string SubjectName = FolderName;

// Create path to this subjects data file

string SubjectsNodeDataStringsPath = CommonStaticMembers.HomeFolderPath + "NodeDataStrings.txt";

CommonStaticMembers.SubjectsNodeDataStringsPath = SubjectsNodeDataStringsPath;

// Test to see if this file exist and if not create it

if (!File.Exists(SubjectsNodeDataStringsPath))

{

// Tell the user he is creating a new subject folder and allow him to escape if necessary

if (MessageBox.Show("Do you want to create a new Subject folder?", "Question", MessageBoxButton.YesNo, MessageBoxImage.Warning) == MessageBoxResult.No)

{

FolderPath = "";

CommonStaticMembers.HomeFolderPath = FolderPath;

goto SubjectFolder;

}

//Call the create new NodeDataStrings.txt file and add a Root node to it

CreateNewSubjectFolderFilesAndfolders();

}// End Test to see if this file exist and if not create it

else //open and read existing ItemCount.txt and NodeDataStrings.txt

{

SubjectNodes RootNode = new SubjectNodes();

// Read in the current item count

string ItemCount = File.ReadAllText(CommonStaticMembers.HomeFolderPath + "ItemCount.txt");

SubjectStaticMembers.ItemCount = ItemCount;

// Instantiate the Dictionary

SubjectStaticMembers.SubjectNodeDictionary = new Dictionary<string, SubjectNodes>();

// Instantiate a new node

SubjectNodes ThisNode = new SubjectNodes();

// Create the delimiter

char D = '\u0240';

//Read in SubjectsNodeDataStringsPath

string[] SubjectNodeDataStringArray = File.ReadAllLines(SubjectsNodeDataStringsPath);

foreach (string line in SubjectNodeDataStringArray)

{

// get the properties of a SubjectNode

string[] ItemsInLine = line.Split(D);

ThisNode.LeadingChars = ItemsInLine[0];

ThisNode.CI = ItemsInLine[1];

ThisNode.TitleText = ItemsInLine[2];

ThisNode.NodeLevelName = ItemsInLine[3];

string IDString = ItemsInLine[4];

ThisNode.ID = Int32.Parse(IDString);

string NOCString = ItemsInLine[5];

ThisNode.NOC = Int32.Parse(NOCString);

string HasDataString = ItemsInLine[6];

if (HasDataString == "false")

{

ThisNode.HasData = false;

}

else

{

ThisNode.HasData = true;

}

ThisNode = new SubjectNodes();

}// End open and read existing ItemCount.txt and NodeDataStrings.txt

SubjectStaticMembers.DisplayParentsAndChildren("\*");

}// End if else file subject file exists

btnOpenSubjectFolder.IsEnabled = false;

//Communicate the FolderPath to the ViewModel.SubjectNodeViewModel's OpenFile method

SubjectStaticMembers.OpenFiles(FolderPath);

// Show a message telling the user to create or select a common references folder and

// create a lable allowing the user to return here if desirec

References: MessageBox.Show("Create of Open a Folder to hold -- R E F E R E N C S --. Normally, this will be in a more generic"+

" folder and its name will reflect the Generic Interest. for example 'Religion References'");

/\*

\* Either select an existing Common References folder or create one

\* using CommonOpenFileDialog and its FileName property to

\* Create a ReferenceFolderPath

\*/

CommonOpenFileDialog referenceDialog = new CommonOpenFileDialog();

referenceDialog.IsFolderPicker = true;

string ReferenceFolderPath = "";

if (referenceDialog.ShowDialog() == CommonFileDialogResult.Ok)

{

//The path to the Common Rererence Folder

ReferenceFolderPath = referenceDialog.FileName + '\\';

}

// Create "NoteReferenceFiles" and "CompositData" folders

// Create a path to the NoteReferenceFiles folder in the Common References folder

string NoteReferenceFilesPath = ReferenceFolderPath + "NoteReferenceFiles";

//Check to see if this folder(Directory) exists and if not create it

if (!Directory.Exists(NoteReferenceFilesPath))

{

// Tell the user he is creating a new references folder and allow him to escape if necessary

if (MessageBox.Show("Do you want to create a new References folder?", "Question", MessageBoxButton.YesNo, MessageBoxImage.Warning) == MessageBoxResult.No)

{

ReferenceFolderPath = "";

goto References;

}

Directory.CreateDirectory(NoteReferenceFilesPath);

}

// Set the NoteReferenceFilesPath

CommonStaticMembers.NoteReferencesPath = NoteReferenceFilesPath;

// Create a pathe to the CompositData folder that will hold data about Key Words

string CompositDataPath = ReferenceFolderPath + "CompositData";

//If this folder doesn't exist create it

if (!Directory.Exists(CompositDataPath))

{

Directory.CreateDirectory(CompositDataPath);

}

// set the CompositDataPath

SubjectStaticMembers.CompositDataPath = CompositDataPath;

// If it doesn't exist create the NotesDictionary file

string KeyWordsDictionaryPath = CompositDataPath + "\\KeyWordsDictionary.txt";

if (!File.Exists(KeyWordsDictionaryPath))

{

var fileStream = File.Create(KeyWordsDictionaryPath);

fileStream.Close();

}

// Set CommonStaticMembers.KeyWordsDictionaryPath

CommonStaticMembers.KeyWordsDictionaryPath = KeyWordsDictionaryPath;

string ListOfKeyWordsPath = CompositDataPath + "\\ListOfKeyWords.txt";

if (!File.Exists(ListOfKeyWordsPath))

{

var fileStream = File.Create(ListOfKeyWordsPath);

fileStream.Close();

}

// Set the ListOfKeyWordsPath in KeyWordsStaticMembers

KeyWordsStaticMembers.ListOfKeyWordsPath = ListOfKeyWordsPath;

// Create a Sorted KeyWord List

string ListOfSortedKeyWordsPath = CompositDataPath + "\\SortedListOfKeyWords.txt";

// Insure that the file doesn't already exist

if (!File.Exists(ListOfSortedKeyWordsPath))

{

var fileStream = File.Create(ListOfSortedKeyWordsPath);

fileStream.Close();

}

// Set the SortedListOfKeyWordsPath in KeyWordsStaticMembers

KeyWordsStaticMembers.ListOfSortedKeyWordsPath = ListOfSortedKeyWordsPath;

// Set the initial value of CurrentNoteIDInt

MessageBox.Show("Click SubjectTreePage and then Click SHOW DISPLAY LIST to see all of the Base subjects for this project");

CommonStaticMembers.CurrentNoteIDInt = -1;

}// End btnOpenSubjectFolder\_Click

/// <summary>

/// The purpose of this method is for the user to

/// either open a previously designated instructions

/// folder or create it and then copy all of

/// the instructions text files into it.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

// 202108251129 inactivated this section

private void btnSetInstructionsFolder\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Either select a previously chosen Instructions folder, " +

"or create it and then manually add all of the text files in the accompanying InstructionsTextFiles to it. ");

// Get the Name of the Instructions folder

CommonOpenFileDialog dialog = new CommonOpenFileDialog();

dialog.IsFolderPicker = true;

string InstructionsFolderPath = "";

if (dialog.ShowDialog() == CommonFileDialogResult.Ok)

{

InstructionsFolderPath = dialog.FileName + '\\';

}

InstructionsStaticMembers.InstructionsFolderPath = InstructionsFolderPath;

}

/// <summary>

/// This private method is called when the user wants to create a new folder

/// </summary>

private void CreateNewSubjectFolderFilesAndfolders()

{

CreateNewNodeDataStringsFile();

CreateItemCountTextFile();

CreateQAFilesFolder();

CreateQAResultFolder();

CreateHyperlinksFolder();

CreateDataNodesNoteReferencesFilesFolder();

}

/// <summary>

/// This is the path to the SubjectFolder’s NodeDataStrings.txt Text file

/// </summary>

private void CreateNewNodeDataStringsFile()

{

// Create path to this subjects data file

CommonStaticMembers.SubjectsNodeDataStringsPath = CommonStaticMembers.HomeFolderPath + "NodeDataStrings.txt";

//Create a new RootNode

SubjectNodes RootNode = new SubjectNodes(0);

// Assign the CurrentItemCount to the Root node's ID

RootNode.ID = 0;

string ItemCount = "1";

//Write the new ItemCount to the ItemCount.txt file

File.WriteAllText(CommonStaticMembers.HomeFolderPath + "ItemCount.txt", ItemCount.ToString());

//Set up the properties of the new RootNode

RootNode.CI = "- ";

RootNode.NodeLevelName = "\*";

int LengthNodeLevelName = RootNode.NodeLevelName.Length;

string LeadingChars = new string(' ', LengthNodeLevelName);

RootNode.LeadingChars = LeadingChars;

RootNode.NOC = 0;

RootNode.TitleText = "Root";

RootNode.HasData = false;

// Create Root Node Data String and write it to the NodeDataStrings.txt file

char D = '\u0240';

string RootNodeDataString = RootNode.LeadingChars + D + RootNode.CI + D + RootNode.TitleText + D + RootNode.NodeLevelName + D +

RootNode.ID + D + RootNode.NOC + D + RootNode.HasData;

// Write RootNodeDataString to the NodeDataStrings.txt file

File.WriteAllText(CommonStaticMembers.SubjectsNodeDataStringsPath, RootNodeDataString);

//Add this rootnode to the SubjectNodeDictionary

SubjectStaticMembers.SubjectNodeDictionary.Add(RootNode.NodeLevelName, RootNode);

// Create the DisplayString for the root node

string RootDisplayString = $"{RootNode.LeadingChars}{RootNode.CI}{RootNode.TitleText}";

//Add this root node to the DisplayList

SubjectStaticMembers.DisplayList.Add(RootDisplayString);

// Add the root Node's NodeLevelName to the DisplaySubjectNodesList

SubjectStaticMembers.SubjectNodesLevelNameList.Add(RootNode.NodeLevelName);

}

/// <summary>

/// Creates the ItemCpunt.txt file

/// Called by CreateNewSubjectFolderFilesAndfolders()

/// </summary>

private void CreateItemCountTextFile()

{

string ItemCount = "1";

File.WriteAllText(CommonStaticMembers.HomeFolderPath + "ItemCount.txt", ItemCount.ToString());

// Write the item count to SubjectStaticMembers.ItemCount

SubjectStaticMembers.ItemCount = ItemCount;

CommonStaticMembers.ItemCountPath = CommonStaticMembers.HomeFolderPath + "ItemCount.txt";

}

private void CreateQAFilesFolder()

{

//Create a folder to hold the QAResults and path

if (!Directory.Exists(CommonStaticMembers.HomeFolderPath + "QAFiles"))

{

Directory.CreateDirectory(CommonStaticMembers.HomeFolderPath + "QAFiles");

CommonStaticMembers.DataNodesQAResultsFilePath = CommonStaticMembers.HomeFolderPath + "QAFiles";

}

}

/// <summary>

/// Creates the QAReslts folder

/// Is called by CreateNewSubjectFolderFilesAndfolders()

/// </summary>

private void CreateQAResultFolder()

{

//Create a folder to hold the QAResults and path

if (!Directory.Exists(CommonStaticMembers.HomeFolderPath + "QAResults"))

{

Directory.CreateDirectory(CommonStaticMembers.HomeFolderPath + "QAResults");

CommonStaticMembers.DataNodesQAResultsFilePath = CommonStaticMembers.HomeFolderPath + "QAResults";

}

}

/// <summary>

/// Creates the Hyperlinks folder

/// Is called by CreateNewSubjectFolderFilesAndfolders()

/// </summary>

private void CreateHyperlinksFolder()

{

//Create a folder to hold the hyperlinks files and path

if (!Directory.Exists(CommonStaticMembers.HomeFolderPath + "Hyperlinks"))

{

Directory.CreateDirectory(CommonStaticMembers.HomeFolderPath + "Hyperlinks");

CommonStaticMembers.DataNodesHyperlinksPath = CommonStaticMembers.HomeFolderPath + "Hyperlinks";

}

}

/// <summary>

/// Creates the DataNodesReferencesFiles folder

/// Is called by CreateNewSubjectFolderFilesAndfolders()

/// </summary>

private void CreateDataNodesNoteReferencesFilesFolder()

{

// Create a folder to hold the subject DataNodesNoteReferencesFiles and path

if (!Directory.Exists(CommonStaticMembers.HomeFolderPath + "DataNodesNoteReferencesFiles"))

{

Directory.CreateDirectory(CommonStaticMembers.HomeFolderPath + "DataNodesNoteReferencesFiles");

CommonStaticMembers.DataNodesNoteReferencesFilesPath = CommonStaticMembers.HomeFolderPath + "DataNodesNoteReferencesFiles";

}

}

}// End partial class Home

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access/Static | Return | CalledBy | Purpose |
| **Home.xaml.cs** | | | | |
| [btnOpenSubjectFolder\_Click](#HUCbtnOpenSubjectFolder_ClickDef) | private | void | btnOpenSubjectFolder | This is the first method called when the user begins the application It forces the user to either create a Subjects folder and a References folder of to select such folders that have been previously created |
| [CreateNewSubjectFolderFilesAndfolders](#CreateNewSubjectFolderFilesAndfolderDef)() | private | void | btnOpenSubjectFolder\_Click | This private method is called when the user wants to create a new folder It calls”  CreateNewNodeDataStringsFile();  CreateItemCountTextFile();  CreateQAFilesFolder();  CreateQAResultFolder();  CreateHyperlinksFolder();  CreateDataNodesNoteReferencesFilesFolder() |
| [CreateNewNodeDataStringsFile](#CreateNewNodeDataStringsFile)(); | private | void | CreateNewSubjectFolderFilesAndfolders() | NOTE: Comment Creates a new NodeDataString.txt file and populates it with a Root node |
| [CreateItemCountTextFile](#HSMCreateItemCountTextFileDef)() | private | void | CreateNewSubjectFolderFilesAndfolders( | Creates the ItemCpunt.txt file |
| [CreateQAFilesFolderDef](#CreateQAFilesFolderDef)() |  |  | CreateNewSubjectFolderFilesAndfolders( | Create a folder to hold the QAResults and path |
| [CreateQAResultFolder](#CreateQAResultFolderDef)() | private | void | CreateNewSubjectFolderFilesAndfolders( | Creates the QAReslts folder |
| [CreateHyperlinksFolder()](#CreateHyperlinksFolderDef) | private | void | CreateNewSubjectFolderFilesAndfolders( | Creates the Hyperlinks folder |
| [CreateDataNodesNoteReferencesFilesFolder](#CreateDataNodesNoteReferencesDef)( | private | void | CreateNewSubjectFolderFilesAndfolders( | Creates the DataNodesReferencesFiles folder |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**KeyWordControl**

[Methods Table](#KeyWordControlMethodsTable)

using System;

using System.Collections.Generic;

using System.IO;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using NewLSP.StaticHelperClasses;

using System.Windows.Controls;

using NewLSP.UserControls;

namespace NewLSP.UserControls

{

/// <summary>

/// Interaction logic for KeyWordControl.xaml

/// </summary>

public partial class KeyWordControl : UserControl

{

private object ucLinkNote;

public KeyWordControl()

{

InitializeComponent();

}

#region RadioButton Add

private void rbtAdd\_Click(object sender, RoutedEventArgs e)

{

btnRevert.Visibility = Visibility.Hidden;

KeyWordsStaticMembers.ListAccess = true;

lblKeyWordsAction.Content = "Add Key Words to a New Note Reference";

btnRevert.Content = "Revert";

}

#endregion RadioButton Add

#region Radio button Search

#endregion Radio button Search

#region tbxInput\_KeyUp

/// <summary>

/// Take the characters typed into lbxKeyWords and show all terms in KeyWordList that start with these characters

/// If the User Hits the Enter Key

/// a. If there are itemn in the list, in either mode return the top item in the list

/// b. If there are no Items

/// 1) In search, Warn and return

/// 2) In Create, create a new KeyWord from the characters in the textbox

///

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void tbxInput\_KeyUp(object sender, KeyEventArgs e)

{

//Clear the current content of lbxKeyWords

lbxKeyWords.Items.Clear();

// Cycle through KeyWordList selecting all that begin with the characters in tbxInput

foreach (string line in KeyWordsStaticMembers.KeyWordList)

{

//In the current line begins with the characters in tbxInput add line to lbxKeyWords

if (line.IndexOf(tbxInput.Text) == 0)

{

lbxKeyWords.Items.Add(line);

}

}

//If the user hits the Enter key

if (e.Key == Key.Enter)

{

if (!KeyWordsStaticMembers.ListAccess)

{

// If lbxKeyWords is empty in the Search statge send a message that you can only accepts existing keywords

if (lbxKeyWords.Items.Count == 0)

{

MessageBox.Show("When You are in the Search mod you can only search for existing Keywords");

// return to UI

return;

}

}

string KeyWord = "";

//Determine if there are Keywords showing in lbxKeyWords and if so select #0 and return

if (lbxKeyWords.Items.Count != 0)

{

// Create a list from the KeyWords in lbxKeyWords so that the 0th item can be chosen

List<string> myCurrentKeyWordsList = new List<string>();

// Populate myCurrentKeyWordsList with the selected columns.

foreach (string thisKeyWordItem in lbxKeyWords.Items)

{

myCurrentKeyWordsList.Add(thisKeyWordItem);

}

//Set the Selected KeyWord to the 0th entry

KeyWord = myCurrentKeyWordsList[0];

// add this Keyword to the list of selected keywords

tbxAllKeyWords.Text = tbxAllKeyWords.Text + KeyWord + ';';

//Clear tbxInput

tbxInput.Text = "";

//Clear lbxKeyWords

lbxKeyWords.Items.Clear();

//retrun to UI

return;

}

// Create a new KeyWord from the current text in tbxInput

KeyWord = tbxInput.Text;

KeyWord = KeyWord.Trim();

// Add this KeyWord to tbxAllKeyWords

tbxAllKeyWords.Text = tbxAllKeyWords.Text + KeyWord + ';';

// If this is a generic(ie it begins with #) return without adding it to the KeyWordList

if (KeyWord.IndexOf("#") != -1)

{

tbxInput.Text = "";

return;

}

// Update the active KeyWordList

KeyWordsStaticMembers.KeyWordList.Add(tbxInput.Text);

// Append this new Keyword to the Keyword Fild

KeyWordsStaticMembers.AppendNewKeyWord(KeyWord);

// Convert Keyword to Dictionary Item

string thisKeyWord = tbxInput.Text;

string ConvertedThisKeyWord = thisKeyWord.Replace(' ', '\_');

//Add the new converted Key word to the dictionary with a value containing only the starting delimiter, ;

KeyWordsStaticMembers.KeyWordsDictionary.Add(ConvertedThisKeyWord, ";");

// Add the new converted keyword to the NotesDictionary.txt file

KeyWordsStaticMembers.AppendNewKeyWordDictionaryItemString(ConvertedThisKeyWord);

tbxInput.Text = "";

}

// Code to clear tbxInput if backspace results in empty text

if (e.Key == Key.Back)

{

if (tbxInput.Text == "") lbxKeyWords.Items.Clear();

}

}

#endregion tbxInput\_KeyUp

#region bxKeyWords\_MouseLeftButtonUp

private void lbxKeyWords\_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

string KeyWord = lbxKeyWords.SelectedItem.ToString();

tbxAllKeyWords.Text = tbxAllKeyWords.Text + KeyWord + ';';

tbxInput.Text = "";

lbxKeyWords.Items.Clear();

}

#endregion bxKeyWords\_MouseLeftButtonUp

#region tbxAllKeyWords\_TextChanged

/// <summary>

/// This method is called whenever the text in the tbxAllKeyWords

/// is changed if the user in not in the Search mode then the program returns

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void tbxAllKeyWords\_TextChanged(object sender, TextChangedEventArgs e)

{

//Get the entry after the ';'

int posSemi = tbxAllKeyWords.Text.IndexOf(';');

string frontItem = "";

string backItem = "";

List<string> frontList = new List<string>();

// Determine if this is the last character, ie there is only a key word entered

if (posSemi == tbxAllKeyWords.Text.Length-1)

{

// This is the originay keyword

// Display all NoteNames whose notes contain this Key word

frontItem = tbxAllKeyWords.Text.Substring(0, tbxAllKeyWords.Text.Length - 1);

string[] InitialKeyWordNotesArray = KeyWordsStaticMembers.delimitedStringOfNoteNames(frontItem).Split(';');

// Create a List<string> of these note names eliminating blank entries

for (int i = 0; i < InitialKeyWordNotesArray.Length; i++)

{

if (InitialKeyWordNotesArray[i] != "")

{

frontList.Add(InitialKeyWordNotesArray[i]);

}

}

// Create a Dictionary<string,string> MatchingNoteRefsDictionary where the Key is the NoteIDName and the Value is the Note

Dictionary<string, string> MatchingNoteRefsDictionary = new Dictionary<string, string>();

foreach (string NoteIDName in frontList)

{

string RefNoteName = NoteIDName + ".txt";

try

{

string NoteFilePath = CommonStaticMembers.NoteReferencesPath + RefNoteName;

if (File.Exists(NoteFilePath))

{

string NoteContent = File.ReadAllText(NoteFilePath);

MatchingNoteRefsDictionary.Add(NoteIDName, NoteContent);

}

}

catch (Exception ex)

{

MessageBox.Show("There is not note for NoteIDName");

return;

}

//Cycle thorugh the Dictionary displaying all note Names in lbxOpenSelectedNote

foreach(KeyValuePair< string, string> KVP in MatchingNoteRefsDictionary)

{

string NoteID = KVP.Key;

string Value = KVP.Value;

string NoteName = StringHelper.ReturnItemAtPos(Value, '^', 0);

}

}

}

else

{

frontItem = tbxAllKeyWords.Text.Substring(0, posSemi - 1);

backItem = tbxAllKeyWords.Text.Substring(posSemi + 1);

//Determine if frontItem is a KeyCombination or a KeyWord

if(frontItem.IndexOf(" + ") != -1)

{

//This is a keyword combination get its list of notereferences from the KeyCombinationsDictionary

}

else

{

//This is the original keyword get its note references from the KeyWordsDictionary

string[] InitialKeyWordNotesArray = KeyWordsStaticMembers.delimitedStringOfNoteNames(frontItem).Split(';');

// Create a List<string> of these note names eliminating blank entries

for (int i=0; i< InitialKeyWordNotesArray.Length; i++)

{

if (InitialKeyWordNotesArray[i] != "")

{

frontList.Add(InitialKeyWordNotesArray[i]);

}

}

}

}

}// End tbxAllKeyWords\_TextChanged

#endregion tbxAllKeyWords\_TextChanged

#region Revert Button Clicked

/// <summary>

/// This button is visible only when the user is in the

/// search mode. when clicked it removes the last

/// KeyWord from the KeyComparison and

/// displays it in the tbxAllKeyWords

/// It also changes the display in the lbxOpenSelectedNote

/// to only those Note Namew whose reference IDs are found in the

/// current KeyComparison or Original Key

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnRevert\_Click(object sender, RoutedEventArgs e)

{

KeyWordsStaticMembers.ListAccess = false;

btnRevert.Visibility = Visibility.Visible;

lblKeyWordsAction.Content = "Revert to the previous Key Combination or Word";

btnRevert.Content = "Revert";

}// End btnRevert\_Click

#endregion Revert Button Clicked

}

}

KeyWordControlMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access/Static | Return | CalledBy | Purpose |
| **KeyWordControl.xaml.cs** | | | | |
| [rbtAdd\_Click](#KWCrbtAdd_ClickDef) |  |  | g.i.cs |  |
| [tbxInput\_KeyUp](#KWCtbxInput_KeyUpDef) |  |  | g.i.cs |  |
| [lbxKeyWords\_MouseLeftButtonUp](#KWClbxKeyWords_MouseLeftButtonUpDef) |  |  | g.i.cs |  |
| [tbxAllKeyWords\_TextChanged](#KWCtbxAllKeyWords_TextChangedDef) |  |  | g.i.cs |  |
| [btnRevert](#KWCbtnRevert_ClickDef)\_Click |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**Link\_Note**

[**LinkNote Methods**](#LinkNoteMethods)

using Microsoft.Win32;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using NewLSP.StaticHelperClasses;

using System.IO;

using NewLSP.DataModels;

using System;

using System.Windows.Input;

namespace NewLSP.UserControls

{

/// <summary>

/// Interaction logic for Link\_Note.xaml

/// </summary>

public partial class Link\_Note : UserControl

{

public Link\_Note()

{

InitializeComponent();

}

#region Menu Click Methods

#region Applications Menu

#region OpenFileDialog MenuItem

/// <summary>

/// This method gets a file path string by calling the

/// ReturnFilePath() private method which uses the OpenFileDialog to get the path

/// to a file that the user wants to save as a hyperlink for a DataNode

/// It then posts the hyperlink to tbxHyperlink.Txt

/// It then gets the file type and posts it to LinkNoteStaticMembers.FileType

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miOpenFileDialog\_Click(object sender, RoutedEventArgs e)

{

string Hyperlink = ReturnFilePath();

LinkNoteStaticMembers.Hyperlink = Hyperlink;

// determine if this link is a web address

if (Hyperlink.IndexOf("http") == 0)

{

//This is a web link

LinkNoteStaticMembers.FileType = "Web";

cmbxFileType.SelectedIndex = 2;

}

else

{

// This is some other type of file so get the file extenxtion

// get the position of the last '.'

int posLastDot = Hyperlink.LastIndexOf('.');

string FileExtension = Hyperlink.Substring(posLastDot + 1);

switch (FileExtension)

{

case "docx":

LinkNoteStaticMembers.FileType = "Word";

cmbxFileType.SelectedIndex = 0;

break;

case "txt":

LinkNoteStaticMembers.FileType = "Text";

cmbxFileType.SelectedIndex = 1;

break;

case "xlsx":

LinkNoteStaticMembers.FileType = "Excel";

cmbxFileType.SelectedIndex = 3;

break;

case "jpg":

LinkNoteStaticMembers.FileType = "Image";

cmbxFileType.SelectedIndex = 4;

break;

case "mp3":

LinkNoteStaticMembers.FileType = "Sound";

cmbxFileType.SelectedIndex = 5;

break;

case "mp4":

LinkNoteStaticMembers.FileType = "Video";

cmbxFileType.SelectedIndex = 6;

break;

}

tbxHyperlink.Text = Hyperlink;

}

}// End

#endregion OpenFileDialog MenuItem

#region Word MenuItem

/// <summary>

/// This Method opens MS Word for the user to get or open an

/// existing .docx file

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miWord\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(@"C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE");

}// miWord\_Click

#endregion Word MenuItem

#region Excel MenuItem

/// <summary>

/// This method opens MS Excel which allows the user

/// to open and existing or create a new excel file

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miExcel\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(@"C:\Program Files\Microsoft Office\root\Office16\EXCEL.EXE");

}// End miExcel\_Click

#endregion Excel MenuItem

#region Windows Media Player MenuItem

/// <summary>

/// This method calls the Windows Media Player so that the user can

/// play a mp4 file

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miWindowsMediaPlayer\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(@"C:\Program Files (x86)\Windows Media Player\wmplayer.exe");

}

#endregion Windows Media Player MenuItem

#region FireFox MenuItem

/// <summary>

/// This method opens the FireFox Browser for the user

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miFireFox\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(@"C:\Program Files\Mozilla Firefox\firefox.exe");

}

#endregion FireFox MenuItem

#region Notepad++ MenuItem

/// <summary>

/// This method opens Notepad++ for the user of

/// open or create a new text note

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void Notepad\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(@"C:\Program Files (x86)\Notepad++\notepad++.exe");

}

#endregion Notepad++ MenuItem

#region Meun Item MS Paint

/// <summary>

/// This method opens MP paint

/// so the user can display a .jpg file

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miPaint\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(@"C:\Windows\system32\mspaint.exe");

}

#endregion Meun Item MS Paint

#endregion Applications Menu

#region Files Menu

#region Save Hyperlink MenuItem

private void miSaveHyperlink\_Click(object sender, RoutedEventArgs e)

{

SaveHyperlink();

return;

}// End miSaveHyperlink\_Click

#endregion Save Hyperlink MenuItem

#region Save Note MenuItem

/// <summary>

/// This method Saves a Note when the program is in the Create mode.

/// It can be called by the Link\_Note.xaml's "miSaveNote" menu item

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miSaveNote\_Click(object sender, RoutedEventArgs e)

{

// Test to see if in Editing mode

if (LinkNoteStaticMembers.EditingBoolean)

{

// create a NoteReference string

string NoteReferenceStr = tbxLinkName.Text + "^" + tbxHyperlink.Text + "^" + tbxBookMark.Text + "^" + tbxAllKeyWords.Text;

// Call the static method to save both old and new note files

LinkNoteStaticMembers.SaveAndUpdateNoteReferenceAndKeywords(NoteReferenceStr);

return;

}

// Make sure that all required items for the NoteReferenceStr are present

if (KeyWordsStaticMembers.ListAccess)

{

// CREATE A NEW NOTE

// 1. Make sure that all required data fields are present

if ((tbxLinkName.Text == "") || (tbxHyperlink.Text == "") || (tbxAllKeyWords.Text == ""))

{

MessageBox.Show("You cannot save a Note unless there is a Name, a hyperlink and KeyWord(s)");

return;

}

// 2. Make sure the Key words are delimited witb ;s

if (!tbxAllKeyWords.Text.Contains(";"))

{

MessageBox.Show("Key words must be delimited wint ';'s ");

return;

}

// create a NoteReference string

string NoteReferenceStr = tbxLinkName.Text + "^" + tbxHyperlink.Text + "^" + tbxBookMark.Text + "^" + tbxAllKeyWords.Text

// Call the static method to save both old and new note files

LinkNoteStaticMembers.SaveAndUpdateNoteReferenceAndKeywords(NoteReferenceStr);

// Get the DataNodesNoteReferenceString and append it to the lbxOpenSelectedNote listBox

string DataNodesNoteReferenceString = LinkNoteStaticMembers.DataNodesNoteReferenceString;

string NoteName = StringHelper.ReturnItemAtPos(DataNodesNoteReferenceString, '^', 0);

string NoteCurrentNote26Name = StringHelper.ReturnItemAtPos(DataNodesNoteReferenceString, '^', 1);

// Delete the terminal \r\n"

NoteCurrentNote26Name = NoteCurrentNote26Name.Replace("\r\n", "");

////Test to see if there is a CommonStaticMembers.CurrentNote26Name and if not create one

// 1. Create a string of ' ' to make the NodeName length = 250

int NumberOfSpaces = 250 - tbxLinkName.Text.Length;

// 2. create a string with this many spacdes

string spacesStr = new String(' ', NumberOfSpaces);

// 3. Create the DataNodesReferenceFileLine

string displayString = NoteName + spacesStr + '^' + NoteCurrentNote26Name;

// If not editing add this note to the listbox

if(LinkNoteStaticMembers.EditingBoolean == false)

{

// Add dispalySgtring to the lbxOpenSelectedNote

lbxOpenSelectedNote.Items.Add(displayString);

}

//Clear Note entry fields

tbxLinkName.Text = "";

tbxHyperlink.Text = "";

tbxAllKeyWords.Text = "";

lbxKeyWords.Items.Clear();

tbxAllKeyWords.Text = "";

tbxInput.Text = "";

tbxBookMark.Text = "";

// Set Editing to false and uncheck the rbtEdit

LinkNoteStaticMembers.EditingBoolean = false;

rbtEdit.IsChecked = false;

}

}// End miSaveNote\_Click

#endregion Save Note MenuItem

#region Open Hyperlink MenuInte

private void miOpenHyperLink\_Click(object sender, RoutedEventArgs e)

{

string DataNodesHyperlinkPath = CommonStaticMembers.HomeFolderPath + "Hyperlinks\\" + SubjectStaticMembers.DataNode.ID.ToString() + ".txt";

// b. Test to see if a hyperlink file exists

if (File.Exists(DataNodesHyperlinkPath))

{

// Read the hyperlinks file into the LinkNoteStaticMembers.HyperlinkDictionary

// read all lines in the hyperlink file into a string []

string[] HyperlinksArray = File.ReadAllLines(DataNodesHyperlinkPath);

LinkNoteStaticMembers.HyperlinkDictionary.Clear();

lbxLinks.Items.Clear();

int HyperlinkCntr = 0;

foreach (string line in HyperlinksArray)

{

// Get the component parts

string[] componentItems = line.Split('^');

string Name = componentItems[0];

string Url = componentItems[1];

string FileType = componentItems[2];

string BookMark = componentItems[3];

LinkNoteModel.HyperlinkObject hyperlinkObject = new LinkNoteModel.HyperlinkObject();

hyperlinkObject.Name = Name;

hyperlinkObject.Url = Url;

hyperlinkObject.FileType = FileType;

hyperlinkObject.BookMark = BookMark;

LinkNoteStaticMembers.AddItemToHyperlinkDictionary(HyperlinkCntr, hyperlinkObject);

lbxLinks.Items.Add(Name);

HyperlinkCntr++;

}

}

}

#endregion Open Hyperlink MenuInte

#region Show MenuItem

/// <summary>

/// Called when the user clicks the "Show Notes" menu item in Files

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miShowNote\_Click(object sender, RoutedEventArgs e)

{

// Get the Node ID for the selected data node

int NodeID = SubjectStaticMembers.DataNode.ID;

//check to see that this data node has a notes file and if so read its values into

if (CommonStaticMembers.NodeHasNoteFile(NodeID))

{

ReadNotesIntoSelectNoteListBox();

}

}

#endregion Show Note MenuItem

#region Display Note Names MenuItem

/// <summary>

/// This method is called when the user clicks the

/// "Display Note Name MenuItem

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miDisplayNoteNames\_Click(object sender, RoutedEventArgs e)

{

ReadNotesIntoSelectNoteListBox();

PopulateNoteListBox();

}

#endregion Display Note Names MenuItem

#endregion Files Menu

#region ResetPage Menu

/// <summary>

/// The user clicks this menu item when they

/// want to clear all the fields, properties and controls so

/// that a new DataNode can be seleted

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void miResetPage\_Click(object sender, RoutedEventArgs e)

{

lbxLinks.Items.Clear();

tbxHyperlink.Text = "";

cmbxFileType.SelectedIndex = -1;

tbxLinkName.Text = "";

tbxAllKeyWords.Text = "";

lbxKeyWords.Items.Clear();

lbxOpenSelectedNote.Items.Clear();

tbxDisplayKeyWords.Text = "";

tbxBookMark.Text = "";

rbtAdd.IsChecked = false;

rbtSearch.IsChecked = false;

rbtEdit.IsChecked = false;

LinkNoteStaticMembers.EditingBoolean = false;

LinkNoteStaticMembers.HyperlinkDictionary.Clear();

}

#endregion ResetPage MenuItem

#region Create New Hyperlink Instructions Menu Item

private void miCreateNewHyperlinkInstructions\_Click(object sender, RoutedEventArgs e)

{

}

#endregion Create New Hyperlink Instructions Menu Item

#endregion Instructions Menu

#endregion Menu Click Methods

#region Private Methods

#region FileType combobox changed method

/// <summary>

/// When the File type is changed this method

/// converts the combobox item tag to a string

/// and sets the LinkNotesStaticMembers FileType to that string

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void cmbxFileType\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

ComboBoxItem thisItem = (ComboBoxItem)cmbxFileType.SelectedItem;

if (thisItem == null) return;

string thisItemsTag = thisItem.Tag.ToString();

LinkNoteStaticMembers.FileType = thisItemsTag;

}

#endregion FileType combobox changed method

#region Mouse up on List box of Links

/// <summary>

/// This method is called when the user clicks a hyperlink Name in

/// the lbxLinks List Box. It gets the HyperlinkObject

/// associated with this name and get the url and book mark,

/// which it copies to the clipboard and opens the ]hyperlink

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void lbxLinks\_PreviewMouseUp(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

int ItemClicked = lbxLinks.SelectedIndex;

LinkNoteModel.HyperlinkObject thisHyperlinkObject = LinkNoteStaticMembers.GetHyperlinkObject(ItemClicked);

string LinkName = thisHyperlinkObject.Name;

string Url = thisHyperlinkObject.Url;

string BookMark = thisHyperlinkObject.BookMark;

string Filetype = thisHyperlinkObject.FileType;

// set the FiltType combo box

switch (Filetype)

{

case "Word":

cmbxFileType.SelectedIndex = 0;

break;

case "Web":

cmbxFileType.SelectedIndex = 1;

break;

case "Excel":

cmbxFileType.SelectedIndex = 2;

break;

case "Image":

cmbxFileType.SelectedIndex = 3;

break;

case "Sound":

cmbxFileType.SelectedIndex = 4;

break;

case "Video":

cmbxFileType.SelectedIndex = 5;

break;

default: break;

}// End Switch

// set the hyperlink

tbxHyperlink.Text = Url;

// Set the BookMark

tbxBookMark.Text = BookMark;

if(tbxBookMark.Text != "")

{

Clipboard.SetText(tbxBookMark.Text);

}

//Open the hyperlink

OpenAnApp(Url);

}// End Mouse Up

#endregion Mouse up on List box of Links

#region private method open an executable or specific file type

private void OpenAnApp(string hyperlink)

{

try

{

System.Diagnostics.Process.Start(hyperlink);

}

catch (Exception e)

{

MessageBox.Show(e.Message+ " = "+ hyperlink);

}

}

#endregion private method open an executable or specific file type

#region private method SaveHyperlink

private void SaveHyperlink()

{

if (SubjectStaticMembers.DataNode == null)

{

MessageBox.Show("You cannot save this hyperlink because there is no designated DataNode");

return;

}

if (tbxLinkName.Text == "")

{

MessageBox.Show("You cannot save this hyperlink because there is no Name");

return;

}

// Get any bookmakr if present

string BookMark = tbxBookMark.Text;

// Get Name

string HyperlinkName = tbxLinkName.Text;

// Get Url

string Url = LinkNoteStaticMembers.Hyperlink;

// Get FileType

string FileType = LinkNoteStaticMembers.FileType;

string thisHyperlink = HyperlinkName + '^' + Url + '^' + FileType + '^' + BookMark;

// Add thisHyperLink to

LinkNoteStaticMembers.AddHyperlinkToList(thisHyperlink);

// Get the updated HyperlinkStringsList

List<string> currentHyperlinkStringsList = LinkNoteStaticMembers.HyperlinkStringsList;

// Create the filepath to the DataNodes HyperlinkFile

string DataNodesHyperlinkPath = CommonStaticMembers.HomeFolderPath + "Hyperlinks\\" + SubjectStaticMembers.DataNode.ID.ToString() + ".txt";

//Append this to the DataNode's Hyperlink file

File.WriteAllLines(DataNodesHyperlinkPath, currentHyperlinkStringsList);

// Add this line to the Dictionary

LinkNoteStaticMembers.HyperlinkDictionary.Clear();

//For each line in currentHyperlinkStringsList get the component parts and convert them into a Dictionary value

int HyperlinkCntr = 0;

foreach (string line in currentHyperlinkStringsList)

{

string[] HyperlinkLineArray = line.Split('^');

// create a new Hyperlink object

LinkNoteModel.HyperlinkObject thisHyperlinkObject = new LinkNoteModel.HyperlinkObject();

thisHyperlinkObject.Name = HyperlinkLineArray[0];

thisHyperlinkObject.Url = HyperlinkLineArray[1];

thisHyperlinkObject.FileType = HyperlinkLineArray[2];

thisHyperlinkObject.BookMark = HyperlinkLineArray[3];

//Add thisHyperlinkObject to the HyperlinkDictionary

LinkNoteStaticMembers.HyperlinkDictionary.Add(HyperlinkCntr, thisHyperlinkObject);

HyperlinkCntr++;

}

// Clear the lbxLinks listbox

lbxLinks.Items.Clear();

LinkNoteStaticMembers.BookMarks = new List<string>();

// add the revised HyperlinkToList to the ListBox

foreach (string line in LinkNoteStaticMembers.HyperlinkStringsList)

{

string[] linkSegments = line.Split('^');

lbxLinks.Items.Add(linkSegments[0]);

LinkNoteStaticMembers.BookMarks.Add(linkSegments[3]);

}

// Clear all the fields

cmbxFileType.SelectedIndex = -1;

tbxHyperlink.Text = "";

tbxLinkName.Text = "";

tbxBookMark.Text = "";

return;

}

#endregion private method SaveHyperlink

#region Return File Path of File Dialog OpenFile

private string ReturnFilePath()

{

OpenFileDialog od = new OpenFileDialog();

if (od.ShowDialog() == true) ;

{

return od.FileName;

}

}

#endregion Return File Path of File Dialog OpenFile

#region Open Selected Note Left Mouse click

/// <summary>

/// This method opens the selected Note when

/// the user left clicks on a selected note name

/// It uses the text string of the selected item to get the NoteReferenceFile's CurrentNote26Name

/// Which it then uses to open the NoteReferene file

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void lbxOpenSelectedNote\_PreviewMouseLeftButtonUp(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

string currnetItem = lbxOpenSelectedNote.SelectedItem.ToString();

//Get the CurrentNote26Name of a NoteReferenceFiles text file

string thisNoteReferenceName = StringHelper.ReturnItemAtPos(currnetItem, '^', 1);

// Update the CommonStaticMembers.CurrentNote26Name

CommonStaticMembers.CurrentNote26Name = thisNoteReferenceName;

//Use this name to create the path to the NoteReferenceFile

string NoteReferenceFilePath = CommonStaticMembers.NoteReferencesPath +"\\"+ thisNoteReferenceName + ".txt";

// Create a string[] to hold the data

string[] CurrentNoteReferenceFileDataArray = new string[4];

//Open the file and read in its text

if (File.Exists(NoteReferenceFilePath))

{

string thisNoteReferencesData = File.ReadAllText(NoteReferenceFilePath);

CurrentNoteReferenceFileDataArray = thisNoteReferencesData.Split('^');

}

//Set the Editing Boolean to true

LinkNoteStaticMembers.EditingBoolean = true;

// get the values in ListOfNoteNames, ListOfNoteHyperlinks, ListOfNoteBookMarks and ListOfNoteKeyWords

// associated with NoteSelectedIndex

string NoteName = CurrentNoteReferenceFileDataArray[0];

string NoteHyperlink = CurrentNoteReferenceFileDataArray[1];

string NoteBookmark = CurrentNoteReferenceFileDataArray[2];

string NoteKeyWords = CurrentNoteReferenceFileDataArray[3];

// Clear all Fields relative to a Note

tbxLinkName.Text = "";

tbxBookMark.Text = "";

tbxHyperlink.Text = "";

tbxAllKeyWords.Text = "";

lbxLinks.Items.Clear();

rbtEdit.IsChecked = true;

//Fill in the fields

tbxLinkName.Text = NoteName;

tbxBookMark.Text = NoteBookmark;

tbxHyperlink.Text = NoteHyperlink;

tbxAllKeyWords.Text = NoteKeyWords;

//If there is no book mark change it from

if(NoteBookmark != null)

{

Clipboard.SetText(NoteBookmark);

}

else

{

NoteBookmark = "";

}

//open the hyperlink

System.Diagnostics.Process.Start(NoteHyperlink);

}

#endregion Open Selected Note Left Mouse click

#region Show Key Words of Right Selected Note Name

/// <summary>

/// This method uses the selected Note name right clicked in the OpenSelectedNote list box

/// to create a global string DelStrOfKeyWordsAndComments

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void lbxOpenSelectedNote\_PreviewMouseRightButtonUp(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

//get the selected items text string

string SelectedItemsText = lbxOpenSelectedNote.SelectedItem.ToString();

// Get the Common references NoteReference file name from this string

string NotereferenceFileName = StringHelper.ReturnItemAtPos(SelectedItemsText, '^', 1);

// Use this value to set the CurrentNote26Name

CommonStaticMembers.CurrentNote26Name = NotereferenceFileName;

// Create a path to the NoteReferenceFile

string NoteReferenceFilePath = CommonStaticMembers.NoteReferencesPath + "\\" + NotereferenceFileName + ".txt";

// Read in the text in the NoteReferenceFile into a single ';' delimited string storred as DelStrOfKeyWordsAndComments

{Hidden here) string NoteReferenceFileText = File.ReadAllText(NoteReferenceFilePath);

// Read in the text in the NoteReferenceFile into a single ';' delimited string storred as DelStrOfKeyWordsAndComments

LinkNoteStaticMembers.DelStrOfKeyWordsAndComments = File.ReadAllText(NoteReferenceFilePath);

// Create a string variable to hold the keywords and comments that will be displayed in the

// tbxDisplayKeyWords TextBox

string KeyWordsString = "";

string tbxDisplayKeyWordsTextStr = "";

if (LinkNoteStaticMembers.ShowAllKeywords)

// The ShowAllKeywords radio button is clicked so show all keywords and comments

{

//Get the KeyWords section

string KeyWordsString = StringHelper.ReturnItemAtPos(LinkNoteStaticMembers.DelStrOfKeyWordsAndComments, '^', 3);

//Delete the last ';'

KeyWordsString = KeyWordsString.Substring(0, KeyWordsString.Length - 1);

// replace all ';' with \r\n

KeyWordsString = KeyWordsString.Replace(";", "\r\n");

// Display tbxDisplayKeyWords

tbxDisplayKeyWords.Text = KeyWordsString;

}

// the Show only selected Keywords radiobutton is checked so show only the selected KeyWord and its following '#' comments

{

KeyWordsString = StringHelper.ReturnItemAtPos(LinkNoteStaticMembers.DelStrOfKeyWordsAndComments, '^', 3);

// Get the SelectedKeyWord

string SearchKeyWord = LinkNoteStaticMembers.SearchKeyWord;

if(SearchKeyWord == null)

{

MessageBox.Show("You must have designated a single search key word.");

return;

}

// Find its position in the KeyWordsString

int posKeyWord = StringHelper.GetItemNumberOfThisSubstring(KeyWordsString, SearchKeyWord, ';');

if (posKeyWord != -1)

{

// this is the position of the selected key word

tbxDisplayKeyWordsTextStr = SearchKeyWord + "\r\n";

// Set the position of the next item in the KeyWordsString

int nextItemPos = posKeyWord + 1;

string nextItem = StringHelper.ReturnItemAtPos(KeyWordsString, ';', nextItemPos);

while (nextItem.Substring(0, 1) == "#")

{

tbxDisplayKeyWordsTextStr = tbxDisplayKeyWordsTextStr + nextItem + "\r\n";

nextItemPos++;

nextItem = StringHelper.ReturnItemAtPos(KeyWordsString, ';', nextItemPos);

if (nextItem == "") break;

}

// Display tbxDisplayKeyWords

tbxDisplayKeyWords.Text = tbxDisplayKeyWordsTextStr;

}

}

}// End lbxOpenSelectedNote\_PreviewMouseRightButtonUp(

#endregion Show Key Words of Right Selected Note Name

#region Private Method to Read noteReference file into lbxOpenSelectedNote

private void ReadNotesIntoSelectNoteListBox()

{

LinkNoteStaticMembers.ReadInNotesFile();

// Clear lbxOpenSelectedNote and tbxDisplayKeyWords

lbxOpenSelectedNote.Items.Clear();

// Fill the list box with ListOfNoteNameAndFileNames

foreach(string line in LinkNoteStaticMembers. ListBoxOfSelectedNotesList)

{

lbxOpenSelectedNote.Items.Add(line);

}

}

#endregion Private Method to Read noteReference file into lbxOpenSelectedNote

#region PopulateNoteListBox

/// <summary>

/// Called by miDisplayNoteNames\_Click(

/// </summary>

private void PopulateNoteListBox()

{

List<string> NoteNamesList = LinkNoteStaticMembers.ListOfNoteNames;

lbxOpenSelectedNote.Items.Clear();

foreach (string noteName in NoteNamesList)

{

lbxOpenSelectedNote.Items.Add(noteName);

}

}

#endregion PopulateNoteListBox

#endregion Private Methods

#region Link\_Note UserControl Methods

#region RadioButton Add

/// <summary>

/// Called when the user checks the Add Radio button

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbtAdd\_Click(object sender, RoutedEventArgs e)

{

btnRevert.Visibility = Visibility.Hidden;

KeyWordsStaticMembers.ListAccess = true;

lblKeyWordsAction.Content = "Add Key Words to a New Note Reference";

btnRevert.Content = "Revert";

}

#endregion RadioButton Add

#region Radio button Search

/// <summary>

/// Sets the ListAccess boolean to false because the program is

/// in the Search mode and new KeyWords are not allowed

/// It also sets the properts KeyWordSearch boolean to true

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbtSearch\_Click(object sender, RoutedEventArgs e)

{

tbxAllKeyWords.Text = "";

KeyWordsStaticMembers.ListAccess = false;

LinkNoteStaticMembers.KeyWordSearch = true;

}

#endregion Radio button Search

#region Input Textbox Key Up Procedure

/// <summary>

/// Take the characters typed into lbxKeyWords and show all terms in KeyWordList that start with these characters

/// If the User Hits the Enter Key

/// a. If there is data in the list of Keywords in the lbxKeyWords ListBox,

/// in either Add or Search mode then it returns the top item in the list

/// b. If there are no Items

/// 1) In search, Warn and return

/// 2) In Create, create a new KeyWord from the characters in the textbox

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param> private void tbxInput\_KeyUp(object sender, KeyEventArgs e)

{

// lbxKeyWords contains all of the keywords that start with the characters typed into the text box tbxInput

//Clear the current content of lbxKeyWords

lbxKeyWords.Items.Clear();

/\* Create a List of KeyWords whose leading chars match the substring in the input textbox

Cycle through the list of all key words in the KeyWords dictionary

and list, selecting all that begin with the characters in tbxInput

and place them in the Listbox of matching key words so that the user

can either select the 0th entry by hitting return or one down in

in the list by left clicking it

\*/

foreach (string line in KeyWordsStaticMembers.KeyWordList)

{

//In the current line begins with the characters in tbxInput add line to lbxKeyWords

if (line.IndexOf(tbxInput.Text) == 0)

{

lbxKeyWords.Items.Add(line);

}

}

//If the user hits the Enter key

if (e.Key == Key.Enter)

{

if (rbtEdit.IsChecked == true) KeyWordsStaticMembers.ListAccess = true;

// !! CHECH TO SEE IF ANY SEARCH WORDS ARE PRESENT IN THE CURRENT LIST OF KEY WORDS !! //

// If ListAccess is false, that means you are in the search mode

// If you are in the search mode and you type characters which do not occur at the start of

// any words in the listof KeyWords yoy get the following message

//ListAccess is the abilits to access the current list of keywords

if (!KeyWordsStaticMembers.ListAccess)

{

// If lbxKeyWords is empty in the Search statge send a message that you can only accepts existing keywords

if (lbxKeyWords.Items.Count == 0)

{

MessageBox.Show("When You are in the Search mode you can only search for existing Keywords");

// return to UI

return;

}

}

string KeyWord = "";

/\* Determine if there are Keywords showing in lbxKeyWords then select #0,

\* Set LinkNoteStaticMembers.SelectedKeyWord to this value,

\* Add the keyword to the tbxAllKeyWords.Text,

\* and return \*/

if (lbxKeyWords.Items.Count != 0)

//There is data in the lbxKeyWords ListBox

{

// Create a list from the KeyWords in lbxKeyWords so that the 0th item can be chosen

List<string> myCurrentKeyWordsList = new List<string>();

// Populate myCurrentKeyWordsList with the selected columns.

foreach (string thisKeyWordItem in lbxKeyWords.Items)

{

myCurrentKeyWordsList.Add(thisKeyWordItem);

}

//Set the Selected KeyWord to the 0th entry

KeyWord = myCurrentKeyWordsList[0];

//Set the LinkNoteStaticMembers.SelectedKeyWord string to this value

LinkNoteStaticMembers.SelectedKeyWord = KeyWord;

LinkNoteStaticMembers.SearchKeyWord = KeyWord;

if (LinkNoteStaticMembers.KeyWordSearch)

// add this Keyword to the list of selected keywords

tbxAllKeyWords.Text = tbxAllKeyWords.Text + KeyWord + ';';

//Clear tbxInput

tbxInput.Text = "";

//Clear lbxKeyWords

lbxKeyWords.Items.Clear();

//retrun to UI

return;

}// End there is data in the lbxKeyWords ListBox

// If the program reaches this point, there is no data in the lbxKeyWords ListBox

// Create a new KeyWord from the current text in tbxInput

KeyWord = tbxInput.Text;

// Remove any leading or trailing spaces

KeyWord = KeyWord.Trim();

// Add this KeyWord to tbxAllKeyWords with a ';' seperator

tbxAllKeyWords.Text = tbxAllKeyWords.Text + KeyWord + ';';

//!! DETECE COMMENTS AND DONT ADD THEM TO THE KEYWORD LIST 11 //

// If this is a generic comment(ie it begins with #) return without adding it to the KeyWordList

if (KeyWord.IndexOf("#") != -1)

{

tbxInput.Text = "";

return;

}

// !! UPDATE THE KEYWORD FILES !! //

// Update the active KeyWordList

KeyWordsStaticMembers.KeyWordList.Add(tbxInput.Text);

// Append this new Keyword to the Keyword txt Fild

KeyWordsStaticMembers.AppendNewKeyWord(KeyWord);

KeyWordsStaticMembers.AppendNewSortedKeyWord(KeyWord);

// Convert Keyword to Dictionary Item by replacing all spaces with '\_'

string thisKeyWord = tbxInput.Text;

string ConvertedThisKeyWord = thisKeyWord.Replace(' ', '\_');

//Add the new converted Key word to the dictionary with a value containing only the starting delimiter, ;

KeyWordsStaticMembers.KeyWordsDictionary.Add(ConvertedThisKeyWord, ";");

// Add the new converted keyword to the NotesDictionary.txt file

KeyWordsStaticMembers.AppendNewKeyWordDictionaryItemString(ConvertedThisKeyWord);

tbxInput.Text = "";

}

// Code to clear tbxInput if backspace results in empty text

if (e.Key == Key.Back)

{

if (tbxInput.Text == "") lbxKeyWords.Items.Clear();

}

}// End tbxInput\_KeyUp

#endregion Input Textbox Key Up Procedure

#region lbxKeyWords\_MouseLeftButtonUp

/// <summary>

/// This method is called when the user selects a KeyWord from the list of KeyWords in the lbxKeyWords ListBox

/// by left clicking it

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void lbxKeyWords\_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

string KeyWord = "";

try

{

KeyWord = lbxKeyWords.SelectedItem.ToString();

LinkNoteStaticMembers.SearchKeyWord = KeyWord;

if (!LinkNoteStaticMembers.ShowAllKeywords)

{

// The user is in the Search Mode

LinkNoteStaticMembers.SelectedKeyWord = KeyWord;

}

}

catch(Exception ex)

{

MessageBox.Show("You cannot have an empty keyword");

return;

}

tbxAllKeyWords.Text = tbxAllKeyWords.Text + KeyWord + ';';

tbxInput.Text = "";

lbxKeyWords.Items.Clear();

}

#endregion lbxKeyWords\_MouseLeftButtonUp

#region Revert Button Clicked

/// <summary>

/// This button is visible only when the user is in the

/// search mode. when clicked it removes the last

/// KeyWord from the KeyComparison and

/// displays it in the tbxAllKeyWords

/// It also changes the display in the lbxOpenSelectedNote

/// to only those Note Namew whose reference IDs are found in the

/// current KeyComparison or Original Key

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnRevert\_Click(object sender, RoutedEventArgs e)

{

KeyWordsStaticMembers.ListAccess = false;

btnRevert.Visibility = Visibility.Visible;

lblKeyWordsAction.Content = "Revert to the previous Key Combination or Word";

btnRevert.Content = "Revert";

}// End btnRevert\_Click

#endregion Revert Button Clicked

#region tbxAllKeyWords\_TextChanged

/// <summary>

/// This method is called whenever the text in the tbxAllKeyWords

/// is changed if the user in not in the Search mode then the program returns

/// a. When a keyword is entered create and call a method to ReturnNoteNameList, which will:

/// 1) Create a Dictionary<int, string> SelectedNoteReferenceValues, in which

/// a) The Key will be an integer from 0 to Number of referecnes containing that key-1 and

/// b) The Value will be the '^' delimited string of the selected NoteReferenceFile

/// 2) the ReturnNoteNameList method will then return a '~' delimited string of the

/// NoteName fields(position 0 in the '^' delimited string of the selected NoteReferenceFile)

/// of the selected Note References

/// 3) The Link\_note.xaml.cs file will display this list in the lbxOpenSelectedNote list box

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void tbxAllKeyWords\_TextChanged(object sender, TextChangedEventArgs e)

{

// Add a new keyword to this new note reference

if (rbtAdd.IsChecked == true)

{

// This is creating a new Note

string CurrentKeyWords = tbxAllKeyWords.Text;

CurrentKeyWords = CurrentKeyWords + tbxInput.Text + ';';

return;

}

//Check to see if in edit mode

if((rbtEdit.IsChecked == true))

{

if(tbxAllKeyWords.Text != "")

{

string CurrentKeyWords = tbxAllKeyWords.Text;

return;

}

else

{

return;

}

}

// Create a string variable from the text in tbxAllKeyWords

var SearchKeyWord = tbxAllKeyWords.Text;

if (SearchKeyWord == "") return;

// Delete the terminal ';' to create the SearchKeyWord

SearchKeyWord = SearchKeyWord.Substring(0, SearchKeyWord.Length - 1);

//Send this SearchKeyWord to KeyWordsStatidcMembers.ReturnNoteNameList to get a List<string> NoteNamesDispalyList

List<string> NoteNamesDispalyList = KeyWordsStaticMembers.ReturnNoteNameList(SearchKeyWord);

// Display this list in lbxOpenSelectedNote

// Clear any previous values

lbxOpenSelectedNote.Items.Clear();

// Cycle through NoteNamesDispalyList adding them to the list box

foreach(string NoteName in NoteNamesDispalyList)

{

lbxOpenSelectedNote.Items.Add(NoteName);

}

}// End tbxAllKeyWords\_TextChanged

#endregion tbxAllKeyWords\_TextChanged

#endregion KeyWord Controls

private void rbtEdit\_Click(object sender, RoutedEventArgs e)

{

LinkNoteStaticMembers.EditingBoolean = true;

KeyWordsStaticMembers.ListAccess = true;

}

private void tbxHyperlink\_TextChanged(object sender, TextChangedEventArgs e)

{

LinkNoteStaticMembers.Hyperlink = tbxHyperlink.Text;

}

private void tbxBookMark\_TextChanged(object sender, TextChangedEventArgs e)

{

LinkNoteStaticMembers.BookMarks.Add(tbxBookMark.Text);

}

private void tbxLinkName\_TextChanged(object sender, TextChangedEventArgs e)

{

LinkNoteStaticMembers.HyperlinkName = tbxLinkName.Text;

}

/// <summary>

/// This Method is called when the user clicks the open All KeyWords radio button.

/// It calls a local private method

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbtKeyWordsAll\_Checked(object sender, RoutedEventArgs e)

{

LinkNoteStaticMembers.ShowAllKeywords = true;

}

private void Selected\_Checked(object sender, RoutedEventArgs e)

{

LinkNoteStaticMembers.ShowAllKeywords = false;

}

}// End class

}// End Name space

**LinkNoteMethods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **Link\_Note.xaml.cs** | | | | |
| [miOpenFileDialog\_Click](#miOpenFileDialog_Click) | private | void | miOpenFileDialog | This method gets a file path string by calling the ReturnFilePath() private method which uses the OpenFileDialog to get the path to a file that the user wants to save as a hyperlink for a DataNode It then posts the hyperlink to bxHyperlink.Txt It then gets the file type and posts it to LinkNoteStaticMembers.FileType |
| [miWord\_Click](#miWord_Click) | private | void | miWord | NOTE: Comment Uses the System.Diagnostics.Process.Start method to call WINWoRD.EXE |
| [miExcel\_Click](#miExcel_Click) | private | void | mExcel | NOTE: Comment Uses the System.Diagnostics.Process.Start method to call EXCEL.EXE |
| [miWindowsMediaPlayer\_Click](#miWindowsMediaPlayer_Click) | private | void | miWindowsMediaPlayer | NOTE: Comment Uses the System.Diagnostics.Process.Start method to call wmplayer.exe |
| [Notepad\_Click](#Notepad_Click) | private | void | Notepad | NOTE: Comment Uses the System.Diagnostics.Process.Start method to call notepad++.exe |
| [miFireFox\_Click](#miFireFox_Click) | private | void | miFireFox | NOTE: Comment Uses the System.Diagnostics.Process.Start method to call firefox.exe |
| [miPaint\_Click](#miPaint_Click) | private | void | miPaint | NOTE: Comment Uses the System.Diagnostics.Process.Start method to call mspaint.ex |
| [miSaveHyperlink\_Click](#miSaveHyperlink_Click) | private | void | miSaveHyperlink | NOTE: Comment Calls the private SaveHyperlink(); method |
| [miSaveNote\_Click](#miSaveNote_Click) | private | void | miSaveNote | This method Saves a Note when the program is in the Create mode. It can be called by the Link\_Note.xaml's "miSaveNote" menu item |
| [miOpenHyperLink\_Click](#miOpenHyperLink_Click) | private | void | miOpenHyperLink | NOTE: Comment Tests to see if a hyperlink file exists for the DataNode and opens it |
| [miShowNote\_Click](#miShowNote_Click) | private | void | miShowNote | Called when the user clicks the "Show Notes" menu item in Files |
| [miDisplayNoteNames\_Click](#miDisplayNoteNames_Click) | private | void | miDisplayNoteNames | This method is called when the user clicks the "Display Note Name MenuItem |
| [miResetPage\_Click](#miResetPage_Click) | private | void | miResetPage | The user clicks this menu item when they want to clear all the fields, properties and controls so that a new DataNode can be seleted |
| [miCreateNewHyperlinkInstructions\_Click](#miCreateNewHyperlinkInstructions_Click) | private | void | miCreateNewHyperlinkInstructions | NOTE:!!! Not Defined Eliminate |
| [cmbxFileType\_SelectionChanged](#cmbxFileType_SelectionChanged) | private | void | cmbxFileType | When the File type is changed this method converts the combobox item tag to a string and sets the LinkNotesStaticMembers FileType to that string |
| [tbxLinkName\_TextChanged](#tbxLinkName_TextChanged) | private | void | tbxLinkName | NOTE: Comment Assigns the tbxLinkName.Text to LinkNoteStaticMembers.HyperlinkName |
| [tbxBookMark\_TextChanged](#tbxBookMark_TextChanged) | private | void | tbxBookMark | NOTE: Comment Adds the tbxBookMark.Text to LinkNoteStaticMembers.BookMarks (??What) |
| [tbxHyperlink\_TextChanged](#tbxHyperlink_TextChanged) | private | void | tbxHyperlink | NOTE: Comment Adds the tbxHyperlink.Text to LinkNoteStaticMembers.Hyperlink |
| [rbtAdd\_Click](#rbtAdd_Click) | private | void | rbtAdd | NOTE: Comment Hides the btnRevert button; sets the KeyWordsStaticMembers.ListAccess to true; and shows the message "Add Key Words to a New Note Reference" in the lblKeyWordsAction and sets the content of the btnRevert button to "Revert"; |
| [rbtSearch\_Click](#rbtSearch_Click) | private | void | rbtSearch | Sets the ListAccess boolean to false because the program is in the Search mode and new KeyWords are not allowed |
| [rbtEdit\_Click](#rbtEdit_Click) | private | void | rbtEdit | NOTE: Comment Sets the LinkNoteStaticMembers.EditingBoolean to true; and the KeyWordsStaticMembers.ListAccess to true; |
| [tbxInput\_KeyUp](#tbxInput_KeyUp) | private | void | tbxInput | If the User Hits the Enter Key a. If there are itemn in the list, in either mode return the top item in the list b. If there are no Items 1) In search, Warn and return 2) In Create, create a new KeyWord from the characters in the textbox |
| [lbxKeyWords\_MouseLeftButtonUp](#lbxKeyWords_MouseLeftButtonUp) | private | void | lbxKeyWords | NOTE: Comment |
| [btnRevert\_Click](#btnRevert_Click) | private | void | btnRevert | This button is visible only when the user is in the search mode. when clicked it removes the last KeyWord from the KeyComparison and displays it in the tbxAllKeyWords It also changes the display in the lbxOpenSelectedNote to only those Note Namew whose reference IDs are found in the current KeyComparison or Original Key  NOTE:!! This probably needs to be removed |
| [tbxAllKeyWords\_TextChanged](#tbxAllKeyWords_TextChanged) | private | void | tbxAllKeyWords | This method is called whenever the text in the tbxAllKeyWords is changed if the user in not in the Search mode then the program returns a. When a keyword is entered create and call a method to ReturnNoteNameList, which will: 1) Create a Dictionary<int, string> SelectedNoteReferenceValues, in which a) The Key will be an integer from 0 to Number of referecnes containing that key-1 and b) The Value will be the '^' delimited string of the selected NoteReferenceFile 2) the ReturnNoteNameList method will then return a '~' delimited string of the NoteName fields(position 0 in the '^' delimited string of the selected NoteReferenceFile) of the selected Note References 3) The Link\_note.xaml.cs file will display this list in the lbxOpenSelectedNote list box |
| [lbxLinks\_PreviewMouseUp](#lbxLinks_PreviewMouseUp) | private | void | lbxLinks | the lbxLinks List Box. It gets the HyperlinkObject associated with this name and get the url and book mark, which it copies to the clipboard and opens the ]hyperlink |
| [lbxOpenSelectedNote\_PreviewMouseLeftButtonUp](#lbxOpenSelectedNoteLeftMouseButtonClick) | private | void | lbxOpenSelectedNote | This method opens the selected Note when the user left clicks on a selected note name It uses the text string of the selected item to get the NoteReferenceFile's CurrentNote26Name Which it then uses to open the NoteReferene file |
| [lbxOpenSelectedNote\_PreviewMouseRightButtonUp](#lbxOpenSelectedNoteRightButtonClick) | private | void | lbxOpenSelectedNote | This method displays the Key words associated with the Note name that the User clicks with the Right Mouse button |
| [OpenAnApp](#OpenAnApp)(string hyperlink) | private | void | lbxLinks\_PreviewMouseU | NOTE: Comment Calls System.Diagnostics.Process.Start the open the file referenced by hyperkink |
| [SaveHyperlink](#SaveHyperlink) | private | void |  | NOTE: Comment After making sure there is a DataNode and a name for the new hyperlink, it creates a string composed of HyperlinkName + '^' + Url + '^' + FileType + '^' + BookMark and sends it to LinkNoteStaticMembers.AddHyperlinkToList |
| [ReturnFilePath](#ReturnFilePath) | private | void | miOpenFileDialog\_Click | Return File Path of File Dialog OpenFile |
| [ReadNotesIntoSelectNoteListBox](#ReadNotesIntoSelectNoteListBox) | private | void | miShowNote\_Click  miDisplayNoteNames\_Clic | NOTE: Comment |
| [PopulateNoteListBox](#PopulateNoteListBox) | private | void | miDisplayNoteNames\_Click | NOTE: Comment |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**SubjectTree.xaml.cs**

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using NewLSP.StaticHelperClasses;

using NewLSP.DataModels;

using System.Collections.Generic;

using System;

using System.IO;

namespace NewLSP.UserControls

{

/// <summary>

/// Interaction logic for SubjectTree.xaml

/// </summary>

public partial class SubjectTree : UserControl

{

public SubjectTree()

{

InitializeComponent();

}

#region Display the Selected Subject's Root and Children

/// <summary>

/// Displays every line in SubjectStaticMembers.DisplayList

/// in the lbSubjects ListView

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnShowDisplayLisgt\_Click(object sender, RoutedEventArgs e)

{

if (SubjectStaticMembers.DisplayList.Count != 0)

{

foreach (string line in SubjectStaticMembers.DisplayList)

{

lvSubjects.Items.Add(line);

}

}

}// EndbtnShowDisplayLisgt\_Click

#endregion Display the Selected Subject's Root and Children

#region Mouse Rigth Button Up to Select new parent

/// <summary>

/// This method is part of the move node procedure

/// After the user has selected a node to move, he

/// is instructed to selecte the new parent by

/// Right Clicking the desired parent and that

/// calls this method

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void lvSubjects\_MouseRightButtonUp(object sender, MouseButtonEventArgs e)

{

if (lvSubjects.SelectedIndex >= 0)

{

// Get the NodeLevelName for this node from the SubjectNodesLevelNameList

string NodeLevelName = SubjectStaticMembers.SubjectNodesLevelNameList[lvSubjects.SelectedIndex];

// Use the NodeLevelName to get the correct node fromthe dictionary of SubjectNodeDictionary

NewParentNode = SubjectStaticMembers.SubjectNodeDictionary[NodeLevelName];

}

MoveNode();

SubjectStaticMembers.SaveFiles();

}// End lvSubjects\_MouseRightButtonUp

#endregion Mouse Rigth Button Up

#region MoveNode()

private void MoveNode()

{

//Increment the new parent's number of children

NewParentNode.NOC ++;

//determine if the leading character is correct

if(NewParentNode.CI != "+ ")

{

NewParentNode.CI = "+ ";

}

//Save the New Parent to the dictionary

SubjectStaticMembers.SubjectNodeDictionary[NewParentNode.NodeLevelName] = NewParentNode;

//Get the Child's NodeLevelPosition

string ChildsNodeLevelPosition = SubjectStaticMembers.GetNodeLevelPosition(NewParentNode.NOC);

// create new NLN for the moved node

string NewNLN = NewParentNode.NodeLevelName + ChildsNodeLevelPosition;

//get Old NodeLevelName

string OldNLN = SubjectStaticMembers.OldNLN;

// Call static to cycly through dictionary replacing OldNLN with NewNLN

SubjectStaticMembers.ChangeMovedNodesNLN(OldNLN, NewNLN);

// Change the display to show the moved node added to the new parent

SubjectStaticMembers.DisplayParentsAndChildren(NewParentNode.NodeLevelName);

}

#endregion MoveNode()

//=======================================================//

#region Properties

private static SubjectNodes SelectedNode;

private static SubjectNodes ParentNode;

private static SubjectNodes OldParentNode;

private static SubjectNodes NewParentNode;

private static SubjectNodes NewChildNode;

private static SubjectNodes NodeToMove;

#endregion Properties

//=======================================================//

#region Chose Selected Node in ListView

/// <summary>

/// Uses the MouseLeftButtonUp event to designate the SelectedNode

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void lvSubjects\_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

// Selected Index = is the 0 based line number of what ever is currently being displayed on the 'tblkSubjectName' textblock

if (lvSubjects.SelectedIndex >= 0)

{

// Get the NodeLevelName for this node from the SubjectNodesLevelNameList

// There are currently 6 possible levels ranging from '\*', '1'...'5' where '\*' is the root level

string NodeLevelName = SubjectStaticMembers.SubjectNodesLevelNameList[lvSubjects.SelectedIndex];

// Use the NodeLevelName to get the correct node fromthe dictionary of SubjectNodeDictionary

// The Key to the dictionary is

SelectedNode = SubjectStaticMembers.SubjectNodeDictionary[NodeLevelName];

SubjectStaticMembers.DataNode = SelectedNode;

//Determine if this node has a QA file

int NodeID = SelectedNode.ID;

if (SubjectStaticMembers.NodeHasQAFile(NodeID))

{

spQA.Visibility = Visibility.Visible;

}

else

{

spQA.Visibility = Visibility.Hidden;

}

//Determine if this node has a Hyperlink file

if (SubjectStaticMembers.NodeHasHyperlinksFile(NodeID))

{

spHyperlink.Visibility = Visibility.Visible;

}

else

{

spHyperlink.Visibility = Visibility.Hidden;

}

// Determin if the node has a Notes file

if (CommonStaticMembers.NodeHasNoteFile(NodeID))

{

spNote.Visibility = Visibility.Visible;

// Fill the DictionaryOfNoteNamesIDs

//CommonStaticMembers.FillDictionaryOfNoteNamesIDs();

}

else

{

spNote.Visibility = Visibility.Hidden;

}

}

}// End lvSubjects\_PreviewMouseLeftButtonU

#endregion Chose Selected Node in ListView

//=======================================================//

#region Radio Button Methods

#region Radio button New Chile

/// <summary>

/// This method is called whenever the user clicks the create a new child node radio button

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbNewChild\_Checked(object sender, RoutedEventArgs e)

{

var rbNewChild = sender as RadioButton;

if (tbxNodeName.Text == "")

{

MessageBox.Show("You must Enter text into the Enter Node Text TextBox and select a Parent Node");

rbNewChild.IsChecked = false;

return;

}

if (SelectedNode == null)

{

MessageBox.Show("You Must select a Parent Node before Clicking Create a New Child Node");

rbNewChild.IsChecked = false;

return;

}

if (SelectedNode.CI == "T ")

{

MessageBox.Show("You Cannot add a child to a Terminal node");

rbNewChild.IsChecked = false;

return;

}

// Get ItemIndex and convert it into an int

int CurrentItemCountInt = System.Int32.Parse(SubjectStaticMembers.ItemCount)+1;

string ItemCount = CurrentItemCountInt.ToString();

SubjectStaticMembers.ItemCount = ItemCount;

// Create a new node

CreateNewChildSubjectNode(CurrentItemCountInt);

SubjectStaticMembers.SaveFiles();

}

#endregion Radio button New Chile

#region #region Radio button Change Title Text of selected node (rbText\_Checked)

/// <summary>

/// The purpose of this method is to

/// change the TitleText of the selected node

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbText\_Checked(object sender, RoutedEventArgs e)

{

if (tbxNodeName.Text == "")

{

MessageBox.Show("You must Enter text into the Enter Node Text TextBox and select the node to Change");

rbText.IsChecked = false;

return;

}

if (SelectedNode == null)

{

MessageBox.Show("You Must select a Node before Clicking Change Title Text");

rbText.IsChecked = false;

return;

}

string NewTitleText = tbxNodeName.Text;

SelectedNode.TitleText = NewTitleText;

string ThisNodeLevelName = SelectedNode.NodeLevelName;

SubjectStaticMembers.SubjectNodeDictionary[ThisNodeLevelName] = SelectedNode;

List<string> NewDisplayList = SubjectStaticMembers.DisplayParentsAndChildren(ThisNodeLevelName);

lvSubjects.Items.Clear();

foreach (string DisplayLine in NewDisplayList)

{

lvSubjects.Items.Add(DisplayLine);

}

SubjectStaticMembers.SaveFiles();

}// End rbText\_Checked

#endregion (rbText\_Checked)

#region Radio button make node terminal

private void rbTerminal\_Checked(object sender, RoutedEventArgs e)

{

SelectedNode.CI = "T ";

SubjectStaticMembers.SubjectNodeDictionary[SelectedNode.NodeLevelName] = SelectedNode;

string ParentNodesNodeLevelName = SelectedNode.NodeLevelName.Substring(0, SelectedNode.NodeLevelName.Length - 1);

ParentNode = SubjectStaticMembers.SubjectNodeDictionary[ParentNodesNodeLevelName];

SubjectStaticMembers.DisplayParentsAndChildren(ParentNodesNodeLevelName);

SubjectStaticMembers.SaveFiles();

}

#endregion Radio button make node terminal

#region Radio button Delete

private void rbDelete\_Checked(object sender, RoutedEventArgs e)

{

// Test to see if the node has data or children before deleting it

if (SelectedNode.HasData || SelectedNode.NOC > 0)

{

MessageBox.Show("You cannot delete a node that has children or data, you can only move it");

return;

}

// Remove the node from the dictionary

SubjectStaticMembers.RemoveNodeFromDictionary(SelectedNode.NodeLevelName);

// Designate the parent node

string ParentNodesNLN = SelectedNode.NodeLevelName.Substring(0, SelectedNode.NodeLevelName.Length - 1);

ParentNode = SubjectStaticMembers.ReturnNodeAtPos(ParentNodesNLN);

// Adjust the parent's number of children

ParentNode.NOC = ParentNode.NOC - 1;

// adjust the Parent's nodes child indicaator if necessarr

if (ParentNode.NOC == 0)

{

ParentNode.CI = "- ";

}

// store updated parent node in the dictionary

SubjectStaticMembers.ReplaceNode(ParentNode.NodeLevelName, ParentNode);

// Change the display list to the Parents Children

List<string> NewDisplayList = SubjectStaticMembers.DisplayParentsAndChildren(ParentNode.NodeLevelName);

lvSubjects.Items.Clear();

foreach (string DisplayLine in NewDisplayList)

{

lvSubjects.Items.Add(DisplayLine);

}

SubjectStaticMembers.SaveFiles();

}// End rbDelete\_Checked

#endregion Radio button Delete

#region Radio Button Show node, ancestory and children

/// When a node's children are not showing clicking

/// will show this node, its parents and it children

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbExpandCollapse\_Checked(object sender, RoutedEventArgs e)

{

if (SelectedNode == null)

{

MessageBox.Show("You must select a subject node before proceeding!");

rbExpandCollapse.IsChecked = false;

return;

}

string SelectedNodeNameLevelName = SelectedNode.NodeLevelName;

SubjectStaticMembers.DisplayParentsAndChildren(SelectedNodeNameLevelName);

lvSubjects.Items.Clear();

foreach (string item in SubjectStaticMembers.DisplayList)

{

lvSubjects.Items.Add(item);

}

rbExpandCollapse.IsChecked = false;

SelectedNode = null;

}

#endregion Radio Button Show node, ancestory and children

#region Radio Button create DataNode

/// <summary>

/// Called when the user clicks the

/// This is the DataNode Radio button

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbDataNode\_Click(object sender, RoutedEventArgs e)

{

// Set the SubjectStaticMembers DataNode

SubjectStaticMembers.DataNode = SelectedNode;

int SubjectNodeId = SelectedNode.ID;

//Set the DataNodeNoteReferenceFilePath()

CommonStaticMembers.DataNodesNoteReferencesFilesPath = CommonStaticMembers.HomeFolderPath + "DataNodesNoteReferencesFiles\\" + SubjectNodeId.ToString() + ".txt";

//blank the dataNodesQAFilePath

SubjectStaticMembers.DataNodesQAFilePath = "";

//sets the int CurrentQANumberInt and QAStaticMembers.CurrentQANumberInt to 0

QAStaticMembers.CurrentQANumberInt = 0;

// Clear the hyperlink dictionary

LinkNoteStaticMembers.HyperlinkDictionary.Clear();

// clear the QAStaticMembers.QADictionary

QAStaticMembers.QADictionary.Clear();

// Set the Path to the DataNode's QAfile

SubjectStaticMembers.SetDataNodesQAFilePath();

// Notify the user that the Data node has been set and uncheck it

MessageBox.Show("The Data Node has been set");

rbDataNode.IsChecked = false;

//Set the CurrentNoteIDInt

CommonStaticMembers.CurrentNoteIDInt = SubjectStaticMembers.DataNode.ID;

}// End rbDataNode\_Click

#endregion Radio Button create DataNode

#region RadioButton Move Node

private void rbMoveNode\_Checked(object sender, RoutedEventArgs e)

{

// Designate the Node to move

NodeToMove = SelectedNode;

SelectedNode = NodeToMove;

//Set the node level name of the node to be moved

SubjectStaticMembers.OldNLN = SelectedNode.NodeLevelName;

// Get the NLN of this node's parent

string ParentNLN = SelectedNode.NodeLevelName.Substring(0, SelectedNode.NodeLevelName.Length - 1);

OldParentNode = SubjectStaticMembers.SubjectNodeDictionary[ParentNLN];

MessageBox.Show("Select the New Parent Node by Clicking the Right Mouse Button");

}// Ed rbMoveNode\_Checked

#endregion RadioButton Move Node

#endregion Radio Button Methods

//=======================================================//

#region Private local methods

#region Create a new child node (CreateNewChildSubjectNode)

/// <summary>

/// This method receives the curent number of items creates

/// (currentItemCount) and creates a new node wiht this as the ID

/// It gets the parent from the list view index of the selected node

/// It Adds the node to the dictionary, adjusts the Parent node

/// and resets the display to reflect the Parents children

/// </summary>

/// <param name="currentItemCount"></param>

private void CreateNewChildSubjectNode(int currentItemCountInt)

{

// Instantiate a SubjectNode with the currentItemCount

NewChildNode = new SubjectNodes(currentItemCountInt);

// Get the Parent Node

ParentNode = GetParentNode();

//Get the Parent's Number of Children to calcuate the child's NodeLevelName

int ParentsNumchildren = ParentNode.NOC;

//Get the Child's NodeLevelPosition

string ChildsNodeLevelPosition = SubjectStaticMembers.GetNodeLevelPosition(ParentsNumchildren);

// Set the NewChildNodes NodeLevel

NewChildNode.NodeLevelName = ParentNode.NodeLevelName + ChildsNodeLevelPosition;

// Set the Child node's leading char sgtring

NewChildNode.LeadingChars = SubjectStaticMembers.GetLeadingChars(NewChildNode.NodeLevelName);

// Set the Child indicator to no children

NewChildNode.CI = "- ";

//Set the TitleText to the text in the node name textbox

NewChildNode.TitleText = tbxNodeName.Text;

// set the has associated data files to false

NewChildNode.HasData = false;

// set the child node's number of children to 0

NewChildNode.NOC = 0;

// Add this child to the dictionary

SubjectStaticMembers.AddNodeToDictionary(NewChildNode);

// Increment the parents NOC and CI

ParentNode.CI = "+ ";

ParentNode.NOC++;

SubjectStaticMembers.DisplayParentsAndChildren(ParentNode.NodeLevelName);

lvSubjects.Items.Clear();

foreach (string item in SubjectStaticMembers.DisplayList)

{

lvSubjects.Items.Add(item);

}

// Clear the NodeName textbox

tbxNodeName.Text = "";

//UnCheck the new child radio button

rbNewChild.IsChecked = false;

//store the updated parents node in the dictionary

SubjectStaticMembers.SubjectNodeDictionary[ParentNode.NodeLevelName] = ParentNode;

//Increment and save the ItemsIndex

SelectedNode = null;

}// End CreateNewChildSubjectNode

#endregion (CreateNewChildSubjectNode)

#region Get the Parent (ie the selected node) of a new child node (GetParentNode)

/// <summary>

/// Use the Selected Index to get the parent of a new child node

/// </summary>

/// <returns></returns>

private SubjectNodes GetParentNode()

{

int SelectedIndex = lvSubjects.SelectedIndex;

string SelectedNodesLevelName = SubjectStaticMembers.SubjectNodesLevelNameList[SelectedIndex];

// Create ParentNode

ParentNode = SubjectStaticMembers.SubjectNodeDictionary[SelectedNodesLevelName];

return ParentNode;

}// End GetParentNode

#endregion GetParentNode

#endregion Private local methods

}// End Class

}//End Namespace

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access/Static | Return | CalledBy | Purpose |
| **SubjectTree.xaml.cs** | | | | |
|  |  |  |  |  |
| lvSubjects\_[PreviewMouseLeftButtonUp](#PreviewMouseLeftButtonUp) | private | void | lvSubjects | Uses the MouseLeftButtonUp event to designate the SelectedNode |
| lvSubjects\_[MouseRightButtonUp](#MouseRightButtonUp) | private | void | lvSubjects | This method is part of the move node procedure After the user has selected a node to move, he is instructed to selecte the new parent by Right Clicking the desired parent and that calls this method |
| [btnShowDisplayLisgt](#btnShowDisplayLisgt_Click)\_Click | private | void | btnShowDisplayLisgt | Displays every line in SubjectStaticMembers.DisplayList in the lbSubjects ListView |
| [rbNewChild](#rbNewChild_Checked)\_Checked | private | void | rbNewChild | This method is called whenever the user clicks the create a new child node radio button |
| [rbText\_Checked](#rbText_Checked) | private | void | rbText | The purpose of this method is to change the TitleText of the selected node |
| [rbTerminal](#rbTerminal_Checked)\_Checked | private | void | rbTerminal | NOTE: Comment Marks the node as Terminal so no children can be added to it |
| <#rbDelete_Checked> | private | void | rbDelete | NOTE: Comment If the node has no children it calls SubjectStaticMembers.RemoveNodeFromDictionary to remove the node |
| [rbMoveNode](#rbMoveNode_Checked)\_Checked | private | void | rbMoveNode | Sets the local variable NodeToMove to the SelectedNode |
| [rbExpandCollapse](#rbExpandCollapse_Checked)\_Checked | private | void | rbExpandCollapse | When a node's children are not showing clicking will show this node, its parents and it children |
| [rbDataNode](#rbDataNode_Click)\_Click | private | void | rbDataNode | Called when the user clicks the This is the DataNode Radio button |
| [MoveNode](#MoveNode) | private | void | lvSubjects\_MouseRightButtonUp | NOTE: Comment Moves the selected node to a new parent and adjusts the dictionary |
| [CreateNewChildSubjectNode](#CreateNewChildSubjectNode) | private | void | rbNewChild\_Checked | This method receives the curent number of items creates (currentItemCount) and creates a new node wiht this as the ID It gets the parent from the list view index of the selected node It Adds the node to the dictionary, adjusts the Parent node and resets the display to reflect the Parents children |
| [GetParentNode](#GetParentNodeDef) | private | SubjectNodes | CreateNewChildSubjectNode | Use the Selected Index to get the parent of a new child node |
|  |  |  |  |  |
|  |  |  |  |  |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**TestReview.xaml.cs**

using System.IO;

using System.Windows;

using System.Windows.Controls;

using NewLSP.StaticHelperClasses;

using System;

using NewLSP.DataModels;

namespace NewLSP.UserControls

{

/// <summary>

/// Interaction logic for TestReview.xaml

/// </summary>

public partial class TestReview : UserControl

{

public TestReview()

{

InitializeComponent();

}

#region Private Fields

#region Boolean IsTest

private static bool IsTest;

#endregion Boolean IsTest

#endregion Private Fields

#region Menu Items

#region Files Menu

#region SaveFile

private void SaveFile\_Click(object sender, RoutedEventArgs e)

{

//Check to see if the QAResults directory exists and if not create it

string ResultsDirectoryPath = CommonStaticMembers.SubjectFolderPath+"QAResults";

if (!Directory.Exists(ResultsDirectoryPath))

{

Directory.CreateDirectory(ResultsDirectoryPath);

}

//check to see if the NodeID result .txt file exists and if not create

//Get DataNode.ID.ToString()

string DataNodeIDStr = SubjectStaticMembers.DataNode.ID.ToString();

//Create File path

string ResultsFilePath = ResultsDirectoryPath + "\\" + DataNodeIDStr + ".txt";

//Get the current date time

DateTime currentDate = DateTime.Now;

string CurrentDateTimeString = currentDate.ToString("yyy MM dd HHmm");

// compose the output line

//string FormattedDateStr = String.Format("YYYYMMDDHHmm", currentDate);

double total =QAStaticMembers.QADictionary.Count;

double wrong = TestReviewStaticMembers.NumberOfWrongAnswers;

double PercentCorrect = ((total - wrong) / total)\*100;

string PercentCorrectStr = string.Format("{0:N2}% correct = ", PercentCorrect);

// create the output line

string OutputStr = CurrentDateTimeString + "| " + PercentCorrectStr + " " + TestReviewStaticMembers.DelimitedWrongAnswersStr;

// Append this line to the existing file

File.AppendAllText(ResultsFilePath, OutputStr + Environment.NewLine);

// Check to see if the dictionary has been changed and if so save it

if (QAStaticMembers.DictionaryChanged == true)

{

QAStaticMembers.SaveQADictionary();

}

// Send a message telling the use that it has been saved

MessageBox.Show("The Results are saved. You can exit now.");

}

#endregion SaveFile

#endregion Files Menu

#region Test/Review Menu

#region Test

private void Test\_Click(object sender, RoutedEventArgs e)

{

IsTest = true;

TestOrReview();

return;

}

#endregion Test

#region Review

private void Review\_Click(object sender, RoutedEventArgs e)

{

IsTest = false;

TestOrReview();

return;

}

#endregion Review

#endregion Test/Review Menu

#region Subject Order Menu

#region Menu Item Seriatim

private void miSeriatim\_Click(object sender, RoutedEventArgs e)

{

TestReviewStaticMembers.QuestionsSeriatim = true;

} #endregion Menu Item Seriatim

#region MenuItem Random

private void miRandom\_Click(object sender, RoutedEventArgs e)

{

}

#endregion MenuItem Random

#endregion Subject Order Menu

#region Instructions Menu

#region UseForm

private void UseForm\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Use Form Menu Clicked");

}

#endregion UseForm

#endregion Instructions

#endregion Menu Items

#region Button Click Methods

#region OpenImage

private void btnOpenImage\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(TestReviewStaticMembers.JpgUrl);

}

#endregion OpenImage

#region OpenMp3

private void btnOpenMp3\_Click(object sender, RoutedEventArgs e)

{

System.Diagnostics.Process.Start(TestReviewStaticMembers.Mp3Url);

}

#endregion OpenMp3

#region ScoreCorrect

private void btnScoreCorrect\_Click(object sender, RoutedEventArgs e)

{

TestReviewStaticMembers.CurrentQuestionNumberString = "";

btnScoreCorrect.IsEnabled = false;

btnScoreWrong.IsEnabled = false;

TestOrReview();

}

#endregion ScoreCorrect

#region ScoreWrong

private void btnScoreWrong\_Click(object sender, RoutedEventArgs e)

{

// if review add CurrentQuestionNumberString to the end QANUmbersString

if (!IsTest)

{

// this is a review

TestReviewStaticMembers.QANUmbersString =

TestReviewStaticMembers.QANUmbersString + TestReviewStaticMembers.CurrentQuestionNumberString + '^';

}

else

{

// this is test

TestReviewStaticMembers.DelimitedWrongAnswersStr =

TestReviewStaticMembers.DelimitedWrongAnswersStr +

TestReviewStaticMembers.CurrentQuestionNumberString+'^';

TestReviewStaticMembers.NumberOfWrongAnswers++;

}

btnScoreCorrect.IsEnabled = false;

btnScoreWrong.IsEnabled = false;

TestOrReview();

}

#endregion ScoreWrong

#region ShowCorrect

private void btnShowCorrect\_Click(object sender, RoutedEventArgs e)

{

tbxCorrectAnswer.Text = TestReviewStaticMembers.ThisAnswer;

AnswerQuestions('A');

btnScoreCorrect.IsEnabled = true;

btnScoreWrong.IsEnabled = true;

btnShowCorrect.IsEnabled = false;

}

#endregion ShowCorrect

#region Button SaveEditsClicked

/// <summary>

/// When this button is clicked and changes made to the

/// Question and Answer file will be saved and the

/// QAFileChanged boolean is set to true so that

/// Results/Edits -> Save File and Return menuitem

/// is clicked the changed QA file will be updated

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnSaveEdits\_Click(object sender, RoutedEventArgs e)

{

// Get the current question

string currentQuestion = tbxQuestion.Text;

// replace all "\r\n" with ~

currentQuestion = currentQuestion.Replace("\r\n", "~");

//Get the current correct answer

string currentCorrectAnswer = tbxCorrectAnswer.Text;

// replace all "\r\n" with ~

currentCorrectAnswer = currentCorrectAnswer.Replace("\r\n", "~");

// get the QuestionNumber Key

string currentQANumStr = TestReviewStaticMembers.CurrentQuestionNumberString;

//// Get the current QADataModel

//QADataModel NewQADataModel = QAStaticMembers.ReturnQAObject(currentQANumStr);

QADataModel NewQADataModel = QAStaticMembers.QADictionary[currentQANumStr];

// replace the currentQuestion and the currentCorrectAnswer

NewQADataModel.Question = currentQuestion;

NewQADataModel.Answer = currentCorrectAnswer;

// replace the QADataModel in the QADictionary

QAStaticMembers.ReplaceThisQADataModel(currentQANumStr, NewQADataModel

// Set the QADataModel's dictionary changed to true

QAStaticMembers.DictionaryChanged = true;

}//End btnSaveEdits\_Click

#endregion Button SaveEditsClicke

#endregion Button Click Methods

#region Private Methods

#region AnswerQuestions()

/// <summary>

/// This method is called when ever

/// a new quesion is going to post

/// or when the use presses the ShowCorrectAnsser button

/// The T is either a 'Q' or an 'A'

/// References:

/// This Review Menu item clicked

/// this Test Menu item clicked

///

/// </summary>

/// <param name="T"></param>

private void AnswerQuestions(char T)

{

// Return all media buttons to disabled

ResetMediaButtons();

//reset all media field to initial values

TestReviewStaticMembers.ResetMediaFields();

//determine whether this is a question or an answer call

if (T == 'Q')

{

//This is a call for a question

// Deternine if there are questions left and if not message

string thisQuestion = TestReviewStaticMembers.ThisQuestion;

if(thisQuestion == "")

{

MessageBox.Show("There are no questions left. Call the Files Menu to either save" +

" the reuslts or clear them.");

return;

}

// get the CurrentQuestionNumberString

string CurrentQANumber = TestReviewStaticMembers.CurrentQuestionNumberString;

// set the QA number

tblkCurrentQANum.Text = CurrentQANumber;

tbxQuestion.Text = thisQuestion;

if(TestReviewStaticMembers.ThisQuestionJPG != "")

{

TestReviewStaticMembers.ThereIsAnImageFile = true;

TestReviewStaticMembers.JpgUrl = TestReviewStaticMembers.ThisQuestionJPG;

}

if(TestReviewStaticMembers.ThisQuestionMp3Url != "")

{

TestReviewStaticMembers.ThereIsASoundFile = true;

TestReviewStaticMembers.Mp3Url = TestReviewStaticMembers.ThisQuestionMp3Url;

}

}//End this is a question

else

{

string thisAnswer = TestReviewStaticMembers.ThisAnswer;

tbxCorrectAnswer.Text = thisAnswer;

if (TestReviewStaticMembers.ThisAnswerJpgUrl != "")

{

TestReviewStaticMembers.ThereIsAnImageFile = true;

TestReviewStaticMembers.JpgUrl = TestReviewStaticMembers.ThisAnswerJpgUrl;

}

if (TestReviewStaticMembers.ThisAnswerMp3Url != "")

{

TestReviewStaticMembers.ThereIsASoundFile = true;

TestReviewStaticMembers.Mp3Url = TestReviewStaticMembers.ThisAnswerMp3Url;

}

}

ResetMediaButtons();

EnableMediaButtons();

btnShowCorrect.IsEnabled = true;

}// End AnswerQuestions(T)

#endregion AnswerQuestions()

#region OpenExternalHperlink

/// <summary>

/// This method is called whenever the enabled

/// Open Image or Open Sound file button is clicked

/// </summary>

/// <param name="Url"></param>

private void OpenExternalHperlink(string Url)

{

System.Diagnostics.Process.Start(Url);

}

#endregion OpenExternalHperlink

#region ResetMediaButtons

/// <summary>

/// This private method insures that both media buttons are disabled

/// </summary>

private void ResetMediaButtons()

{

btnOpenImage.IsEnabled = false;

btnOpenMp3.IsEnabled = false;

}

private void EnableMediaButtons()

{

if (TestReviewStaticMembers.ThereIsAnImageFile)

{

btnOpenImage.IsEnabled = true;

}

if (TestReviewStaticMembers.ThereIsASoundFile)

{

btnOpenMp3.IsEnabled = true;

}

}

#endregion EnableMediaButton()

#endregion Private Methods

#region Private Methods

#region AnswerQuestions()

/// <summary>

/// This method is called when ever

/// a new quesion is going to post

/// or when the use presses the ShowCorrectAnsser button

/// The T is either a 'Q' or an 'A'

/// References:

/// This Review Menu item clicked

/// this Test Menu item clicked

///

/// </summary>

/// <param name="T"></param>

private void AnswerQuestions(char T)

{

// Return all media buttons to disabled

ResetMediaButtons();

//reset all media field to initial values

TestReviewStaticMembers.ResetMediaFields();

//determine whether this is a question or an answer call

if (T == 'Q')

{

//This is a call for a question//

// Deternine if there are questions left and if not message

string thisQuestion = TestReviewStaticMembers.ThisQuestion;

if(thisQuestion == "")

{

MessageBox.Show("There are no questions left. Call the Files Menu to either save" +

" the reuslts or clear them.");

return;

}

// get the CurrentQuestionNumberString

string CurrentQANumber = TestReviewStaticMembers.CurrentQuestionNumberString;

// set the QA number

tblkCurrentQANum.Text = CurrentQANumber;

tbxQuestion.Text = thisQuestion;

if(TestReviewStaticMembers.ThisQuestionJPG != "")

{

TestReviewStaticMembers.ThereIsAnImageFile = true;

TestReviewStaticMembers.JpgUrl = TestReviewStaticMembers.ThisQuestionJPG;

}

if(TestReviewStaticMembers.ThisQuestionMp3Url != "")

{

TestReviewStaticMembers.ThereIsASoundFile = true;

TestReviewStaticMembers.Mp3Url = TestReviewStaticMembers.ThisQuestionMp3Url;

}

}//End this is a question

else

{

string thisAnswer = TestReviewStaticMembers.ThisAnswer;

tbxCorrectAnswer.Text = thisAnswer;

if (TestReviewStaticMembers.ThisAnswerJpgUrl != "")

{

TestReviewStaticMembers.ThereIsAnImageFile = true;

TestReviewStaticMembers.JpgUrl = TestReviewStaticMembers.ThisAnswerJpgUrl;

}

if (TestReviewStaticMembers.ThisAnswerMp3Url != "")

{

TestReviewStaticMembers.ThereIsASoundFile = true;

TestReviewStaticMembers.Mp3Url = TestReviewStaticMembers.ThisAnswerMp3Url;

}

}

ResetMediaButtons();

EnableMediaButtons();

btnShowCorrect.IsEnabled = true;

}// End AnswerQuestions(T)

#endregion AnswerQuestions()

#region OpenExternalHperlink

/// <summary>

/// This method is called whenever the enabled

/// Open Image or Open Sound file button is clicked

/// </summary>

/// <param name="Url"></param>

private void OpenExternalHperlink(string Url)

{

System.Diagnostics.Process.Start(Url);

}

#endregion OpenExternalHperlink

#region ResetMediaButtons

/// <summary>

/// This private method insures that both media buttons are disabled

/// </summary>

private void ResetMediaButtons()

{

btnOpenImage.IsEnabled = false;

btnOpenMp3.IsEnabled = false;

}

private void EnableMediaButtons()

{

if (TestReviewStaticMembers.ThereIsAnImageFile)

{

btnOpenImage.IsEnabled = true;

}

if (TestReviewStaticMembers.ThereIsASoundFile)

{

btnOpenMp3.IsEnabled = true;

}

}

#endregion EnableMediaButton()

#endregion Private Methods

#region Public Methods

#region Test/Review Public Method

/// <summary>

/// This Method cycles through the QADictionary

/// until:

/// a. Test = all questions have been answered and scordx

/// b. Review = all questions have been answered correctly

/// but not scored

/// Depending on whether TextReviewStaticMember.QuestionsSeriatim is true or false

/// prepare a deliited string of question numbers

/// </summary>

internal void TestOrReview()

{

// Return all media buttons to disabled

ResetMediaButtons();

//reset all media field to initial values

TestReviewStaticMembers.ResetMediaFields();

tbxQuestion.Text = "";

tbxCorrectAnswer.Text = "";

tbxYourAnswer.Text = "";

btnShowCorrect.IsEnabled = true;

//Retrieve the first item in the QANUmbersString until its length =0

// The QANUmbersString is a string of question numbeers, beginning with 0 that are delimited with ^

while (TestReviewStaticMembers.QANUmbersString.Length != 0)

{

string CurrentQANumberString = TestReviewStaticMembers.QANUmbersString;

// get the 0th item from the QANumberString

// thisKey is the current qa pair number

string thisKey = StringHelper.ReturnItemAtPos(CurrentQANumberString, '^', 0);

//string thisKey = StringHelper.GetAndRemoveNthItem(ref CurrentQANumberString, '^', 0);

TestReviewStaticMembers.QANUmbersString = CurrentQANumberString;

//Save the current key so that if this is a review and the answer is wrong

// it can be appended to the end of the CurrentQANumberString

TestReviewStaticMembers.CurrentQuestionNumberString = thisKey;

TestReviewStaticMembers.SetCurrentQAValues(thisKey);

CurrentQANumberString = StringHelper.RemoveFirstItem(CurrentQANumberString, '^');

TestReviewStaticMembers.QANUmbersString = CurrentQANumberString;

AnswerQuestions('Q');

return;

}

if(TestReviewStaticMembers.QANUmbersString.Length == 0)

{

MessageBox.Show("This was the last QA Pair, Save or Reset the data");

}

}

#endregion Test/Review Public Method

#endregion Public Methods

}//End TestReview User Control/

}// End NameSpace NewSP.UserControl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access/Static | Return | CalledBy | Purpose |
| **TestReview.xaml.cs** | | | | |
| [SaveFile\_Click](#SaveFile_ClickTestReview) | private | void | SaveFile | NOTE: Comment Creates a result string and saves it to the QAResults folder using the DataNode’s ID as a file name` |
| [Test\_Click](#Test_Click) | private | void | Test | NOTE: Comment Sets IsTest to true; and calls TestOrReview(); |
| [Review\_Click](#Review_Click) | private | void | Review | NOTE: Comment Sets IsTest to fasle; and calls TestOrReview(); |
| [miSeriatim\_Click](#miSeriatim_Click) | private | void | miSeriatim | NOTE: Comment TestReviewStaticMembers.QuestionsSeriatim to true |
| [miRandom\_Click](#miRandom_Click) | private | void | miRandom | NOTE: NOT PROGRAMMED YET !!! |
| [UseForm\_Click](#UseForm_Click) | private | void | UseForm | NOTE: NOT PROGRAMMED YET !!! |
| [btnOpenImage\_Click](#btnOpenImage_Click) | private | void | btnOpenImage | NOTE: Comment Uses System.Diagnostics.Process.Start to open the image at TestReviewStaticMembers.JpgUrl |
| [btnOpenMp3\_Click](#btnOpenMp3_Click) | private | void | btnOpenMp3 | NOTE: Comment Uses System.Diagnostics.Process.Start to open the sound file @ TestReviewStaticMembers.Mp3Url |
| [btnShowCorrect\_Click](#btnShowCorrect_Click) | private | void | btnShowCorrect | NOTE: Comment Scores the user’s answer as correct |
| [btnScoreWrong\_Click](#btnScoreWrong_Click) | private | void | btnScoreWrong | NOTE: Comment Scores the answer wrong, and depending whether the user is reviewing the questions or taking a test will add the question to the end of the list of questions if reviewing |
| [btnSaveEdits\_Click](#btnSaveEdits_Click) | private | void | btnSaveEdit | When this button is clicked and changes made to the Question and Answer file will be saved and the QAFileChanged boolean is set to true so that Results/Edits -> Save File and Return menuitem is clicked the changed QA file will be updated |
| [AnswerQuestiond](#AnswerQuestionsDef) | private | void | btnShowCorrect\_Click  TestOrReview() | a new quesion is going to post or when the use presses the ShowCorrectAnsser button The T is either a 'Q' or an 'A' References: This Review Menu item clicked this Test Menu item clicked |
| [OpenExternalHperlink](#OpenExternalHperlinkDef) | private | void | NOT CALLED | Note: This method is not called, Eliminate??? This method is called whenever the enabled Open Image or Open Sound file button is clicked |
| [ResetMediaButton](#ResetMediaButtonDef) | private | void | AnswerQuestions()  AnswerQuestions()  TestOrReview() | This private method insures that both media buttons are disabled |
| [TestOrReview](#TestOrReviewDef) | internal | Void |  | This Method cycles through the QADictionary until: a. Test = all questions have been answered and scordx b. Review = all questions have been answered correctly but not scored Depending on whether TextReviewStaticMember.QuestionsSeriatim is true or false prepare a deliited string of question numbers |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access/Static | Return | CalledBy | Purpose |
| **.xaml.cs** | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |