Map of Static Helper Classes Methods and Globals

[CommonStaticMembers](#CommonStaticMembers)

[KeyWordStaticMembers](#KeyWordStaticMembers)

[LinkNoteStaticMembers](#LinkNoteStaticMembers)

[QAStaticMembers](#QAStaticMembers)

[StringHelpers](#StringHelpers)

[SubjectStaticMembers](#SubjectStaticMembers)

[TestReviewStaticMembers](#TestReviewStaticMembers)

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

CommonStaticMembers

[CSMMethodsTable](#CSMMethodsTable)

[CSGlobalsTable](#CSGlobalsTable)

using System.IO;

namespace NewLSP.StaticHelperClasses

{

public static class CommonStaticMembers

{

#region Properties

#region Paths

#region SubjectsNodeDataStringsPath

// This is the path to the NodeDataString text file SubjectsNodeDataStringsPath

private static string \_SubjectsNodeDataStringsPath;

public static string SubjectsNodeDataStringsPath

{

get { return \_SubjectsNodeDataStringsPath; }

set { \_SubjectsNodeDataStringsPath = value; }

}

#endregion SubjectsNodeDataStringsPath

#region NoteReferencesPath

/// <summary>

/// Holds the path the the DataNode's Note file

/// </summary>

private static string \_NoteReferencesPath;

/// <summary>

/// This is the pathe to the NoteReferencdesFiles in the

/// Common References folder

/// </summary>

public static string NoteReferencesPath

{

get { return \_NoteReferencesPath; }

set { \_NoteReferencesPath = value; }

}

#endregion NoteReferencesPath

#region HomeFolderPath;

private static string \_HomeFolderPath;

/// <summary>

/// Holds the path to the main subject folder

/// This folder holds all data relative to a given subject

/// except for NoteReferenceFiles and KeyWords in the

/// NoteReferenceFiles which can be shared between multiple

/// folders

/// </summary>

public static string HomeFolderPath

{

get { return \_HomeFolderPath; }

set { \_HomeFolderPath = value; }

}

#endregion HomeFolderPath;

#region SubjectFolderPath

private static string \_SubjectFolderPath;

/// <summary>

/// This is the path to the Subject folder

/// selected by the User in response to the opening

/// message to select or create a main Subject folder

/// </summary>

public static string SubjectFolderPath

{

get { return \_SubjectFolderPath; }

set { \_SubjectFolderPath = value; }

}

#endregion SubjectFolderPath

#region DataNodesHyperlinksPath

private static string \_DataNodesHyperlinksPath;

/// <summary>

/// This is the path to the folder that holds hyperlinks for

/// all DataNodes that have hyperlinks attached to them

/// </summary>

public static string DataNodesHyperlinksPath

{

get { return \_DataNodesHyperlinksPath; }

set { \_DataNodesHyperlinksPath = value; }

}

#endregion DataNodesHyperlinksPath

#region DataNodesNotesPath

private static string \_DataNodesNotesPath;

/// <summary>

/// This is the path to the folder that holds the

/// NoteRefereces file names and CurrentNote26Names that have been assigned to any DataNode

/// Each file is names from the designated DataNode's ID

/// and each line in each file contains a NoteName a '^' delimiter

/// and CurrentNote26Name (a String of capital alpha characters representing the

/// number of files in the Common References' NoteReferenceFiles

/// converted to base 26)

/// </summary>

public static string DataNodesNoteReferencesFilesPath

{

get { return \_DataNodesNotesPath; }

set { \_DataNodesNotesPath = value; }

}

#endregion DataNodesNotesPath

#region DataNodesQAResultsFilePath

/// <summary>

/// This is the path to the folder that holds the QAResults, the

/// results of taking a text on a QA File

/// </summary>

public static string DataNodesQAResultsFilePath { get; internal set; }

#endregion DataNodesQAResultsFilePath

#region DataNodesQAFilePath

/// <summary>

/// This is path to the Subject's QA files

/// </summary>

public static string DataNodesQAFilePath { get; internal set; }

#endregion DataNodesQAFilePath

#region ItemCountPath

/// <summary>

/// This is the path to the file that contains an integer

/// representing the number of subject items currently

/// assigned to a subject

/// </summary>

public static string ItemCountPath { get; internal set; }

#endregion ItemCountPath

#region KeyWordsDictionaryPath

private static string \_KeyWordsDictionaryPath;

/// <summary>

/// The KeyWordsDictionary contains a

/// list of KeyWord strings (where all spaces have been replaced by '\_'

/// followed by a ^ delimiter and then a

/// list of ';' delimited CurrentNote26Name names of all of the NoteReference

/// files containing that key word

/// MAKE SURE THAT EACH ALPHACHAR NAME IS SURROUNDED ON BOTH SIDES BY ';'

///

/// </summary>

public static string KeyWordsDictionaryPath

{

get { return \_KeyWordsDictionaryPath; }

set

{

KeyWordsDictionaryPath = value;

KeyWordsStaticMembers.KeyWordsDictionaryPath = \_KeyWordsDictionaryPath;

}

}

#endregion KeyWordsDictionaryPath

#endregion Paths

#region Booleans

#region HasNote Boolean

/// <summary>

/// Returns true if a designated DataNode has one or more DataNodesNoteReferencesFiles

/// </summary>

/// <param name="nodeID"></param>

/// <returns></returns>

internal static bool NodeHasNoteFile(int nodeID)

{

string DataFilePath = HomeFolderPath + "DataNodesNoteReferencesFiles\\" + nodeID.ToString() + ".txt";

if (File.Exists(DataFilePath))

{

return true;

}

else

{

return false;

}

}

#endregion HasNote boolean

#endregion Booleans

#region CurrentNoteIDInt Property

private static int \_CurrentNoteIDInt = -1;

/// <summary>

/// The CurrentNoteIDInt is the name of a selectedDataNode (i.e. "4.txt" for Jerusalem Council)

/// converted to an int (ie 4)

/// </summary>

public static int CurrentNoteIDInt

{

get { return \_CurrentNoteIDInt; }

set { \_CurrentNoteIDInt = value; }

}

#endregion CurrentNoteIDInt Property

#region CurrentNote26Name

/// <summary>

/// This string represents the CurrentNote26Name name

/// which is defined at NoteReference file creation

/// from the number of files in the NoteReferenceFiles folder

/// of a Common Referencdes folder converted to a

/// base 26 Alpha character

/// </summary>

private static string \_CurrentNote26Name = "";

public static string CurrentNote26Name

{

get { return \_CurrentNote26Name; }

set { \_CurrentNote26Name = value; }

}

#endregion CurrentNote26Name

#endregion Properties

}//End class CommonStaticMembers

}// End StaticHelperClasses namespace

CSMMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **CommonStaticMembers.cs** | | | | |
| [NodeHasNoteFile](#NodeHasNoteFileDef) (int nodeID) | internal | bool | Link\_Note.xaml.cs miShowNote\_Click  SubjectTree.xaml.cs  lvSubjects\_PreviewMouseLeftButtonUp | Returns true if a designated DataNode has one or more DataNodesNoteReferencesFiles |

CSGlobalsTable

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | CalledBy | Purpose |
| **CommonStaticMembers.cs** | | | |
| [SubjectsNodeDataStringsPath](#SubjectsNodeDataStringsPathDef) | string | [Home.xaml.cs](http://Home.xaml.cs) –  [btnOpenSubjectFolder\_Click](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#btnOpenSubjectFolder_Click);  [CreateNewNodeDataStringsFil](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#CreateNewNodeDataStringsFileDef);  CreateNewNodeDataStringsFile | This is the path to the SubjectFolder’s **NodeDataStrings.txt Text file** |
| [NoteReferencesPath](#NoteReferencesPath) | string | KeyWordStaticMembers –  ReturnNoteNameList;  LinkNoteStaticMembers –  SaveAndUpdateNoteReferenceAndKeywords;  SaveNoteReference;  SaveNoteReference  [Home.xaml.cs](http://Home.xaml.cs) –  btnOpenSubjectFolder\_Click  KeyWordControlStaticMembers.cs –  tbxAllKeyWords\_TextChanged;  Link\_Note.xaml.cs –  lbxOpenSelectedNote\_PreviewMouseLeftButtonUp;  lbxOpenSelectedNote\_PreviewMouseRightButtonU | Holds the path the References’ folder NodeReferenceFile Folder |
| [HomeFolderPath](#CSMHomeFolderPathPropDef) | string | 34 References | Holds the path to the main subject folder This folder holds all data relative to a given subject except for NoteReferenceFiles and KeyWords in the NoteReferenceFiles which can be shared between multiple folders |
| [SubjectFolderPath](#CSMSubjectFolderPathPropDef) | string | SubjectStaticMembers.xaml.cs –  SetDataNodesQAFilePath(;  [Home.xaml.cs](http://Home.xaml.cs) –  [btnOpenSubjectFolder\_Click(](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#btnOpenSubjectFolder_Click);  TestReview.xaml.cs –  SaveFile\_Click(; | This is the path to the Subject folder selected by the User in response to the opening message to select or create a main Subject folder |
| [DataNodesHyperlinksPath](#CSMDataNodesHyperlinksPathPropDef) | string | SubjectStaticMembers.xaml.cs –  OpenFiles(;  NodeHasHyperlinksFile(;  [Home.xaml.cs](http://Home.xaml.cs) –  [CreateHyperlinksFolder](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#CreateHyperlinksFolderDef)(; | This is the path to the folder that holds hyperlinks for all DataNodes that have hyperlinks attached to them |
| [DataNodesNoteReferencesFilesPath](#DataNodesNoteReferencesFilesPathPropDef) | string | 10 References | NoteRefereces file names and CurrentNote26Names that have been assigned to any DataNode Each file is names from the designated DataNode's ID and each line in each file contains a NoteName a '^' delimiter and CurrentNote26Name (a String of capital alpha characters representing the number of files in the Common References' NoteReferenceFiles converted to base 26) |
| [DataNodesQAResultsFilePath](#CSMDataNodesQAResultsFilePathPropDef) | string | [Home.xaml.cs](http://Home.xaml.cs) –  [CreateQAFilesFolder(](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#CreateQAFilesFolderDef);  [CreateQAResultFolder](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#CreateQAResultFolderDef)(; | This is the path to the folder that holds the QAResults, the results of taking a text on a QA File |
| [ItemCountPath](#CSMItemCountPathPropDef) | string | [Home.xaml.cs](http://Home.xaml.cs) –  [CreateItemCountTextFile(](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#HSMCreateItemCountTextFileDef); | This is the path to the file that contains an integer representing the number of subject items currently assigned to a subject |
| [KeyWordsDictionaryPath](#CSMKeyWordsDictionaryPathPropDef) | string | [Home.xaml.cs](http://Home.xaml.cs) -  [btnOpenSubjectFolder\_Click](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#HUCbtnOpenSubjectFolder_ClickDef)(;  ERROR ABOVE | The KeyWordsDictionary contains a list of KeyWord strings (where all spaces have been replaced by '\_' followed by a ^ delimiter and then a list of ';' delimited CurrentNote26Name names of all of the NoteReference files containing that key word MAKE SURE THAT EACH ALPHACHAR NAME IS SURROUNDED ON BOTH SIDES BY ';' |
|  |  |  |  |
|  |  |  |  |

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

KeyWordStaticMembers

[KWSMMethodsTable](#KWSMMethodsTable)

[KWSlobalsTable](#KWSlobalsTable)

using System;

using System.Collections.Generic;

using System.IO;

namespace NewLSP.StaticHelperClasses

{

public static class KeyWordsStaticMembers

{

#region Properties

#region KeyWordList

private static List<string> \_KeyWordList = new List<string>();

/// <summary>

/// This is the current list of key words in thier unconverted fors (may contain spaces)

/// </summary>

public static List<string> KeyWordList

{

get { return \_KeyWordList; }

set { \_KeyWordList = value; }

}

#endregion KeyWordList

#region SortedKeyWordList

private static List<string> \_SortedKeyWordList = new List<string>();

/// <summary>

/// This is the current list of key words in thier unconverted fors (may contain spaces)

/// </summary>

public static List<string> SortedKeyWordList

{

get { return \_SortedKeyWordList; }

set { \_SortedKeyWordList = value; }

}

#endregion SortedKeyWordList

#region KeyWordsDictionary

private static Dictionary<string, string> \_KeyWordsDictionary = new Dictionary<string, string>();

/// <summary>

/// The KeyWordDictionary uses a modified KeyWord (where all ' ' have been

/// replaces with '\_' as a Key.

/// The Value is a ';' delimited string of NoteReferenceFile CurrentNote26Names

/// NOTE: all CurrentNote26Names must be surrounded by ';' to allow

/// unique identification

/// </summary>

public static Dictionary<string, string> KeyWordsDictionary

{

get { return \_KeyWordsDictionary; }

set { \_KeyWordsDictionary = value; }

}

//static Dictionary<string, string> KeyWordsDictionary = new Dictionary<string, string>();

#endregion KeyWordsDictionary

#region ListAccess boolean

/// <summary>

/// Indicates whether the user is in the Create note reference mode = true

/// or Search for note references containing listed Key words = false

/// If true, the user can have access to add new key words to the list of key words

/// </summary>

private static bool \_ListAccess = true;

public static bool ListAccess

{

get { return \_ListAccess; }

set { \_ListAccess = value; }

}

#endregion ListAccess boolean

#region KeyWordsDictionaryPath

/// <summary>

/// This is the path to the folder that holds the Keywords dictionary

/// The Key to the KeyWords dictionary is a string of words, where all

/// spaces have been replaced with undersocres('\_')

/// The value of the KeyWords dictionary is a ';' delimited string

/// of CurrentNote26Names

private static string \_KeyWordsDictionaryPath;

public static string KeyWordsDictionaryPath

{

get { return \_KeyWordsDictionaryPath; }

set

{

\_KeyWordsDictionaryPath = value;

// Open the KeyWord Dictionary text file, split it into Key and Value and add it to the Dictioanry

string[] KeyWordsEntriesArray = File.ReadAllLines(\_KeyWordsDictionaryPath);

foreach (string line in KeyWordsEntriesArray)

{

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

KeyWordsDictionary.Add(lineItems[0], lineItems[1]);

}

}

}

#endregion KeyWordsDictionaryPath

#region ListOfKeyWordsPath

private static string \_ListOfKeyWordsPath;

/// <summary>

/// This is the path to the list of all of the orriginal current KeyWords

/// which can contain spaces

/// </summary>

public static string ListOfKeyWordsPath

{

get { return \_ListOfKeyWordsPath; }

set

{

\_ListOfKeyWordsPath = value;

//If file doesn't exist, create it

if (!File.Exists(\_ListOfKeyWordsPath))

{

var fileStream = File.Create(\_ListOfKeyWordsPath);

fileStream.Close();

}

// read Keywords into KeyWordsList

string[] KeyWordsArray = File.ReadAllLines(\_ListOfKeyWordsPath);

foreach(string line in KeyWordsArray)

{

KeyWordList.Add(line);

}

//Sort the KeyWordsList

SortKeyWordsList();

}

}

#endregion ListOfKeyWordsPath

#region List of Sorted KeyWords

private static string \_ListOfSortedKeyWordsPath;

/// <summary>

/// This is the path to the list of all of the orriginal current KeyWords

/// which can contain spaces

/// </summary>

///

public static string ListOfSortedKeyWordsPath

{

get { return \_ListOfSortedKeyWordsPath; }

set

{

\_ListOfSortedKeyWordsPath = value;

//If file doesn't exist, create it

if (!File.Exists(\_ListOfSortedKeyWordsPath))

{

var fileStream = File.Create(\_ListOfSortedKeyWordsPath);

fileStream.Close();

}

// read Keywords into KeyWordsList

string[] SortedKeyWordsArray = File.ReadAllLines(\_ListOfSortedKeyWordsPath);

foreach (string line in SortedKeyWordsArray)

{

SortedKeyWordList.Add(line);

}

}

}

#endregion List of Sorted KeyWords

#endregion Properties

#region Private Methods

#region SortKeyWordsList

/// <summary>

/// This Method sorts the list of KeyWords into alphabetic order

/// </summary>

private static void SortKeyWordsList()

{

KeyWordList.Sort();

}

#endregion SortKeyWordsList

#endregion Private Methods

#region Public Methods

#region AppendNewKeyWord

/// <summary>

/// Append a new KeyWord to the list of KeyWords in File(ListOfKeyWordsPath)

/// </summary>

/// <param name="newKeyWord"></param>

public static void AppendNewKeyWord(string newKeyWord)

{

File.AppendAllText(ListOfKeyWordsPath, newKeyWord+"\r\n");

}

#endregion AppendNewKeyWord

#region Append new Sorted Key Word

/// <summary>

/// This method received a new KeyWord

/// It first reads in the keywords in the SortedListOfKeyWords.txt file

/// It adds the new keyWord and then resorts the list

/// it then writes the updated list to the file

/// </summary>

/// <param name="keyWord"></param>

internal static void AppendNewSortedKeyWord(string keyWord)

{

string ListOfSortedKeyWordsPath = KeyWordsStaticMembers.ListOfSortedKeyWordsPath;

// check check to see if the file exists and if so read in all lines

if (File.Exists(ListOfSortedKeyWordsPath))

{

string[] ArrayOfSortedKeyWords = File.ReadAllLines(ListOfSortedKeyWordsPath);

List<string> ListOfSortedKeyWords = new List<string>(ArrayOfSortedKeyWords);

ListOfSortedKeyWords.Add(keyWord);

ListOfSortedKeyWords.Sort();

//conveert the list back to an array

ArrayOfSortedKeyWords = ListOfSortedKeyWords.ToArray();

File.WriteAllLines(ListOfSortedKeyWordsPath, ArrayOfSortedKeyWords);

}

}

#endregion Append new Sorted Key Word

#region AppendNewKeyWordDictionaryItemString()

/// <summary>

/// Receives a converted KewWord(where all ' ' have been converted to '\_'

/// and created a Dictionary string entry with

/// Key = convertedKeyWord

/// delimiter of '^'

/// and value = ";"

/// All subsequent CurrentNote26Names will be added bracked by ';' so that

/// every unique number char can be located in the search operation

/// each CurrentNote26Name char referres to the name of a NoteRefeerenceFile

/// </summary>

/// <param name="convertedKeyWord"></param>

public static void AppendNewKeyWordDictionaryItemString(string convertedKeyWord)

{

File.AppendAllText(KeyWordsDictionaryPath, convertedKeyWord + "^;\r\n");

}

#endregion AppendNewKeyWordDictionaryItemString()

#region ChangeDictionaryValue()

/// <summary>

/// This method receives a convertedKeyWord(' ' -> '\_') and

/// a ';' delimited string of CurrentNote26Names

/// it then replaces the old value with the new value (a new NoteReference now contains this key word)

/// </summary>

/// <param name="keyWord"></param>

/// <param name="delimitedNoteNamesString"></param>

internal static void ChangeDictionaryValue(string keyWord, string delimitedNoteNamesString)

{

KeyWordsDictionary[keyWord] = delimitedNoteNamesString;

}

#endregion ChangeDictionaryValue()

#region SaveDictionary()

/// <summary>

/// This method is callbe by both the

/// MainWindow's miCloseApplication\_Click menu item method

/// LinkNoteStaticMembers ProcessKeywrods method

/// It converts each dictionary item to string of a '^' delimited string of

/// KeyWord key and ';' delimited CurrentNote26Name value a

/// adds each string to a list of strings and then

/// writes them to the KeyWordsDictionaryPath, overriding any

/// preexisting data

/// </summary>

internal static void SaveDictionary()

{

// Create a List<string> to hold all of the dictionary lines

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

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

{

string key = KVP.Key;

string value = KVP.Value;

string line = key + '^' + value;

KeyWordsDictionaryList.Add(line);

}

// convert the list to an array

string[] KeyWordsDictionarArray = KeyWordsDictionaryList.ToArray();

// Write all these lines to the dictionary file

File.WriteAllLines(KeyWordsDictionaryPath, KeyWordsDictionarArray);

}// End SaveDictionary

#endregion SaveDictionary()

#region Return delimitedStringOfNoteNames()

/// <summary>

/// This method receives a converted KeyWord (all spaces have been replaced c '\_')

/// key to the KeyWordDictionary and returns the associated value

/// which is a ';' delimited list of all CurrentNote26Names

/// </summary>

/// <param name="NoteName"></param>

/// <returns></returns>

public static string delimitedStringOfNoteNames(string ConvertedKeyWord)

{

return KeyWordsDictionary[ConvertedKeyWord];

}

#endregion Return delimitedStringOfNoteNames()

#region ReturnNoteNameList

/// <summary>

/// This is called when in Search mode and the text in the list of all Key words text box is changed

/// 0. It creats a List<string> NoteNamesList of NoteReferenceNames-CurrentNote26Name

/// 1. It converte the KeyWord to a dictionary compatable form

/// 2. It gets the value associated with that key word

/// 3. It adjust the value removing the leading and terminel ';' and then

/// splits the value on '^' to get a string [] of NoteReferenceFile CurrentNote26Name

/// 4. It cycles thru the array getting the text of each NoteReferenceFile with these names

/// getting the name and creates a display string using it and the CurrentNote26Name key and

/// add it to the NoteNamesList

/// </summary>

/// <param name="searchKeyWord" is a single KeyWord to use as the Key to the KeyWordDictionar></param>

/// <returns></returns>

internal static List<string> ReturnNoteNameList(string searchKeyWord)

{

//In searchKeyword convert ' ' to '\_'

searchKeyWord = searchKeyWord.Replace(' ', '\_');

// Create the return List

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

// Get the Value from the KeyWordsDictionary for the searchKeyWord KeyWord

string CurrentNote26NameString = KeyWordsDictionary[searchKeyWord];

//Remove the initial and terminal ';' from this string

CurrentNote26NameString = CurrentNote26NameString.Substring(1);

CurrentNote26NameString = CurrentNote26NameString.Substring(0, CurrentNote26NameString.Length - 1);

// Create a string [] from CurrentNote26NameString

string[] currentNote26NamesArray = CurrentNote26NameString.Split(';');

string CommonNoteReferenceFilePath = CommonStaticMembers.NoteReferencesPath;

string NoteReferenceFileText = "";

foreach (string currentNote26Name in currentNote26NamesArray)

{

// get the value associated with this currentNote26Name

if(File.Exists(CommonNoteReferenceFilePath+"\\"+ currentNote26Name + ".txt"))

{

NoteReferenceFileText = File.ReadAllText(CommonNoteReferenceFilePath + "\\"+currentNote26Name + ".txt");

}

// Get the NoteName

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

int lengthOfSpaces = 250 - NoteName.Length;

string spacesStr = new string(' ', lengthOfSpaces);

string DisplayString = NoteName + spacesStr + '^' + currentNote26Name;

NoteNamesList.Add(DisplayString);

}

// return NoteNamesList

return NoteNamesList;

}// End ReturnNoteNameList(

#endregion ReturnNoteNameList

#endregion Public Methods

}// End KeyWordsStaticMembers class

}

\* \* \* \* \* \* \* \* \*

KWSMMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **KeyWordStaticMembers.cs** | | | | |
| [SortKeyWordsList](#KWSMSortKeyWordsListDef)( | private | void | KeyWordStaticMembers –  ListOfKeyWordsPath Prop | This Method sorts the list of KeyWords into alphabetic order |
| [AppendNewKeyWord](#KWSMAppendNewKeyWordDef)( | public | void | KeyWordControl.xaml.cx –  [tbxInput\_KeyUp](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#KWCtbxInput_KeyUpDef);  Link\_Note.xaml.cs –  [tbxInput\_KeyUp](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#tbxInput_KeyUp) | Append a new KeyWord to the list of KeyWords in File(ListOfKeyWordsPat |
| [AppendNewSortedKeyWord](#KWSMAppendNewSortedKeyWordDef)( | internal | void | **NOTE: Not References** | This method received a new KeyWord It first reads in the keywords in the SortedListOfKeyWords.txt file It adds the new keyWord and then resorts the list it then writes the updated list to the file |
| [AppendNewKeyWordDictionaryItemString](#KWSMAppendNewKeyWordDictItemStringDef)( | public | void | KeyWordControl.xaml.cs [tbxInput\_KeyUp](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#KWCtbxInput_KeyUpDef);  Link\_Note.xaml.cs –  [tbxInput\_KeyUp(](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#tbxInput_KeyUp) | Receives a converted KewWord(where all ' ' have been converted to '\_' and created a Dictionary string entry with Key = convertedKeyWord delimiter of '^' and value = ";" All subsequent CurrentNote26Names will be added bracked by ';' so that every unique number char can be located in the search operation each CurrentNote26Name char referres to the name of a NoteRefeerenceFile |
| [ChangeDictionaryValue](#KWSMChangeDictionaryValueDef)( | internal | void | LinkNoteStaticMember.cs –  UpdateKeyWordDictionary; | This method receives a convertedKeyWord(' ' -> '\_') and a ';' delimited string of CurrentNote26Names it then replaces the old value with the new value (a new NoteReference now contains this key word) |
| [SaveDictionary](#KWSMSaveDictionaryDef)( | internal | void | MainWindow.xaml.cs –  [miCloseApplication\_Click](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#miCloseApplication_Click) | This method is callbe by both the MainWindow's miCloseApplication\_Click menu item method LinkNoteStaticMembers ProcessKeywrods method It converts each dictionary item to string of a '^' delimited string of KeyWord key and ';' delimited CurrentNote26Name value a adds each string to a list of strings and then writes them to the KeyWordsDictionaryPath, overriding any preexisting data |
| [delimitedStringOfNoteNames](#KWSMdelimitedStringOfNoteNamesDef)( | public | string | LinkNoteStaticMember.cs –  UpdateKeyWordDictionary;  KeyWordControl.xaml.cs –  [tbxAllKeyWords\_TextChanged](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#KWCtbxAllKeyWords_TextChangedDef);  [tbxAllKeyWords\_TextChanged;](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#KWCtbxAllKeyWords_TextChangedDef) | This method receives a converted KeyWord (all spaces have been replaced c '\_') key to the KeyWordDictionary and returns the associated value which is a ';' delimited list of all CurrentNote26Names |
| [ReturnNoteNameList](#KWSMReturnNoteNameListDef)( | internal | List<string> | LinkNoteStaticMember.cs –  [tbxAllKeyWords\_TextChanged](https://d.docs.live.net/04e5e2c0204c421a/Documents/2021CSharpProjects/NewLSP/Logic/_Maps%20and%20Links/XAML%20CS%20Files.docx#KWCtbxAllKeyWords_TextChangedDef) | 0. It creats a List<string> NoteNamesList of NoteReferenceNames-CurrentNote26Name 1. It converte the KeyWord to a dictionary compatable form 2. It gets the value associated with that key word 3. It adjust the value removing the leading and terminel ';' and then splits the value on '^' to get a string [] of NoteReferenceFile CurrentNote26Name 4. It cycles thru the array getting the text of each NoteReferenceFile with these names getting the name and creates a display string using it and the CurrentNote26Name key and  /// add it to the NoteNamesList |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

KWSGlobalsTable

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | CalledBy | Purpose |
| **KeyWordStaticMembers.cs** | | | |
| [KeyWordList](#KWSMKeyWordListPropDef) | List<string> | KeyWordStaticMembers.cs –  ListOfKeyWordsPath(Prop);  SortKeyWordsList()  KeyWordControl.xaml.cs -  tbxInput\_KeyUp(;  tbxInput\_KeyUp(;  Link\_Note.xaml.cs –  tbxInput\_KeyUp(;  tbxInput\_KeyUp(; | This is the current list of key words in their unconverted form (may contain spaces) |
| [SortedKeyWordList](#KWSMSortedKeyWordListPropDef) | void | KeyWordStaticMembers.cs –  ListOfSortedKeyWordsPath Set; | This Method sorts the list of KeyWords into alphabetic order |
| [KeyWordsDictionary](#KWSMKeyWordsDictionaryPropDef) | Dictionary<string, string> | KeyWordStaticMembers.cs –  KeyWordsDictionaryPathProp Set;  ChangeDictionaryValue(;  SaveDictionary(;  delimitedStringOfNoteNames(;  ReturnNoteNameList(;  LinkNoteStaticMembers.cs –  ProcessKeyWords(;  ProcessKeyWords(;  KeyWordControl.xaml.cs –  tbxInput\_KeyUp(;  Link\_Note.xaml.cs –  tbxInput\_KeyUp(; | The KeyWordDictionary uses a modified KeyWord (where all ' ' have been replaces with '\_' as a Key. The Value is a ';' delimited string of NoteReferenceFile CurrentNote26Names NOTE: all CurrentNote26Names must be surrounded by ';' to allow unique identification |
| [ListAccess](#KWSMListAccessPropDef) | bool | KeyWordControl.xaml.cs –  rbtAdd\_Click(;  tbxInput\_KeyUp(;  btnRevert\_Click(;  Link\_Note.xaml.cs –  miSaveNote\_Click(;  rbtAdd\_Click(;  rbtSearch\_Click(;  tbxInput\_KeyUp(;  tbxInput\_KeyUp(;  btnRevert\_Click(;  rbtEdit\_Click; | Indicates whether the user is in the Create note reference mode = true or Search for note references containing listed Key words = false If true, the user can have access to add new key words to the list of key words |
| [KeyWordsDictionaryPath](#KWSMKeyWordsDictionaryPathPropDef) | string | CommonStaticMembers.cs –  KeyWordsDictionaryPath Prop;  KeyWordStaticMembers.cs –  AppendNewKeyWordDictionaryItemString(;  SaveDictionary(; | This is the path to the folder that holds the Keywords dictionary The Key to the KeyWords dictionary is a string of words, where all  spaces have been replaced with undersocres('\_') The value of the KeyWords dictionary is a ';' delimited string of CurrentNote26Names |
| [ListOfKeyWordsPath](#KWSMListOfKeyWordsPathPropDef) | string | KeyWordStaticMembers.cs –  AppendNewKeyWord(;  [Home.xaml.cs](http://Home.xaml.cs) –  btnOpenSubjectFolder\_Click(; | This is the path to the list of all of the orriginal current KeyWords which can contain spaces |
| [ListOfSortedKeyWordsPath](#KWSMListOfSortedKeyWordsPathPropDef) | string | KeyWordStaticMembers.cs –  AppendNewSortedKeyWord(;  [Home.xaml.cs](http://Home.xaml.cs) –  btnOpenSubjectFolder\_Click(; | This is the path to the list of all of the orriginal current KeyWords which can contain spaces |

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

LinkNoteStaticMembers

[LNSMMethodsTable](#LNSMMethodsTable)

[LNSMGlobalsTable](#LNSMGlobalsTable)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using NewLSP.DataModels;

namespace NewLSP.StaticHelperClasses

{

public static class LinkNoteStaticMembers

{

#region Properties and fields

#region Hyperlink

private static string \_Hyperlink;

/// <summary>

/// This is the path to a hyperlinked file chosen by the user as

/// a hyperlink file using the OpenFildDialog

/// </summary>

public static string Hyperlink

{

get { return \_Hyperlink; }

set { \_Hyperlink = value; }

}

#endregion Hyperlink

#region FileType

private static string \_FileType;

/// <summary>

/// This gets and sets the file type of multiple different file formats

/// chosed by the user, ie web, docx, jpg etc

/// </summary>

public static string FileType

{

get { return \_FileType; }

set { \_FileType = value; }

}

#endregion FileType

#region BookMarks

private static List<string> \_BookMarks = new List<string>();

/// <summary>

/// This stores any book mark to a docx file specified by the user,

/// if the user opens that docx file this bookmark is copied to the

/// ScratchPad and can be recalled to use in th GoTo bookmark menu

/// </summary>

public static List<string> BookMarks

{

get { return \_BookMarks; }

set { \_BookMarks = value; }

}

#endregion BookMarks

#region HyperlinkName

private static string \_HyperlinkName;

public static string HyperlinkName

{

get { return \_HyperlinkName; }

set { \_HyperlinkName = value; }

}

#endregion HyperlinkName

#region HyperlinkDictionary

/// <summary>

/// This Dictionary uses the item's position in the lbxLinks ListBox

/// as the Key and a HyperlinkObject as the value

/// The HyperlinkObject contains: 1) the BookMark, 2) the Name,

/// 3) the Url and 4) the FileType

/// </summary>

public static Dictionary<int, LinkNoteModel.HyperlinkObject> HyperlinkDictionary = new Dictionary<int, LinkNoteModel.HyperlinkObject>();

#endregion HyperlinkDictionary

#region HyperlinkUrls

/// <summary>

/// This is a list of hyperlinks to files of web sites chosen by the user

/// to be associated with a givin DataNode

/// </summary>

public static List<string> HyperlinkUrls = new List<string>();

#endregion HyperlinkUrls

#region HyperlinkStringsList

/// <summary>

/// a delimites string containing the elements of

/// a hyperlink objects added to the list<string> HyperlinkStringsList

/// Called by LinkNote.xaml.cs miSaveHyperlink\_Clic

/// </summary>

public static List<string> HyperlinkStringsList = new List<string>();

#endregion HyperlinkStringsList

#region ListOfNoteNames

private static List<string> \_ListOfNoteNames;

public static List<string> ListOfNoteNames

{

get { return \_ListOfNoteNames; }

set { \_ListOfNoteNames = value; }

}

#endregion ListOfNoteNames

#region ListOfNoteReferceFileNames

private static List<string> \_ListOfNoteReferceFileNames;

/// <summary>

/// This is the list on NoteReference Char26Names belonging to a DataNode

/// </summary>

public static List<string> ListOfNoteReferceFileNames

{

get { return \_ListOfNoteReferceFileNames; }

set { \_ListOfNoteReferceFileNames = value; }

}

#endregion ListOfNoteReferceFileNames

#region ListOfNoteHyperlinks

private static List<string> \_ListOfNoteHyperlinks;

public static List<string> ListOfNoteHyperlinks

{

get { return \_ListOfNoteHyperlinks; }

set { \_ListOfNoteHyperlinks = value; }

}

#endregion ListOfNoteHyperlinks

#region ListOfNoteBookMarks

private static List<string> \_ListOfNoteBookMarks;

public static List<string> ListOfNoteBookMarks

{

get { return \_ListOfNoteBookMarks; }

set { \_ListOfNoteBookMarks = value; }

}

#endregion ListOfNoteBookMarks

#region ListOfNoteKeyWords

private static List<string> \_ListOfNoteKeyWords;

public static List<string> ListOfNoteKeyWords

{

get { return \_ListOfNoteKeyWords; }

set { \_ListOfNoteKeyWords = value; }

}

#endregion ListOfNoteKeyWords

#region CurrentNote26Name

/// <summary>

/// This is the name of the currently active CommonReferences NoteReferenceFile

/// is is created by taking the current number of files in the

/// NoteReferenceFile folder and converting it to a base 26 integer which

/// uses all Alpha Characters from A to Z where A = 0 and Z= 25 and AA = 26 etc

/// </summary>

public static string CurrentNote26Name;

#endregion CurrentNote26Name

#region ListBoxOfSelectedNotesList

private static List<string> \_ListBoxOfSelectedNotesList;

public static List<string> ListBoxOfSelectedNotesList

{

get { return \_ListBoxOfSelectedNotesList; }

set { \_ListBoxOfSelectedNotesList = value; }

}

#endregion ListBoxOfSelectedNotesList

#region NewDataNodesNoteReferenceFileName

private static string \_NewDataNodesNoteReferenceFileName;

public static string NewDataNodesNoteReferenceFileName

{

get { return \_NewDataNodesNoteReferenceFileName; }

set { \_NewDataNodesNoteReferenceFileName = value; }

}

#endregion NewDataNodesNoteReferenceFileName

#region DataNodesNoteReferenceString

private static string \_DataNodesNoteReferenceString;

public static string DataNodesNoteReferenceString

{

get { return \_DataNodesNoteReferenceString; }

set { \_DataNodesNoteReferenceString = value; }

}

#endregion DataNodesNoteReferenceString

#region EditingBoolean

private static bool \_EditingBoolean = false;

public static bool EditingBoolean

{

get { return \_EditingBoolean ; }

set { \_EditingBoolean = value; }

}

#endregion EditingBoolean

#endregion Properties

#region Public Methods

#region AddHyperlinkToList

/// <summary>

/// Adds a delimites string containing the elements of

/// a hyperlink object to the list<string> HyperlinkStringsList

/// Called by LinkNote.xaml.cs miSaveHyperlink\_Clic

/// </summary>

/// <param name="delimitedHyperlink"></param>

public static void AddHyperlinkToList(string delimitedHyperlink)

{

HyperlinkStringsList.Add(delimitedHyperlink);

}

#endregion AddHyperlinkToList

#region AddItemToHyperlinkDictionary

public static void AddItemToHyperlinkDictionary(int cntr, LinkNoteModel.HyperlinkObject thisHyperlinkObject)

{

HyperlinkDictionary.Add(cntr, thisHyperlinkObject);

}

#endregion AddItemToHyperlinkDictionary

#region Get Hyperlink

public static LinkNoteModel.HyperlinkObject GetHyperlinkObject(int itemsIndex)

{

return (LinkNoteModel.HyperlinkObject) HyperlinkDictionary[itemsIndex];

}

/// <summary>

/// Called when the LinkNote control is expanded

/// It reads all on the lines in the hyperlink file into the DataNodeHyperlinkArray and then

/// for each line it:

/// 1) Adds the delimitd line to the HyperlinkStringsList

/// 2) Sets the int HyperlinkCntr to 0

/// 3) It cycles through each line the DataNodeHyperlinkArray

/// creates the HyperlinkDictionary using its position

/// in the array as the key and the component itmes as

/// properties of the LinkNoteModel.HyperlinkObject that is the value

/// </summary>

internal static void SetHyperlinkStringsList()

{

//read in the hyperlinks file

string[] DataNodeHyperlinkArray =

File.ReadAllLines(CommonStaticMembers.HomeFolderPath + "Hyperlinks\\" + SubjectStaticMembers.DataNode.ID.ToString() + ".txt");

// Create a counter to use as the Key to the Hyperlinks dictionary

int HyperlinkCntr = 0;

//process each delimited line

foreach (string line in DataNodeHyperlinkArray)

{

// add each line to the HyperlinkStringsList

HyperlinkStringsList.Add(line);

// Get the items in each line

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

HyperlinkUrls.Add(HyperlinkLineArray[0]);

// create a new Hyperlink object

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

thisHyperlinkObject.Url = HyperlinkLineArray[0];

thisHyperlinkObject.FileType = HyperlinkLineArray[1];

thisHyperlinkObject.BookMark = HyperlinkLineArray[2];

HyperlinkDictionary.Add(HyperlinkCntr, thisHyperlinkObject);

HyperlinkCntr++;

}// End foreach (string line in DataNodeHyperlinkArray

}// End SetHyperlinkStringsList

#endregion Get Hyperlink

#region Create 4 Lists from DataNode's Note file

/// <summary>

/// This method receives a string array

/// where every line is a '^' delimited string of note references

/// It creates 4 lists (Names, Hyperlinks, BookMarkd, and KeyWords)

/// and it returns the List of Note Names

/// </summary>

/// <param name="lines"></param>

internal static List<string> CreateNoteLists(string[] lines)

{

//Docetism-Wikipeida^https://en.wikipedia.org/wiki/Docetism^^Docetism;#Definition;Ignatius of Antioch;1 John;

// create Lists

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

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

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

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

foreach (string line in lines)

{

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

NamesList.Add(thisLineArray[0]);

HyperlinksList.Add(thisLineArray[1]);

BookmarkList.Add(thisLineArray[2]);

KeyWordsList.Add(thisLineArray[3]);

}

return NamesList;

}// End CreateNoteLists

#endregion Create 4 Lists from DataNode's Note file

#region ReadInNotesFile

public static void ReadInNotesFile()

{

//Create path to the DataNodeIDs notes file

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

CommonStaticMembers.CurrentNoteIDInt = SubjectStaticMembers.DataNode.ID;

string DataNodesNotesPath = CommonStaticMembers.DataNodesNoteReferencesFilesPath;

// create a string array of all lines in the DataNodeReferences file

string[] lines = System.IO.File.ReadAllLines(DataNodesNotesPath);

// Create a new instance of ListBoxOfSelectedNotesList

ListBoxOfSelectedNotesList = new List<string>();

// Convert lines to a display string and add it to the lbxSelectedNodes

foreach (string line in lines)

{

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

string NoteReferenceFileName = StringHelper.ReturnItemAtPos(line, '^', 1);

int NumberOfSpaces = 250 - NoteName.Length;

string spacedStr = new string(' ', NumberOfSpaces);

string outputStr = NoteName + spacedStr + '^' + NoteReferenceFileName;

ListBoxOfSelectedNotesList.Add(outputStr);

}

// Crear and instantiate all the required lists

ListOfNoteNames = new List<string>();

ListOfNoteReferceFileNames = new List<string>();

foreach(string line in lines)

{

// create an array of the note components

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

ListOfNoteNames.Add(NoteComponents[0]);

ListOfNoteReferceFileNames.Add(NoteComponents[1]);

}

}

#endregion ReadInNotesFile

#region SaveAndUpdateNoteReferenceAndKeywords()

/// <summary>

/// This Method saves a note reference called Clicking the miSaveNote menu item

/// The TASKS if must perform are to:

/// 1. \*Save the NoteReferenceStr to the NoteReference file

/// 2. \*Cycle through the ';' delimited keywords

/// a. Eliminating those that begin with '#'

/// b. converting ' ' to '\_' in the remaining

/// c. Cycle through the list of Remaining key words

/// 1) Checking to the the that keyword's dictionary value contains the

/// CurrentNote26Name, and if not appending it and then saving the

/// revused value to the KeyWordDictionary object

/// 2) When all Keywords have been processed writing th KeyWordDictionary to its file

/// 3. \*Create a DataNodesNoteReferenceString

/// </summary>

public static void SaveAndUpdateNoteReferenceAndKeywords(string NoteReferenceStr)

{

//TASK1 Save the NoteReferenceStr to the NoteReference file and Add a link to the DataNode's DataNodeNoteReferenceFile

// Determine if it is an edited file

if(EditingBoolean && (CommonStaticMembers.CurrentNote26Name != ""))

{

// This is an edited note reference

// Get path to the Common NoteReferenceFiles

string NoteReferenceFilePath = CommonStaticMembers.NoteReferencesPath + "\\" + CommonStaticMembers.CurrentNote26Name + ".txt";

File.WriteAllText(NoteReferenceFilePath, NoteReferenceStr);

//Add a link to the DataNode's DataNodeNoteReferenceFile

// Get the DataNodes DataNodeNoteReferenceFile

string DataNodesID = CommonStaticMembers.CurrentNoteIDInt.ToString();

string DataNodesReferenceNotesPath = CommonStaticMembers.DataNodesNoteReferencesFilesPath+ DataNodesID+".txt";

// Test to see if the file exists and if so read all lines

string[] DataNodesCurrentReferencesArray = null;

if (File.Exists(DataNodesReferenceNotesPath))

{

DataNodesCurrentReferencesArray = File.ReadAllLines(DataNodesReferenceNotesPath);

// Construct the DataNode's reference string

string thisNoteName = StringHelper.ReturnItemAtPos(NoteReferenceStr, '^', 0);

string DataNodesReferenceStr = thisNoteName + '^' + CommonStaticMembers.CurrentNote26Name;

// create a bool to indicate wether this refernces is already in the DataNodes referencew

bool ReverencePresent = false;

// Cycle through DataNodesCurrentReferencesArray seein if DataNodesReferenceStr is present and if it isn't add it

foreach (string line in DataNodesCurrentReferencesArray)

{

if (line.IndexOf(DataNodesReferenceStr) == 0)

{

ReverencePresent = true;

break;

}

}

// If refeerence is not present add it

if (!ReverencePresent)

{

List<string> ListOfDataNodesReferences = DataNodesCurrentReferencesArray.ToList();

ListOfDataNodesReferences.Add(DataNodesReferenceStr);

File.WriteAllLines(DataNodesReferenceNotesPath, ListOfDataNodesReferences);

}

}

else

{

// Construct the DataNode's reference string

string thisNoteName = StringHelper.ReturnItemAtPos(NoteReferenceStr, '^', 0);

string DataNodesReferenceStr = thisNoteName + '^' + CommonStaticMembers.CurrentNote26Name;

File.WriteAllText(DataNodesReferenceNotesPath, DataNodesReferenceStr);

}

ProcessKeyWords(NoteReferenceStr);

return;

}

else

{

SaveNoteReference(NoteReferenceStr);

}

//TASK Create a DataNodesNoteReferenceString and save it to the DataNode's Reference File

ProcessKeyWords(NoteReferenceStr);

//Create a DataNodesNoteReferenceString

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

DataNodesNoteReferenceString = NoteName + '^' + CommonStaticMembers.CurrentNote26Name + "\r\n";

//Get the DataNode's name

//string DataNodeNoteReferenceFilePath = SubjectStaticMembers.GetDataNodeNoteReferenceFilePath();

//Append this to the DataNodes DataNodesReferenceFile

if (File.Exists(CommonStaticMembers.DataNodesNoteReferencesFilesPath))

{

File.AppendAllText(CommonStaticMembers.DataNodesNoteReferencesFilesPath, DataNodesNoteReferenceString);

}

else

{

// Create a new DataNodesReferenceFile

FileStream fs = File.Create(CommonStaticMembers.DataNodesNoteReferencesFilesPath);

fs.Close();

File.AppendAllText(CommonStaticMembers.DataNodesNoteReferencesFilesPath, DataNodesNoteReferenceString);

}

}// End SaveAndUpdateNoteReferenceAndKeywords(

#endregion SaveAndUpdateNoteReferenceAndKeywords()

LNSMMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **LinkNoteStaticMembers.cs** | | | | |
| [AddHyperlinkToList](#LNSMAddHyperlinkToListDef)( | public | void | Link\_Note.xaml.cs –  SaveHyperlink(; | Adds a delimites string containing the elements of a hyperlink object to the list<string> HyperlinkStringsList Called by LinkNote.xaml.cs miSaveHyperlink\_Clic |
| [AddItemToHyperlinkDictionary](#LNSMAddItemToHyperlinkDictionaryDef)( | public | void | Link\_Note.xaml.cs –  miOpenHyperLink\_Click(; | NOTE: Comment Adds a LinkNoteModel.HyperlinkObject to the HyperlinkDictionary |
| [GetHyperlinkObject](#LNSMGetHyperlinkObjectDef)( | public | LinkNoteModel.  HyperlinkObject | Link\_Note.xaml.cs –  lbxLinks\_PreviewMouseUp(; | NOTE: Comment Returns a LinkNoteModel HyperlinkObject based on the itemsIndex to the HyperlinkDictionary |
| [SetHyperlinkStringsList](#LNSMSetHyperlinkStringsListDef)( | internal | void | MainWindow.xaml.cs-  miLinkNotes\_Click(; | Called when the LinkNote control is expanded It reads all on the lines in the hyperlink file into the DataNodeHyperlinkArray and then for each line it: 1) Adds the delimitd line to the HyperlinkStringsList 2) Sets the int HyperlinkCntr to 0 3) It cycles through each line the DataNodeHyperlinkArray creates the HyperlinkDictionary using its position in the array as the key and the component itmes as properties of the LinkNoteModel.HyperlinkObject that is the value |
| [CreateNoteLists](#LNSMCreateNoteListsDef)( | internal | List<string> | NOTE!!! NOT CALLED | This method receives a string array where every line is a '^' delimited string of note references It creates 4 lists (Names, Hyperlinks, BookMarkd, and KeyWords) and it returns the List of Note Names |
| [ReadInNotesFile](#NLSMReadInNotesFileDef)( | public | void | Link\_Note.xaml.cs –  ReadNotesIntoSelectNoteListBox(; | NOTE: Comment Gets the DataNode’s ID and uses it to create a path to the the Subject’s DataNodesNoteReferencesFiles folder. It then reads in the DataNode’s file in that folder and extracts all of the NoteNames and NotePaths and adds them to a List of each type of component |
| [SaveAndUpdateNoteReferenceAndKeywords](#LNSMSaveUpdateNoteReferenceAndKeywordDef)( | public | void | Link\_Note.xaml.cs –  miSaveNote\_Click(;  miSaveNote\_Click(; | This Method saves a note reference. Called by Clicking the miSaveNote menu item. The TASKS if must perform are to: 1. \*Save the NoteReferenceStr to the NoteReference file 2. \*Cycle through the ';' delimited keywords a. Eliminating those that begin with '#' b. converting ' ' to '\_' in the remaining c. Cycle through the list of Remaining key words 1) Checking to the the that keyword's dictionary value contains the CurrentNote26Name, and if not appending it and then saving the revused value to the KeyWordDictionary object 2) When all Keywords have been processed writing th KeyWordDictionary to its file 3. \*Create a DataNodesNoteReferenceString |

LNSMGlobalsTable

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | CalledBy | Purpose |
| **LinkNoteStaticMembers.cs** | | | |
| [Hyperlink](#LNSMHyperlinkPropDef) | string | Link\_Note.xaml.cs –  miOpenFileDialog\_Click(;  SaveHyperlink(;  tbxHyperlink\_TextChanged(; | This is the path to a hyperlinked file chosen by the user as a hyperlink file using the OpenFildDialog |
| [FileType](#LNSMFileTypePropDef) | string | Link\_Note.xaml.cs –  miOpenFileDialog\_Click(;  nine times | This gets and sets the file type of multiple different file formats chosen by the user, ie web, docx, jpg etc |
| [BookMarks](#LNSMBookMarksPropDef) | List<string> | Link\_Note.xaml.cs –  SaveHyperlink(; x2  tbxBookMark\_TextChanged(; | This stores any book mark to a docx file specified by the user, if the user opens that docx file this bookmark is copied to the ScratchPad and can be recalled to use in the GoTo bookmark menu |
| [HyperlinkName](#LNSMHyperlinkNamePropDef) | string | Link\_Note.xaml.cs –  tbxLinkName\_TextChanged(; | NOTE: Comment Presumable the name that the user gives to the hyperlinked file |
| [HyperlinkDictionary](#LNSMHyperlinkDictionaryGlobalDef) | **Global**  Dictionary<int, LinkNoteModel.HyperlinkObject> | Link\_Note.xaml.cs –  miOpenHyperLink\_Click(;  miResetPage\_Click(;  SaveHyperlink(;  miOpenHyperLink\_Click(;  miResetPage\_Click(; x3 | This Dictionary uses the item's position in the lbxLinks ListBox as the Key and a HyperlinkObject as the value The HyperlinkObject contains: 1) the BookMark, 2) the Name, 3) the Url and 4) the FileType |
| [HyperlinkUrls](#LNSMHyperlinkUrlsGlobalDef) | **Global** List<string> | NOT CALLED | This is a list of hyperlinks to files of web sites chosen by the user to be associated with a givin DataNode |
| [HyperlinkStringsList](#LNSMHyperlinkStringsListGlobalDef) | List<string> | Link\_Note.xaml.cs –  SaveHyperlink(; x2 | a delimites string containing the elements of a hyperlink objects added to the list<string> HyperlinkStringsList  called by LinkNote.xaml.cs miSaveHyperlink\_Click |
| [ListOfNoteNames](#LNSMListOfNoteNamesPropDef) | List<string> | LinkNoteStaticMembers.cs –  ReadInNotesFile(; x2  Link\_Note.xaml.cs –  PopulateNoteListBox(; | a delimites string containing the elements of a hyperlink objects added to the list<string> HyperlinkStringsList  Called by LinkNote.xaml.cs miSaveHyperlink\_Click |
| [ListOfNoteReferceFileNames](#LNSMListOfNoteReferceFileNamesPropDef) | List<string> | LinkNoteStaticMembers.cs –  ReadInNotesFile(; x2 | This is the list on NoteReference Char26Names belonging to a DataNode |
| [ListOfNoteHyperlinks](#LNSMListOfNoteHyperlinksPropDef) | List<string> | NOTE!!! NOT CALLED | NOTE: Comment Not DEfined |
| [ListOfNoteBookMarks](#LNSMListOfNoteBookMarksPropDef) | List<string> | NOTE!!! NOT CALLED | NOTE: Comment Not DEfined |
| [ListOfNoteKeyWords](#LNSMListOfNoteKeyWordsPropDef) | List<string> | NOTE!!! NOT CALLED | NOTE: Comment Not DEfined |
| [CurrentNote26Name](#LNSMCurrentNote26NameGlobalDef) | **Global** string | NOTE!!! NOT CALLED | This is the name of the currently active CommonReferences NoteReferenceFile is is created by taking the current number of files in the NoteReferenceFile folder and converting it to a base 26 integer which uses all Alpha Characters from A to Z where A = 0 and Z= 25 and AA = 26 etc |
| [ListBoxOfSelectedNotesList](#LNSMListBoxOfSelectedNotesListPropDef) | List<string> | Link\_Note.xaml.cs –  ReadInNotesFile(; | NOTE: Comment Not DEfined |
| [NewDataNodesNoteReferenceFileName](#LNSMNwDatNodNoteReferenceFileNamePropDef) | string | NOTE!!! NOT CALLED | NOTE: Comment Not DEfined |
| [DataNodesNoteReferenceString](#LNSMDataNodesNoteReferenceStringPropDef) | string | LinkNoteStaticMembers.cs –  SaveAndUpdateNoteReferenceAndKeywords(;  x3  Link\_Note.xaml.cs –  miSaveNote\_Click(; | NOTE: Comment Not DEfined |
| [EditingBoolean](#EditingBoolean) | bool | LinkNoteStaticMembers.cs –  SaveAndUpdateNoteReferenceAndKeywords(;  Link\_Note.xaml.cs –  miSaveNote\_Click(; x3  miResetPage\_Click(; | NOTE: Comment Presumably set true if the Editing KeyWord radio button is checked |

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

QAStaticMembers

[QASMMethodsTable](#QASMMethodsTable)

[QASMGlobalsTable](#QASMGlobalsTable)

using System.Collections.Generic;

using NewLSP.DataModels;

using System.IO;

using System;

namespace NewLSP.StaticHelperClasses

{

public static class QAStaticMembers

{

#region public fields and properties

#region QADictionary public field

/// <summary>

/// This dictionary uses the question number as a string

/// as the Key and an instation of a QADataModel object

/// as the value

/// </summary>

public static Dictionary<string, QADataModel> QADictionary = new Dictionary<string, QADataModel>();

#endregion QADictionary public field

#region CurrentQANumberInt property

/// <summary>

/// gets and sets the Current QANumber integer

/// </summary>

private static int \_CurrentQANumberInt = 0;

public static int CurrentQANumberInt

{

get { return \_CurrentQANumberInt; }

set { \_CurrentQANumberInt = value; }

}

#endregion CurrentQANumberInt property

#region Public property dictionary changed

private static bool \_DictionaryChanged = false;

public static bool DictionaryChanged

{

get { return \_DictionaryChanged; }

set { \_DictionaryChanged = value; }

}

#endregion Public property dictionary change

#endregion public fields and properties

#region Public methods

#region Return DataModel obect in Array public method

/// <summary>

/// This method receives a key to a QAMOdelObject as a string

/// and returns the QAModelObject associted with it in the

/// QADictionary

/// </summary>

/// <param name="NumberKey"></param>

/// <returns></returns>

public static QADataModel ReturnQAObject(string NumberKey)

{

QADataModel qADataModelObject = QADictionary[NumberKey];

return qADataModelObject;

}

#endregion Return DataModel obect in Array

#region CreateNewQAFile public method

/// <summary>

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

/// Called by CreateEditQA.xaml.cs NewFile\_Click(

/// it 1: Sets the dataNodesQAFilePath and then passes

/// that to the File.Create procedure in system.IO

/// </summary>

internal static void CreateNewQAFile()

{

// Get DataNodesQAFielPath

string dataNodesQAFilePath = SubjectStaticMembers.GetDataNodesQAFilePath();

File.Create(dataNodesQAFilePath);

}

#endregion CreateNewQAFile

#region Check to see if QAFile exists

/// <summary>

/// This method used the path to the currently

/// selected DataNode and returns true if

/// a file already exists or false if it doesn't

/// Used by CreateEditQA.xaml.cs

/// </summary>

/// <returns></returns>

public static bool DoesQAFileExist()

{

if (File.Exists(SubjectStaticMembers.GetDataNodesQAFilePath()))

{

return true;

}

else

{

return false;

}

}

#endregion Check to see if QAFile exists

#region Add a new QADataModel object to the Dictionary

/// <summary>

/// Receives a question number as a string and a QADataModel object

/// and stores them in the QADictionary

/// </summary>

/// <param name="NumberKey"></param>

/// <param name="qADataModelObject"></param>

public static void AddQAObjectToDictionary(string NumberKey, QADataModel qADataModelObject)

{

QADictionary.Add(NumberKey, qADataModelObject);

}// End AddQAObjectToDictionary

#endregion Add a new QADataModel object to the Dictionary

#region Save the QADictionary to an external .txt. file

/// <summary>

/// This method saves the current QADictionary to a text file

/// Called By QACreateEdit.xaml.cs

/// </summary>

internal static void SaveQADictionary()

{

// Create a List<string> to hold the delimited QAFile lines

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

// cycle thru the QADictionary creating and storing the QALines

foreach (KeyValuePair<string, QADataModel> KVP in QADictionary)

{

string key = KVP.Key;

QADataModel thisQAObject = KVP.Value;

string OutputString = key + '^' + ConvertQADataObjectToDelimitedString(thisQAObject);

QALines.Add(OutputString);

}

string dataNodesQAFilePath = SubjectStaticMembers.GetDataNodesQAFilePath();

File.WriteAllLines(dataNodesQAFilePath, QALines);

}

#endregion Save the QADictionary

#region Replace an Entry in the QA Dictionary

public static void ReplaceThisQADataModel(string placeKey, QADataModel thisQADataModel)

{

QADictionary[placeKey] = thisQADataModel;

}// End

#endregion Replace an Entry in the QA Dictionary

#region Read the QAFile into the Dictionary ReadQAFileIntoDictionary()

public static void ReadQAFileIntoDictionary()

{

// Read all of the lines in the qa file into an array

string[] QALinesArray = File.ReadAllLines(SubjectStaticMembers.GetDataNodesQAFilePath());

//process each delimited line converting it into a key(question number)

// and value QADataModel object

foreach (string line in QALinesArray)

{

// split this string on ^

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

// Create a new QADataModel object

QADataModel qADataModel = new QADataModel();

//Get the key and store it

string Key = thisQALineArray[0];

qADataModel.QANumber = Int32.Parse(Key);

string question = thisQALineArray[1];

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

qADataModel.Question = question;

string answer = thisQALineArray[2];

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

qADataModel.Answer = answer;

qADataModel.QuestionJpgUrl = thisQALineArray[3];

qADataModel.QuestionMp3Url = thisQALineArray[4];

qADataModel.AnswerJpgUrl = thisQALineArray[5];

qADataModel.AnswerMp3Url = thisQALineArray[6];

QAStaticMembers.QADictionary.Add(Key, qADataModel);

}// End for each line

}

#endregion Read the QAFile into the DictionaryReadQAFileIntoDictionary()

#endregion Public methods

#region Private Methods

#region Convert QADataModel object to delimited string

/// <summary>

/// This method converts a QADataModel object into a ^

/// delimited string for external storage in a .txt file

/// </summary>

/// <param name="qADataModel"></param>

/// <returns></returns>

private static string ConvertQADataObjectToDelimitedString(QADataModel qADataModel)

{

string ReturnString = qADataModel.Question + '^' + qADataModel.Answer + '^' + qADataModel.QuestionJpgUrl +

'^' + qADataModel.QuestionMp3Url + '^' + qADataModel.AnswerJpgUrl + '^' + qADataModel.AnswerMp3Url;

return ReturnString;

}

#endregion Convert QADataModel object to delimited string

#endregion Private Methods

}

}

QASMMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **QAStaticMembers.cs** | | | | |
| [ReturnQAObject](#QASMReturnQAObjectDef)( | QADataModel |  | NOT CALLED | This method receives a key to a QAMOdelObject as a string and returns the QAModelObject associted with it in the QADictionary |
| [CreateNewQAFile](#QASMCreateNewQAFileDef)( | void |  | CreateEditQA.xaml.cs –  NewFile\_Click(; x2 | Check to insure that the file doesn't exist and then creat it Called by CreateEditQA.xaml.cs NewFile\_Click(it 1: Sets the dataNodesQAFilePath and then passesthat to the File.Create procedure in system.IO |
| [DoesQAFileExist](#QASMDoesQAFileExistDef)( |  |  | MainWindow.xaml.cs –  miTest\_Click(;  CreateEditQA.xaml.cs –  NewFile\_Click(; | This method used the path to the currently selected DataNode and returns true if a file already exists or false if it doesn't Used by CreateEditQA.xaml.cs |
| [AddQAObjectToDictionary](#QASMAddQAObjectToDictionaryDef)( | void |  | CreateEditQA.xaml.cs –  AddThisQAPairToTheDictionary(; | This method saves the current QADictionary to a text file Called By QACreateEdit.xaml.cs |
| [SaveQADictionary](#QASMSaveQADictionaryDef)( | void |  | CreateEditQA.xaml.cs –  SaveFile\_Click(;  TestReview.xaml.cs –  SaveFile\_Click(; | NOTE: Comment Not defined |
| [ReplaceThisQADataModel](#QASMReplaceThisQADataModelDef)( | void |  | TestReview.xaml.cs –  btnSaveEdits\_Click(; | NOTE: Comment Not defined |
| [ReadQAFileIntoDictionary](#QASMReadQAFileIntoDictionaryDef)( | void |  | TestReview.xaml.cs –  SetupDictionaryAndQAString(; | NOTE: Comment Presumable to read the data in the QAFile into the QADictionary |
| [ConvertQADataObjectToDelimitedString](#QASMCnvtQADataObjectToDelimitedStringDef)( | private string |  | QAStaticMembers.cs –  SaveQADictionary(; | This method converts a QADataModel object into a ^ delimited string for external storage in a .txt file |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

QASMGlobalsTable

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | CalledBy | Purpose |
| **QAStaticMembers.cs** | | | |
| [QADictionary](#QASMQADictionaryGlobalDef) | **Global** Dictionary<string, QADataModel> |  | This dictionary uses the question number as a string as the Key and an instation of a QADataModel object as the value |
| [CurrentQANumberInt](#QASMCurrentQANumberIntPropDef) | int |  | gets and sets the Current QANumber integer |
| [DictionaryChanged](#QASMDictionaryChangedPropDef) | bool | QAStaticMembers.cs -  ReadQAFileIntoDictionary(;  CreateEditQA.xaml.cs -  Append\_Click(;  SubjectTree.xaml.cs –  rbDataNode\_Click(;  TestReview.xaml.cs –  SaveFile\_Click(;  btnSaveEdits\_Click(;  SetCurrentQAValues(;  SetupDictionaryAndQAString(; | NOTE: Comment Not defined |

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

StringHelpers

[SHMethodsTable](#SHMethodsTable)

[SHGlobalsTable](#SHGlobalsTable)

using System.Collections.Generic;

using System.Linq;

namespace NewLSP.StaticHelperClasses

{

public static class StringHelper

{

#region ReturnNumberOfDeliniters

public static int ReturnNumberOfDeliniters(string line, char del)

{

char[] LineCharArray = line.ToCharArray();

int count = 0;

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

{

char thisChar = LineCharArray[i];

if (thisChar == del) count++;

}

return count;

}

#endregion ReturnNumberOfDeliniters

#region ReturnItemAtPos

public static string ReturnItemAtPos(string delString, char del, int Pos)

{

string[] itemsArray = delString.Split(del);

return itemsArray[Pos];

}

#endregion ReturnItemAtPos

#region ReturnLastItem

public static string ReturnLastItem(string line, char del)

{

var PosLastDel = line.LastIndexOf(del);

var ReturnString = line.Substring(PosLastDel + 1);

return ReturnString;

}

#endregion ReturnLastItem

#region ReturnLastAndUpdate

public static string[] ReturnLastAndUpdate(string delString, char del)

{

var returnArr = new string[2];

int posLastDelimiter = delString.LastIndexOf(del);

returnArr[0] = delString.Substring(posLastDelimiter);

returnArr[1] = delString.Substring(0, posLastDelimiter - 1);

return returnArr;

}

#endregion ReturnLastAndUpdate

#region RemoveLastValue

public static string RemoveLastValue(string delString, char del)

{

int posLastDelimiter = delString.LastIndexOf(del);

if (posLastDelimiter < 0)

{

return "";

}

string updatedString = delString.Substring(0, posLastDelimiter);

return updatedString;

}

#endregion RemoveLastValue

#region CreateDisplayString

/// <summary>

/// Create a display string to show in a list box

/// </summary>

/// <param name="LeedingChar"> is a + or - </param>

/// <param name="Text"></param>

/// <param name="ID"> the Items AlphaNumber</param>

/// <param name="NumberOfChildren"> The Items number of Children</param>

/// <returns></returns>

public static string CreateDisplayString(char LeedingChar, string Text, string ID, int NumberOfChildren)

{

string thisItemsListString;

int LengthOFItemText = Text.Length;

int addSpacesNumber = 100 - LengthOFItemText;

string spacesString = new string(' ', addSpacesNumber);

if (LeedingChar == '-')

{

thisItemsListString = "- " + Text + spacesString + '^' + ID + '^' + NumberOfChildren.ToString();

}

else

{

thisItemsListString = "+ " + Text + spacesString + '^' + ID + '^' + NumberOfChildren.ToString();

}

return thisItemsListString;

}

#endregion CreateDisplayString

#region ReplaceItemAtPosition

public static void ReplaceItemAtPosition(ref string line, char del, int postions, string item)

{

string[] LineArray = line.Split(del);

LineArray[postions] = item;

line = "";

foreach (string Item in LineArray)

{

line = line + Item + del;

}

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

}

#endregion ReplaceItemAtPosition

#region GetAndRemoveNthItem

public static string GetAndRemoveNthItem(ref string delimitedString, char del, int pos)

{

string[] itemsArray = delimitedString.Split(del);

string returnItem = itemsArray[pos];

List<string> ItemsList = itemsArray.ToList<string>();

ItemsList.RemoveAt(pos);

delimitedString = "";

foreach(string item in ItemsList)

{

delimitedString = delimitedString + item + del;

}

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

return returnItem;

}//End GetAndRemoveNthItem

#endregion GetAndRemoveNthItem

#region RemoveFirstItem

public static string RemoveFirstItem(string delString, char del)

{

int posFirstDel = delString.IndexOf(del);

delString = delString.Substring(posFirstDel + 1);

return delString;

}

#endregion RemoveFirstItem

}// End Class

}// End Namespace

SHMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **StringHelpers.cs** | | | | |
| ReturnNumberOfDeliniters( |  | int | [Home.xaml.cs](http://Home.xaml.cs) –  btnOpenSubjectFolder\_Click(; |  |
| string ReturnItemAtPos( |  | string | 15 references |  |
| string ReturnLastItem( |  | string | NOT CALLED |  |
| string[] ReturnLastAndUpdate( |  | string | NOT CALLED |  |
| string RemoveLastValue( |  | string | NOT CALLED |  |
| string CreateDisplayString( |  | string | NOT CALLED |  |
| void ReplaceItemAtPosition( |  | void | NOT CALLED |  |
| string GetAndRemoveNthItem( |  | string | NOT CALLED |  |
| string RemoveFirstItem( |  | string | TestReview.xaml.cs –  TestOrReview(; |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

SHGlobalsTable

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | CalledBy | Purpose |
| **StringHelpers.cs** | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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

SubjectStaticMembers

[SSMMethodsTable](#SSMMethodsTable)

[SSMGlobalsTable](#SSMGlobalsTable)

using NewLSP.DataModels;

using System;

using System.Collections.Generic;

using System.IO;

namespace NewLSP.StaticHelperClasses

{

public static class SubjectStaticMembers

{

#region Properties

#region Property SelectedNode

/// <summary>

/// The selected node is the subject node in the ListView which the user

/// clicked

/// </summary>

private static SubjectNodes \_SelectedNode;

public static SubjectNodes SelectedNode

{

get { return \_SelectedNode; }

set { \_SelectedNode = value; }

}// End SubjectNodes

#endregion SelectedNode

#region Property ItemCount

/// <summary>

/// ItemCount is the number of items created,

/// Not the number of items present because

/// some may have been deleted

/// It is used to tie a subject to

/// various data files

/// </summary>

private static string \_ItemCount;

public static string ItemCount

{

get { return \_ItemCount; }

set { \_ItemCount = value; }

}

#endregion ItemCount

#region Propety Data Node Selected

private static SubjectNodes \_DataNode;

public static SubjectNodes DataNode

{

get { return \_DataNode; }

set

{

\_DataNode = value;

}

}

#endregion Data Node Selected

#region Property NoteReferenceFilesPath

#endregion Property NoteReferenceFilesPath

#region Property CompositDataPath

private static string \_CompositDataPath;

/// <summary>

/// Holds the path to the folder that will contain the

/// KeyWords Dictionary.txt, and a Timelin file,

/// </summary>

public static string CompositDataPath

{

get { return \_CompositDataPath; }

set { \_CompositDataPath = value; }

}

#endregion Property CompositDataPath

#region Property NotesDictionaryPath

private static string \_KeyWordsDictionaryPath;

public static string KeyWordsDictionaryPath

{

get { return \_KeyWordsDictionaryPath; }

set { \_KeyWordsDictionaryPath = value; }

}

#endregion Property NotesDictionaryPath

#endregion Properties

#region Public Fields

#region Field Dictionary of Subject Nodes (SubjectNodeDictionary)

// Create a dictionary of all subject nodes whose key is the node.NodeLevelName

public static Dictionary<string, SubjectNodes> SubjectNodeDictionary = new Dictionary<string, SubjectNodes>();

#endregion SubjectNodeDictionary

#region Field List of Display strings (DisplayList)

//Create an List of strings for the ListView display and array of SubjectNodes to match it

/// <summary>

/// The DisplayList is what shows up in the Subjects Tree, Each item has only 3 components

/// 1. The leading blanks to indicate the Item level

/// 2. The Has Chidred indication (+/-)

/// 3. The Name yoy assigned to the node

/// </summary>

public static List<string> DisplayList = new List<string>();

#endregion DisplayList

#region Field List of Subject NodeLevelName strings ListView display (SubjectNodesLevelName)

// Create a List of SubjectNode NodeLevelName strings to match DisplayList

public static List<string> SubjectNodesLevelNameList = new List<string>();

#endregion SubjectNodesLevelNam

#region Field OldNLN

public static string OldNLN;

#endregion Field OldNLN

#endregion Public Fields

#region Private Fields

private static string ItemsCountFilePath;

private static string SubjectName;

// This is the path to the folder that holds the 'ɀ' delimited tree nodes in the selected Subject tree

private static string SubjectsNodeDataStringsPath;

// This is the path to the Folder that holds the questions and answers for a given subject tree node

public static string DataNodesQAFilePath;

#endregion Private Fields

#region Public Methods

#region Open Data Files method (OpenFiles)

/// <summary>

/// This method receives the path to the main holding folder

/// for a particular subject and opens add of the

/// reqired data files to populate the ListView display

/// of Subjects. The Name of the main holding folder

/// is the display name of the Subects as well

/// as of the .txt file that holds the display data

/// References:

/// 1. Home.xaml.cs btnOpenSubjectFolder\_Click

/// </summary>

/// <param name="HomeFolderPat"></param>

public static void OpenFiles(string ThisHomeFolderPath)

{

// Set DataNode Hyperlinks Path

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

// Set the Path to the DataNodesNoteReferencesFilesPath

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

//// Create path to this subjects data file

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

//string SubjectNodesDataFilePath = SubjectsNodeDataStringsPath;

SubjectNodes RootNode = new SubjectNodes();

// Read in the current item count

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

SubjectStaticMembers.ItemCount = ItemCount;

// Instantiate the Dictionary

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

// Instantiate a new node

SubjectNodes ThisNode = new SubjectNodes();

// Create the delimiter

char D = '\u0240';

//Read in SubjectsNodeDataStringsPath into a string array

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

//Create the SubjectNodeDictionary

// Parse each line in the SubjectNodeDataStringArray to add a dictionary item

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;

}

SubjectNodeDictionary.Add(ThisNode.NodeLevelName, ThisNode);

ThisNode = new SubjectNodes();

}// End foreach

DisplayParentsAndChildren("\*");

}//End OpenFiles method

#endregion OpenFiles

#region GetLeadingChars

public static string GetLeadingChars(string nodeLevelName)

{

int LengthOfNodeLevelName = nodeLevelName.Length - 1;

return new string(' ', LengthOfNodeLevelName \* 3);

}// End GetLeadingChars

#endregion GetLeadingChars

#region GetNodeLevelPosition

public static string GetNodeLevelPosition(int ParentsNOC)

{

string NodeLevelPositionString = "0123456789abcedfghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

char[] NodeLevelPositionChars = NodeLevelPositionString.ToCharArray();

char NodeLevelChar = NodeLevelPositionChars[ParentsNOC];

return NodeLevelChar.ToString();

}// End GetNodeLevelPosition

#endregion GetNodeLevelPosition

#region Remove Node From Dictionary

public static void RemoveNodeFromDictionary(string thisNodesLevelName)

{

SubjectNodeDictionary.Remove(thisNodesLevelName);

}// EndRemove Node From Dictionary

#endregion Remove Node From Dictionary

public static void ReplaceNode(string thisNodesLevelName, SubjectNodes newNode)

{

SubjectNodeDictionary[thisNodesLevelName] = newNode;

}

#region SaveFiles

public static void SaveFiles()

{

// Create a List<string> OutputNodeDataStringList adding delimited node properties to the List

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

// Cycle thorugh SubjectNodeDictionary adding delimited node properties to the List

foreach (KeyValuePair<string, SubjectNodes> KVP in SubjectNodeDictionary)

{

string Key = KVP.Key;

SubjectNodes ThisNode = KVP.Value;

string SubjectNodeDelimitedString = RetrunNodeDelimitedString(ThisNode);

OutputNodeDataStringList.Add(SubjectNodeDelimitedString);

}

// Create a string array of OutputNodeDataStringList

string[] OutputNodeDataStringArray = OutputNodeDataStringList.ToArray();

// Save OutputNodeDataStringList to SubjectsNodeDataStringsPath.txt

File.WriteAllLines(CommonStaticMembers.HomeFolderPath + "NodeDataStrings.txt", OutputNodeDataStringArray);

// Save the CurrentItemCount

string ItemCount = SubjectStaticMembers.ItemCount;

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

}// End SaveFiles

#endregion SaveFiles

#region Display a node's parents, the node and the nodes children (DisplayParentsAndChildren)

/// <summary>

/// This method creates the List of display strings for the ListView

/// as well as a dictionary of Subject nodes whose

/// </summary>

/// <param name="ThisNode"></param>

/// <returns></returns>

public static void SaveFiles()string ThisNodesLevelName)

{

// Clear the existing List

DisplayList.Clear();

SubjectNodesLevelNameList.Clear();

SubjectNodes CurrentNode = new SubjectNodes();

// Display Parents and chosen node

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

{

// Increase the length of the NodeLevelName by 1 on each iteration

string CurrentNodeLevelName = ThisNodesLevelName.Substring(0, i + 1);

if (SubjectNodeDictionary.ContainsKey(CurrentNodeLevelName))

{

CurrentNode = SubjectNodeDictionary[CurrentNodeLevelName];

}

string ThisNodesDisplayString = ReturnDisplayString(CurrentNode);

DisplayList.Add(ThisNodesDisplayString);

SubjectNodesLevelNameList.Add(CurrentNodeLevelName);

}

// Get the children of the chosen node

// Get the Length of the choden nodes node level name because its children NLN will be 1 longer

int L = ThisNodesLevelName.Length;

// begins witn ThisNodesLevelName and whose length is 1 greater that that of ThisNodesLevelName

foreach (KeyValuePair<string, SubjectNodes> KVP in SubjectNodeDictionary)

{

string Key = KVP.Key;

SubjectNodes ThisNode = SubjectNodeDictionary[Key];

if ((Key.IndexOf(ThisNodesLevelName) == 0))

{

if (ThisNode.NodeLevelName.Length == L + 1)

{

string ThisDisplaystring = ReturnDisplayString(ThisNode);

DisplayList.Add(ThisDisplaystring);

string ThisNodeLevelName = ThisNode.NodeLevelName;

SubjectNodesLevelNameList.Add(ThisNodeLevelName);

}

}

}

return DisplayList;

}// End DisplayParentsAndChildren method

#endregion DisplayParentsAndChildren

#region Read in the count of items existing at startup (GetCurrentItemCount)

public static int GetCurrentItemCount()

{

//ItemCount = -1;

if (File.Exists(ItemsCountFilePath))

{

//BinaryReader binReader = new BinaryReader(File.Open(ItemsCountFilePath, FileMode.Open));

//ItemCount = binReader.ReadInt32();

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

}

return Int32.Parse(ItemCount);

}// End GetCurrentItemCount

#endregion (GetCurrentItemCount)

#region AddNodeToDictionary

public static void AddNodeToDictionary(SubjectNodes ThisNode)

{

SubjectNodeDictionary.Add(ThisNode.NodeLevelName, ThisNode);

}

#endregion AddNodeToDictionary

#region SetDataNodesQAFilePath

/// <summary>

/// Called by STxc .rbDataNode\_Click

/// which then sets the DataNodesQAFilePath to the

/// QAFiles folder's DataNode's ID.txt file

/// </summary>

public static void SetDataNodesQAFilePath()

{

//

DataNodesQAFilePath = CommonStaticMembers.SubjectFolderPath + "QAFiles\\" + DataNode.ID.ToString() + ".txt";

}

public static string GetDataNodesQAFilePath()

{

return DataNodesQAFilePath;

}

#endregion SetDataNodesQAFilePath

#region GetSetDataNodeNoteReferenceFilePath

static string DataNodeNoteReferenceFilePath;

public static void SetDataNodeNoteReferenceFilePath()

{

SubjectNodes DateNode = new SubjectNodes();

DateNode = SubjectStaticMembers.DataNode;

int DataNodeId = DataNode.ID;

DataNodeNoteReferenceFilePath = CommonStaticMembers.HomeFolderPath + "DataNodesNoteReferencesFiles\\" + DataNodeId.ToString() + ".txt";

}

public static string GetDataNodeNoteReferenceFilePath()

{

return DataNodeNoteReferenceFilePath;

}

#endregion GetSetDataNodeNoteReferenceFilePath

public static SubjectNodes ReturnNodeAtPos(string NodeLevelName)

{

return SubjectNodeDictionary[NodeLevelName];

}

#region Public Method to change all nodes with an old NLN to a new NLN: ChangeMovedNodesNLN()

internal static void ChangeMovedNodesNLN(string oldNLN, string newNLN)

{

int currentCountOfDictionary = SubjectNodeDictionary.Count;

// Create an array of SubjectNodes to change the values

SubjectNodes[] ArrayOfSubjectNodes = new SubjectNodes[currentCountOfDictionary];

//Cycle through the dictionary adding each node to the array

int cntr = 0;

foreach(KeyValuePair<string, SubjectNodes> KVP in SubjectNodeDictionary)

{

string key = KVP.Key;

SubjectNodes thisNode = KVP.Value;

ArrayOfSubjectNodes[cntr] = thisNode;

cntr++;

}

//Cycle through the ArrayOfSubjectNodes add nodes to be moved

foreach (SubjectNodes thisNode in ArrayOfSubjectNodes)

{

string NodeLevelName = thisNode.NodeLevelName;

//Find all nodes that beign with OldNLN

if (NodeLevelName.IndexOf(oldNLN) == 0)

{

// Get the currnt NLN

string thisNodeLevelName = thisNode.NodeLevelName;

// Repalcde it with newNLN

thisNodeLevelName = thisNodeLevelName.Replace(oldNLN, newNLN);

// Change the NLN in the node

thisNode.NodeLevelName = thisNodeLevelName;

// Adjust the leading chars

int LengthNodeLevelName = thisNode.NodeLevelName.Length\*3;

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

thisNode.LeadingChars = LeadingChars;

}

}// End //Cycle through the ArrayOfSubjectNodes add nodes to be moved

// Instantiate a new dictionary

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

// Cycle through ArrayOfSubjectNodes adding them to SubjectNodeDictionary

foreach (SubjectNodes thisNode in ArrayOfSubjectNodes)

{

string Key = thisNode.NodeLevelName;

// Add thisNode to the dictionary

SubjectNodeDictionary.Add(Key, thisNode);

}

}// End ChangeMovedNodesNLN

#endregion ChangeMovedNodesNLN()

#region public method NodeHasQAFile

internal static bool NodeHasQAFile(int nodeID)

{

string QAFilePath = CommonStaticMembers.HomeFolderPath + "QAFiles\\" + nodeID.ToString() + ".txt";

if (File.Exists(QAFilePath))

{

return true;

}

else

{

return false;

}

}//End NodeHasQAFile

#endregion public method NodeHasQAFile

#region public method NodeHadDataFile

internal static bool NodeHasHyperlinksFile(int nodeID)

{

string DataFilePath = CommonStaticMembers.DataNodesHyperlinksPath + "\\" + nodeID.ToString() + ".txt";

if (File.Exists(DataFilePath))

{

return true;

}

else

{

return false;

}

}

#endregion public method NodeHadDataFi

#endregion Public Methods

#region Priate Methods

#region Retrun the display string for a node (ReturnDisplayString)

private static string ReturnDisplayString(SubjectNodes ThisNode)

{

string DisplayString = "";

string LeadingString = ThisNode.LeadingChars;

string ChildIndicator = ThisNode.CI;

string NodeText = ThisNode.TitleText;

DisplayString = LeadingString + ChildIndicator + NodeText;

return DisplayString;

}// End ReturnDisplayString

#endregion (ReturnDisplayString)

#region Return a delimited string of the items in a SubjectNode (RetrunNodeDelimitedString)

private static string RetrunNodeDelimitedString(SubjectNodes thisNode)

{

string LeadingCharsString = thisNode.LeadingChars;

string TitleText = thisNode.TitleText;

int ItemIDInt = thisNode.ID;

string ItemIDString = ItemIDInt.ToString();

int ItemsNumberOfChildren = thisNode.NOC;

string ItemsNumberOfChildrenString = ItemsNumberOfChildren.ToString();

string ItemsChildrenIncidator = thisNode.CI;

string ItemsNodeLevelName = thisNode.NodeLevelName;

bool ItemHasData = thisNode.HasData;

string ItemHasDataString;

if (ItemHasData == false)

{

ItemHasDataString = "false";

}

else

{

ItemHasDataString = "true";

}

char D = '\u0240';

string OutputString = LeadingCharsString + D + ItemsChildrenIncidator + D + TitleText +

D + ItemsNodeLevelName + D + ItemIDString + D + ItemsNumberOfChildrenString + D + ItemHasDataString;

return OutputString;

}// End RetrunNodeDelimitedString

#endregion RetrunNodeDelimitedString

#endregion Priate Methods

}// End SubjectStaticMembers

}// End NameSpace

SSMMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **SubjectStaticMembers.cs** | | | | |
| [OpenFiles](#SSMOpenFilesDef)( |  | void |  | This method receives the path to the main holding folder for a particular subject and opens add of the reqired data files to populate the ListView display of Subjects. The Name of the main holding folder is the display name of the Subects as well as of the .txt file that holds the display data References: 1. Home.xaml.cs btnOpenSubjectFolder\_Click |
| [GetLeadingChars](#SSMGetLeadingCharsDef)( |  | string |  | NOTE: COMMENT Returns a string of blanks where every 3 spaces represent an increase in the node level |
| [GetNodeLevelPosition](#SSMGetNodeLevelPositionDef)( |  | string |  | NOTE: COMMENT ???? |
| [RemoveNodeFromDictionary](#SSMRemoveNodeFromDictionaryDef)( |  | void |  | NOTE: COMMENT Apparently removes a node from the dictionary |
| [ReplaceNode](#SSMReplaceNodeDef)( |  | void |  | Replaces the node at SubjectNodeDictionary[thisNodesLevelName] with the newNode; |
| [SaveFiles](#SSMSaveFilesDef)() |  | void |  |  |
| [GetCurrentItemCount](#GetCurrentItemCount)() |  | int |  |  |
| [AddNodeToDictionary](#SSMAddNodeToDictionaryDef)( |  | void |  |  |
| [SetDataNodesQAFilePath](#SSMSetDataNodesQAFilePathDef)( |  | void |  |  |
| [GetDataNodesQAFilePath](#SSMGetDataNodesQAFilePathDef)() |  | string |  |  |
| [SetDataNodeNoteReferenceFilePath](#SSMSetDataNodeNoteReferenceFilePathDef)() |  | void |  |  |
| [GetDataNodeNoteReferenceFilePath](#SSMGetDataNodeNoteReferenceFilePathDef)( |  | string |  |  |
| [ReturnNodeAtPos](#SSMReturnNodeAtPosDef)( |  | SubjectNodes |  |  |
| [ChangeMovedNodesNLN](#SSMChangeMovedNodesNLNDef)( |  | void |  |  |
| [NodeHasQAFile](#SSMNodeHasQAFileDef)( |  | bool |  |  |
| [NodeHasHyperlinksFile](#SSMNodeHasHyperlinksFileDef)( |  | bool |  |  |
| [ReturnDisplayString](#SSMReturnDisplayStringDef)( |  | string |  |  |
| [RetrunNodeDelimitedString](#SSMRetrunNodeDelimitedStringDef)( |  | string |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

SSMGlobalsTable

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | CalledBy | Purpose |
| **SubjectStaticMembers.cs** | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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

TestReviewStaticMembers

[TRSMMethodsTable](#TRSMMethodsTable)

[TRSMGlobalsTable](#TRSMGlobalsTable)

TRSMMethodsTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Access | Return | CalledBy | Purpose |
| **TestReviewStaticMembers.cs** | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

TRSMGlobalsTable

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | CalledBy | Purpose |
| **TestReviewStaticMembers.cs** | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |