I could use your assistance to help me tackle the following assignment. I am developing a Windows Forms (WinForms) in .NET 6 that would allow me to display and search data. It has three TreeView controls and three search boxes positioned on the left of the form, one search box per TreeView control right above it and one ListView control on the right of the form according to the following specifications.

Each TreeView shall have nodes.

Each node may have children nodes.

Each node, regardless whether it is a parent node or a child node, has a checkbox.

Each node has a text field that stores the node's name and two numeric fields that store the node's unique ID number and a unique Handle.

The nodes in the TreeView controls are filled out with data from three respective text .cvs files located in the logs folder, namely logs\\_lst\_LogData\_elm.cvs, logs\\_lst\_LogData\_chn.cvs, logs\\_lst\_LogData\_dbs.cvs.

For example, given the following hierarchy, which is read from a text file line by line where the text represents the node's name and it is respective location in the family hierarchy followed by its ID and Handle numerical values separated by the @ delimiting character.

grandparent1@1@1

grandparent1/parent1@2@2

grandparent1/parent1 / child3@3@3

grandparent2@4@4

grandparent2/parent1@5@5

grandparent2/parent1/child1@6@6

grandparent2/parent1/child2@7@7

grandgrandparent3@8@8

grandgrandparent3/grandparent1@9@9

grandgrandparent3/grandparent1/parent1@10@10

grandgrandparent3/grandparent1/parent1/child1@11@11

I need to have the following representation in the TreeView control:

grandparent1, 1, 1

- parent1, 2, 2

- child3, 3, 3

grandparent2, 4, 4

- parent1, 5, 5

- child1, 6, 6

- child2, 7, 7

grandgrandparent3, 8, 8

- grandparent1, 9, 9

- parent1, 10, 10

- child1, 11, 11

where the ID and Handle numerical values must be part of the TreeNode object for later retrieval and display in the respective columns in the ListView. The ID and Handle need not be displayed in the TreeView and are shown here for the sake of the clarity.

The file read operation and the filling out of the respective TreeView controls with the read data shall be done in parallel, which is currently not implemented in my C++/CLI code that I am supplying you with below.

Once a node’s checkbox is checked its data are displayed in the ListView in three columns, namely Long Name (node’s text), ID and Handle. Once a node’s checkbox is unchecked, the respective node’s data is removed from the ListView. This functionality is not implemented yet in my C++/CLI code.

Here is my new ElementSearch solution and project files without any underlying business logic that may assist you:

ElemenSearchApp.sln file top few lines:

Microsoft Visual Studio Solution File, Format Version 12.00

# Visual Studio Version 17

VisualStudioVersion = 17.5.33502.453

MinimumVisualStudioVersion = 10.0.40219.1

ElementSeacrh.Designer.cs file contents without comments:

namespace ElementSearch

{

partial class FormElementSearch

{

private System.ComponentModel.IContainer components = null;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

private void InitializeComponent()

{

textBoxElemType = new TextBox();

treeViewElemType = new TreeView();

treeViewChannel = new TreeView();

treeViewDatabase = new TreeView();

listViewElements = new ListView();

columnHeaderLongName = new ColumnHeader();

columnHeaderShortName = new ColumnHeader();

columnHeaderElemType = new ColumnHeader();

columnHeaderChannel = new ColumnHeader();

columnHeaderDatabase = new ColumnHeader();

columnHeaderLocation = new ColumnHeader();

columnHeaderID = new ColumnHeader();

columnHeaderHandle = new ColumnHeader();

buttonSearch = new Button();

labelElemType = new Label();

labelChannel = new Label();

textBoxChannel = new TextBox();

labelDatabase = new Label();

textBoxDatabase = new TextBox();

SuspendLayout();

textBoxElemType.Name = "textBoxElemType";

treeViewElemType.Name = "treeViewElemType";

treeViewChannel.Name = "treeViewChannel";

treeViewDatabase.Name = "treeViewDatabase";

listViewElements.Columns.AddRange(new ColumnHeader[] { columnHeaderLongName, columnHeaderShortName, columnHeaderElemType, columnHeaderChannel, columnHeaderDatabase, columnHeaderLocation, columnHeaderID, columnHeaderHandle });

listViewElements.Name = "listViewElements";

columnHeaderLongName.Text = "Long Name";

columnHeaderShortName.Text = "Short Name";

columnHeaderElemType.Text = "Elem Type";

columnHeaderChannel.Text = "Channel";

columnHeaderDatabase.Text = "Database";

columnHeaderLocation.Text = "Location";

columnHeaderID.Text = "ID";

columnHeaderHandle.Text = "Handle";

buttonSearch.Name = "buttonSearch";

labelElemType.Name = "labelElemType";

labelChannel.Name = "labelChannel";

textBoxChannel.Name = "textBoxChannel";

labelDatabase.Name = "labelDatabase";

textBoxDatabase.Name = "textBoxDatabase";

Name = "FormElementSearch";

Text = "Form1";

ResumeLayout(false);

PerformLayout();

}

#endregion

private TextBox textBoxElemType;

private TreeView treeViewElemType;

private TreeView treeViewChannel;

private TreeView treeViewDatabase;

private ListView listViewElements;

private Button buttonSearch;

private Label labelElemType;

private Label labelChannel;

private TextBox textBoxChannel;

private Label labelDatabase;

private TextBox textBoxDatabase;

private ColumnHeader columnHeaderLongName;

private ColumnHeader columnHeaderShortName;

private ColumnHeader columnHeaderElemType;

private ColumnHeader columnHeaderChannel;

private ColumnHeader columnHeaderDatabase;

private ColumnHeader columnHeaderLocation;

private ColumnHeader columnHeaderID;

private ColumnHeader columnHeaderHandle;

}

}

Here is my C++/CLI code that I am asking you to implement, expand to include all the necessary functionality, improve and optimize, if possible, using modern C#.

MyTreeNode.h file contents:

using namespace System;

// Define a custom TreeNode class that derives from System::Windows::Forms::TreeNode

ref class MyTreeNode : public System::Windows::Forms::TreeNode

{

public:

// Custom data members

unsigned int m\_id;

unsigned int m\_handle;

// Default constructor

MyTreeNode() : TreeNode()

{

this->m\_id = 0;

this->m\_handle = 0;

}

// Constructor with text ONLY

MyTreeNode(String^ text) : TreeNode(text)

{

// Initialize custom data members to default values

this->m\_id = 0;

this->m\_handle = 0;

}

// Constructor with name, ID and handle parameters (name is not used)

MyTreeNode(String^ text, unsigned int id, unsigned int handle) : TreeNode(text)

{

this->m\_id = id;

this->m\_handle = handle;

}

// Constructor with text and children parameters

MyTreeNode(String^ text, array<MyTreeNode^>^ children) : TreeNode(text)

{

// Initialize custom data members to default values

this->m\_id = 0;

this->m\_handle = 0;

// Add children to node

this->Nodes->AddRange(children);

}

// Copy constructor

MyTreeNode(MyTreeNode^ other) : TreeNode(other->Text)

{

// Copy any custom data members

this->m\_id = other->m\_id;

this->m\_handle = other->m\_handle;

this->Checked = other->Checked;

}

};

MainForm.cpp file contents:

#include "MainForm.h"

using namespace System;

using namespace System::Windows::Forms;

using namespace System::Collections::Generic;

[STAThreadAttribute]

int main(array<String^>^ args)

{

Application::SetCompatibleTextRenderingDefault(false);

Application::EnableVisualStyles();

ElementSearch::MainForm frm;

Application::Run(% frm);

}

// This function loads the main form

System::Void ElementSearch::MainForm::MainForm\_Load(System::Object^ sender, System::EventArgs^ e)

{

// Add some sample data to ListView

//ListViewItem^ row\_item = gcnew ListViewItem(gcnew array<String^> { L"Long Name 1", L"Short Name 1",

// L"Elem Type 1", L"Channel 1", L"Database 1", L"Location 1", L"ID 1", L"Handle 1" });

// Define vectors to store vectors of tokens for each data file

std::vector<std::vector<std::string>> file\_tokens\_elm, file\_tokens\_chn, file\_tokens\_dbs;

file\_tokens\_elm.reserve(8);

file\_tokens\_chn.reserve(2000);

file\_tokens\_dbs.reserve(950);

std::vector<std::vector<std::vector<std::string>>> file\_tokens = { file\_tokens\_elm, file\_tokens\_chn, file\_tokens\_dbs };

// Define file paths

std::vector<std::string> files\_paths = { "logs\\\_lst\_LogData\_elm.cvs", "logs\\\_lst\_LogData\_chn.cvs", "logs\\\_lst\_LogData\_dbs.cvs" };

// Read data from text files

ReadTextFiles(files\_paths, file\_tokens);

}

// This function reads text files

System::Void ElementSearch::MainForm::ReadTextFiles(const std::vector<std::string>& file\_paths, std::vector<std::vector<std::vector<std::string>>>& file\_tokens)

{

ReadTextFile(file\_paths[0], file\_tokens[0]);

ReadTextFile(file\_paths[1], file\_tokens[1]);

ReadTextFile(file\_paths[2], file\_tokens[2]);

FillTreeView(this->treeViewElement, file\_tokens[0]);

FillTreeView(this->treeViewChannel, file\_tokens[1]);

FillTreeView(this->treeViewDatabase, file\_tokens[2]);

}

System::Void ElementSearch::MainForm::ReadTextFile(const std::string file\_path, std::vector<std::vector<std::string>>& file\_tokens)

{

// Open a file in read mode

std::ifstream infile;

// std::istringstream ss(msclr::interop::marshal\_as<std::string>(file\_path));

std::istringstream ss(file\_path);

infile.open(ss.str());

// Check whether the file is open

if (infile.is\_open())

{

std::string line, token;

// Define vector to store tokens for each line

std::vector<std::string> line\_tokens;

line\_tokens.reserve(3);

// Read text lines from file one line at a time

while (getline(infile, line))

{

std::istringstream ss(line);

// Tokenize line

while (std::getline(ss, token, '@'))

{

if (token.empty() == false)

{

// Store line tokens

line\_tokens.push\_back(token);

}

}

// Store file tokens

file\_tokens.push\_back(line\_tokens);

line\_tokens.clear();

line\_tokens.reserve(3);

}

infile.close(); // Close file

}

}

// This function fills out TreeView control with data from vector of strings (previously extracted from the text files)

System::Void ElementSearch::MainForm::FillTreeView(TreeView^ tree\_view, std::vector<std::vector<std::string>>& file\_tokens)

{

// Disable redrawing of TreeView while adding nodes

tree\_view->BeginUpdate();

// Clear any existing nodes

tree\_view->Nodes->Clear();

// Traverse file tokens

for each (std::vector<std::string> line\_tokens in file\_tokens)

{

if (line\_tokens[0].empty() == false)

{

// Store parent/children description

std::istringstream ss(line\_tokens[0]);

int id = 0, handle = 0;

if (line\_tokens[1].empty() == false)

{

// Store node ID

id = std::stoi(line\_tokens[1]);

}

if (line\_tokens[2].empty() == false)

{

// Store node Handle

handle = std::stoi(line\_tokens[2]);

}

List<MyTreeNode^>^ family = gcnew List<MyTreeNode^>();

// Tokenize hierarchy of family relatives

std::string relative;

while (getline(ss, relative, '/'))

{

family->Add(gcnew MyTreeNode(gcnew System::String(relative.c\_str()), id, handle));

}

AddNode(tree\_view->Nodes, family, 0);

}

}

// Enable redrawing of TreeView

tree\_view->EndUpdate();

}

void ElementSearch::MainForm::AddNode(TreeNodeCollection^ nodes, List<MyTreeNode^>^ family, int index)

{

MyTreeNode^ current = family[index];

// Find existing node with matching text, or create new node

MyTreeNode^ node = nullptr;

for each (MyTreeNode^ child in nodes)

{

if (child->Text == current->Text)

{

node = child;

break;

}

}

if (node == nullptr)

{

node = gcnew MyTreeNode(current->Text, current->m\_id, current->m\_handle);

nodes->Add(node);

}

// Recursively add child nodes

if (index < family->Count - 1)

{

AddNode(node->Nodes, family, index + 1);

}

}

Can you please re-implement the above solution in a most efficient, possibly using multithreading/multitasking approach, and in a most elegant manner, possibly using advanced C# language intrinsic constructs and tools and provide me with the code that loads the form, reads text files, fills out TreeView controls? That is, the application needs to read three text files and fill out three TreeView controls with data from the files. For your convenience here are two sets of method declarations, both worked, used in previous implementation scenarios:

Set1:

private void FormElementSearch\_Load(object sender, EventArgs e);

private List<List<string>> ReadTextFile(string filePath);

private void FillTreeView(TreeView treeView, IReadOnlyList<List<string>> fileTokens);

private void AddNode(TreeNodeCollection nodes, List<MyTreeNode> family, int index);

Set2:

private async void FormElementSearch\_Load2(object sender, EventArgs e);

private async Task ReadTextFile2(string filePath, List<List<string>> fileTokens);

private void FillTreeView2(TreeView treeView, List<List<string>> fileTokens);

private void AddNode(TreeNodeCollection nodes, List<MyTreeNode> family, int index);

In my application there are so far three TreeViews and one ListView control. Namely:

private TreeView treeViewElemType;

private TreeView treeViewChannel;

private TreeView treeViewDatabase;

private ListView listViewElements;

In the TreeList control there are three columns, namely:

listViewElements.Columns.AddRange(new ColumnHeader[] {

columnHeaderElemType, columnHeaderChannel, columnHeaderDatabase });

columnHeaderElemType.Text = "Element";

columnHeaderChannel.Text = "Channel";

columnHeaderDatabase.Text = "Database";

Once a node in a TreeView control is checked the node’s name/text shall be displayed in the respective column in ListView. Once unchecked, it shall be removed from ListView.

Can you please assist me implementing this logic in computationally optimal manner?

The following line: while ((line = await reader.ReadLineAsync()) != null)

generates the following warning: CS 8600: Converting null literal or possible nul value to non-nullable type

The following line: UpdateParentNode(node.Parent as MyTreeNode); generates the following warning: CS8604: Possible null reference argument for parameter 'node'

I get the following runtime error: System.ArgumentOutOfRangeException:

'Specified argument was out of the range of valid values. (Parameter 'index')' on line: newItem.SubItems.Insert(columnIndex, new ListViewItem.ListViewSubItem(newItem, node.Text)); in the AddCheckedNodesToListView method:

private void AddCheckedNodesToListView(IEnumerable<TreeNode> checkedNodes, int columnIndex)

{

foreach (var node in checkedNodes)

{

var newItem = new ListViewItem();

if (columnIndex == 0)

{

newItem.Text = node.Text;

}

else

{

newItem.Text = "";

newItem.SubItems.Insert(columnIndex, new ListViewItem.ListViewSubItem(newItem, node.Text));

}

listViewElements.Items.Add(newItem);

}

}

when I try to mark the nodes checked ONLY in the treeViewDatabase TreeView control. The other two controls work so far. Any ideas why?

UpdateParentNode(node.Parent as MyTreeNode);

CS8604: Possible null reference argument for parameter ‘node’ in void FormElementSearch.UpdateParentNode(MyTreeNode node)’

treeView.AfterCheck -= HandleTreeViewAfterCheck;

CS8622: Nullability of reference types in type of parameter 'sender' of 'void FormElementSearch.HandleTreeViewAfterCheck(object sender, TreeViewEventArgs e)' doesn't match the target delegate 'TreeViewEventHandler' (possibly because of nullability attributes)

My TreeView control has the following 8 columns arranged in that particular order:

ID, Long Name, Short Name, Elem Type, Channel, Database, Location, Handle.

The sample contents of the text file titled: lst\_LogData\_elm\_all.cvs that stores the necessary data:

1@grandparent1/parent1/child1@child1@type1@channel1@database1@location1@1234567

2@grandparent1/parent1/child2@child2@type1@channel2@database2@location2@1234568

3@grandparent1/parent1/child3@child3@type1@channel3@database3@location3@1234569

Can you please modify the implementation of the FormElementSearch\_Load(object sender, EventArgs e) function to read data from this text file and fill out the ListView with the read data?

Would you like to take a look at the entire working code now before we proceed to further develop and enhance it?

It is functioning as expected. Still I was wondering if you could suggest any further optimizations with respect to time and/or space complexities?

**Answer:**

I am glad that the code is functioning as expected. The current implementation is efficient for the scope of this project, but there are a few possible optimizations that you can consider:

1. Use **Dictionary<string, MyTreeNode>** for looking up existing tree nodes: Currently, the **AddNode** method uses a linear search to find an existing node with the same text. You can use a dictionary to speed up this process. This will be especially useful if you have a large number of nodes in your TreeViews.
2. Use **HashSet<uint>** to track checked element IDs in **UpdateListView**: Instead of creating a **HashSet<uint>** in the **UpdateListView** method, you could maintain a set of checked element IDs as a field in your form class. This will prevent the need to create a new set every time you call **UpdateListView**. You can update this set whenever a node is checked or unchecked.
3. Use data binding to simplify the ListView population: Instead of manually creating **ListViewItem** objects and setting their properties, you can use data binding to automatically populate the ListView based on a data source (e.g., a **BindingList<ElementData>**). This will reduce the amount of code you need to write and maintain.
4. Parallelize the search for checked nodes in the TreeViews: When you call **GetCheckedNodes** for each TreeView, you can use **Task.Run()** to execute them in parallel. This can improve performance if you have a large number of nodes in your TreeViews.
5. Consider using a Virtual Mode for the ListView: If you expect the ListView to have a large number of items, using a Virtual Mode can significantly improve the performance and memory usage. In Virtual Mode, the ListView only creates the ListViewItem objects that are currently visible. You can enable Virtual Mode by setting the **VirtualMode** property of the ListView to **true** and handling the **RetrieveVirtualItem** event.

Please keep in mind that these optimizations might be unnecessary if the size of the data and the complexity of the tree structures are relatively small. It's essential to balance the trade-off between code complexity and performance improvements, especially if the improvements won't be noticeable in your use case.

The next assignment for us to tackle is as follows:

Please have a look at the following 4 artificial sample data sets, which are the contents of the text files we use in our application for input and playing around with the programing logic and write me please DONE when you are finished analyzing them.

\_lst\_LogData\_elm\_type.txt, where the data are arranged in the following format:

ID@Long Name@Handle

0@Elm\_Type\_1@123456701

1@Elm\_Type\_1/Elm\_Name\_1@123456702

2@Elm\_Type\_1/Elm\_Name\_2@123456703

3@Elm\_Type\_1/Elm\_Name\_3@123456704

4@Elm\_Type\_2@123456705

5@Elm\_Type\_2/Elm\_Name\_1@123456706

6@Elm\_Type\_2/Elm\_Name\_2@123456707

7@Elm\_Type\_2/Elm\_Name\_3@123456708

8@Elm\_Type\_3@123456709

9@Elm\_Type\_3/Elm\_Name\_1@123456710

10@Elm\_Type\_3/Elm\_Name\_2@123456711 11@Elm\_Type\_3/Elm\_Name\_2/Elm\_Leaf\_1@123456712 12@Elm\_Type\_3/Elm\_Name\_2/Elm\_Leaf\_2@123456712

\_lst\_LogData\_chn.txt, where the data are arranged in the following format:

ID@Long Name@Handle

0@Channel\_1@123456800

1@Channel\_1/Name\_1@123456801

2@Channel\_1/Name\_2@123456802

3@Channel\_1/Name\_3/Leaf\_1@123456803

4@Channel\_2@123456805

5@Channel\_2/Name\_1@123456806

6@Channel\_2/Name\_1/Leaf\_1@123456807

7@Channel\_2/Name\_2@123456808

8@Channel\_3@123456809

9@Channel\_3/Name\_1@123456810

10@Channel\_3/Name\_2@123456811

11@Channel\_3/Name\_2/Leaf\_1@123456812 12@Channel\_3/Name\_2/Leaf\_2@123456813

\_lst\_LogData\_dbs.txt, where the data are arranged in the following format:

ID@Long Name@Handle

0@Database\_1@123456900

1@Database\_1/Name\_1@123456901

2@Database\_1/Name\_2@123456902

3@Database\_1/Name\_3@123456903

4@Database\_2@123456904

5@Database\_2/Name\_1@123456905

6@Database\_2/Name\_2/Leaf\_1@123456906 7@Database\_2/Name\_2/Leaf\_2@123456907

8@Database\_3@123456908

9@Database\_3/Name\_1@123456909

10@Database\_3/Name\_2@123456910

11@Database\_3/Name\_3/Leaf\_1@123456911 12@Database\_3/Name\_3/Leaf\_2@123456912

\_lst\_LogData\_elm\_all.txt, where the data are arranged in the following format: Line\_ID@Long Name@Short Name@Elem Type@Channel@Database@Location@Handle

0@LONG\_ELM\_0@SHORT\_ELM\_0@0@0@0@aaaaa@000000000 1@LONG\_ELM\_1@SHORT\_ELM\_1@1@1@1@bbbbb@111111111 2@LONG\_ELM\_2@SHORT\_ELM\_2@2@2@2@ccccc@222222222 3@LONG\_ELM\_3@SHORT\_ELM\_3@3@3@3@ddddd@333333333 4@LONG\_ELM\_4@SHORT\_ELM\_4@4@4@4@eeeee@444444444 5@LONG\_ELM\_5@SHORT\_ELM\_5@5@5@5@fffff@555555555 6@LONG\_ELM\_6@SHORT\_ELM\_6@6@6@6@ggggg@666666666 7@LONG\_ELM\_7@SHORT\_ELM\_7@7@7@7@hhhhh@777777777 8@LONG\_ELM\_8@SHORT\_ELM\_8@8@8@8@iiiii@888888888 9@LONG\_ELM\_9@SHORT\_ELM\_9@9@9@9@jjjjj@999999999 10@LONG\_ELM\_10@SHORT\_ELM\_10@9@9@9@kkkkk@999999991 11@LONG\_ELM\_11@SHORT\_ELM\_11@9@9@9@hhhhh@999999992 12@LONG\_ELM\_12@SHORT\_ELM\_12@12@12@12@ggggg@999999993

Next, I would like you to modify our application so that when a node in a TreeView control is selected then the underlying leaf node’s ID is looked up in the respective column in the dataset obtained from the \_lst\_LogData\_elm\_all.txt file. That is, a selected node from the treeViewElemType control is looked up in the Elem Type column, a selected node from the treeViewChannel control is looked up in the Channel column and a selected node from the treeViewDatabase control is looked up in the Database column all based on the selected node’s ID number and only then if the respective ID number is found in the respective column the data from the entire row is displayed in the ListView.

Moreover, I would like the numerical data in the Elem Type, Channel and Database columns not to be numerical but rather textual corresponding to the Long Name from the respective dataset. That is, for example if node ‘11@Elm\_Type\_3/Elm\_Name\_2/Elm\_Leaf\_1@123456712’ is selected, then its second token ‘Elm\_Type\_3/Elm\_Name\_2/Elm\_Leaf\_1’ is looked up in the dataset obtained from the ‘\_lst\_LogData\_elm\_type.txt’ file based on the selected ID number and that token is displayed in the listViewElements control instead of just a numeric figure. Same logic applies to Channel and Database columns. Can you do that for me please?

Firstly, your solution does not work. None of the selected nodes are displayed in the ListView. Secondly, I personally think that it would be wise to create dictionaries to map the ID to the Long Name for element types, channels, and databases at the start of the application and then use them throughout the lifetime of the application as opposed to creating them every time the UpdateListView is invoked. Here is what I suggest. Let me provide you with the entire working code again and then you will analyze the whole thing and try to add this additional logic. tell me if you agree with me, please.

But now only the ID numbers are displayed in the ListView as opposed to their matching names as before, please have a close look at the following two methods that you came up with and perhaps merge the functionality to meet the requirements. Please allow me to reiterate a few key concepts. I have 4 data sets. One for Element Type, one for Channel, one for Database and one for a list of All Elements. In the all elements dataset each row contains 7 columns, namely ID, Long Name, Short Name, Elem Type, Channel, Database, Location, Handle. The ID numbers are only unique within a given dataset. So, when we select a node from any TreeView control, that is either from Element Type dataset or Channel dataset or Database dataset the node’s ID number is used to lookup the entire element data in the All Elements dataset and only then display it in the ListView. It also replaces the numeric figures in the respective columns, namely Element Type, Channel and Database with the node’s name that is looked up in the selected node’s dataset. Do you see what I mean?

private void UpdateListView()

{

listViewElements.Items.Clear();

var checkedElemTypeNodes = GetCheckedNodes(treeViewElemType.Nodes).OfType<TreeNode>().ToList();

var checkedChannelNodes = GetCheckedNodes(treeViewChannel.Nodes).OfType<TreeNode>().ToList();

var checkedDatabaseNodes = GetCheckedNodes(treeViewDatabase.Nodes).OfType<TreeNode>().ToList();

var allCheckedNodeIds = new HashSet<uint>(checkedElemTypeNodes.Concat(checkedChannelNodes).Concat(checkedDatabaseNodes).Select(node => (node.Tag as MyTreeNode)?.m\_ID ?? 0));

foreach (var id in \_elementDataById.Keys)

{

if (allCheckedNodeIds.Contains(id))

{

ElementData elementData = \_elementDataById[id];

ListViewItem newItem = new ListViewItem(elementData.ID.ToString());

newItem.SubItems.Add(elementData.LongName);

newItem.SubItems.Add(elementData.ShortName);

newItem.SubItems.Add(\_elementTypeById.TryGetValue(uint.Parse(elementData.ElementType), out string elementType) ? elementType : elementData.ElementType);

newItem.SubItems.Add(\_channelById.TryGetValue(uint.Parse(elementData.Channel), out string channel) ? channel : elementData.Channel);

newItem.SubItems.Add(\_databaseById.TryGetValue(uint.Parse(elementData.Database), out string database) ? database : elementData.Database);

newItem.SubItems.Add(elementData.Location);

newItem.SubItems.Add(elementData.Handle.ToString());

listViewElements.Items.Add(newItem);

}

}

}

private void UpdateListView()

{

listViewElements.Items.Clear();

var checkedElemTypeNodes = GetCheckedNodes(treeViewElemType.Nodes).OfType<TreeNode>().ToList();

var checkedChannelNodes = GetCheckedNodes(treeViewChannel.Nodes).OfType<TreeNode>().ToList();

var checkedDatabaseNodes = GetCheckedNodes(treeViewDatabase.Nodes).OfType<TreeNode>().ToList();

var allCheckedNodeIds = new HashSet<uint>(checkedElemTypeNodes.Concat(checkedChannelNodes).Concat(checkedDatabaseNodes).Select(node => (node.Tag as MyTreeNode)?.m\_ID ?? 0));

var allCheckedNodes = checkedElemTypeNodes.Concat(checkedChannelNodes).Concat(checkedDatabaseNodes).Select(node => node.Tag as MyTreeNode).Where(node => node != null).GroupBy(node => node.m\_ID).ToDictionary(g => g.Key, g => g.First().m\_Handle);

foreach (var id in \_elementDataById.Keys)

{

if (allCheckedNodeIds.Contains(id))

{

ElementData elementData = \_elementDataById[id];

ListViewItem newItem = new ListViewItem(elementData.ID.ToString());

newItem.SubItems.Add(elementData.LongName);

newItem.SubItems.Add(elementData.ShortName);

newItem.SubItems.Add(elementData.ElementType);

newItem.SubItems.Add(elementData.Channel);

newItem.SubItems.Add(elementData.Database);

newItem.SubItems.Add(elementData.Location);

if (allCheckedNodes.TryGetValue(id, out uint handle))

{

newItem.SubItems.Add(handle.ToString());

}

else

{

newItem.SubItems.Add(elementData.Handle.ToString());

}

listViewElements.Items.Add(newItem);

}

}

}

**Question:**

My application also has 3 textboxes, one per TreeView and a Search button:

private TextBox textBoxElemType;

private TreeView treeViewElemType;

private TreeView treeViewChannel;

private TreeView treeViewDatabase;

private ListView listViewElements;

private Button buttonSearch;

private Label labelElemType;

private Label labelChannel;

private TextBox textBoxChannel;

private Label labelDatabase;

private TextBox textBoxDatabase;

They are yet to be implemented. Can you please assist me with this ordeal as well? Each text box is used to search data in the respective TreeView dataset. That is, the textBoxElemType is used to search the treeViewElemType, the textBoxChannel is used to search the treeViewChannel and the textBoxDatabase is used to search the treeViewDatabase based on the user-supplied text in the respective text box when the Search button is pressed. At this point it shall search only for the leaf node’s precise name within a given dataset. It might be a good idea to have some internal data set behind each TreeView control in order to perform the search faster. I will let you decide on it. Once and if there is a match, then the node with the matching name is marked checked and its data is displayed in the ListView in the manner we currently have.