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);

}

}