XML PROJECT - PART B
DEVELOPER DOCUMENTATION

SUBMISSION 2

DAN
GRIZLI777

Contents

Introduction

The contents of this document will provide a general overview of the functions of this application and how the application performs those functions.

Overview

The WoodStocks Stock Count Software is a simple Windows Forms Application which loads data from a comma delimited csv file or xml file and displays the contents on screen in a data table, allowing the user to change data fields, add new items and reorder the visual sorting order for each item.

The application uses StreamReader to retrieve data from the file line by line. Each line that comes in gets separated into its component fields and those fields are used to make the line into an object of type “item”. Each item is added to a List of Items, and that list is then passed into the Data Grid View which then displays the Item List for the user.

The application utilises Language Integrated Query (LINQ) for re-ordering the list depending on the selected field.

Saving to file casts the current information from the Data Grid View into a List of Items, which is then iterated over and used by StreamWriter to write line by line, the data fields for each item.

Each code block in the source code has been commented for clarity.

XML Code Generation in C#

XML can be generated using C# sharp code, utilizing the tools in the System.Xml.Linq namespace. In here are classes and functions for generating XML files using LINQ to XML C# code.

To generate our XML documents, we are first declaring a variable to store the XML as an “XDocument”, and adding an “XProcessingInstruction” which will appear at the top of the XML document. Within this processing instruction we are passing in a variable which will select between each of the style sheets to be referenced by the XML Document.

After that we create our first element which we name “items”, using the XElement class. We give it a tag name of “Items” followed by a comma, and then start a new line.

Because we have not input a value, nor have we closed the constructor parameter yet, every element we create now will become a child element of the “Items” element.

Now we start our LINQ query by specifying a placeholder variable named “s” and selecting where to look for our data; in this case “saveList” which is a List<Items> that has been passed into the method.

We instantiate a new “XElement” for each data field contained within “saveList” and nest those inside of another XElement called “Item”.

After our LINQ query, we add our “items” XElement (which will now retrieve all the data in saveList and organize it according to our defined structure) and add it to our XDocument.

The final step is to save the XDocument to our specified path as an XML file.

```
public void SaveAsXML(List<Item> saveList, int styleSelect)
{
    // Save the XML file with reference to style sheet
    try
    {
            var document = new XDocument(
                new XProcessingInstruction("xml-stylesheet", $"type='text/css' href='Stylesheet{styleSelect}.css'"
                ));

        var items = new XElement("Items",

            from s in saveList
            select new XElement("Item",
                new XElement("ItemCode", s.ItemNumber),
                new XElement("Description", s.ItemDescription),
                new XElement("CurrentCount", s.ItemCount),
                new XElement("OnOrder", s.ItemOnOrder)));

        document.Add(items);
        document.Save("C:\\StockFile\\stocklistxml.xml");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

XML Utilization

Classes and Code Blocks

The application is separated into two projects (excluding tests). A data layer and a user interface (UI) layer.

Contained in the UI Layer (WSSC.UI) the WoodStocks Stock Counter.cs file contains code for:

- Binding to and displaying the Data Grid View
- Calling for validation from outside methods
- Construction of Data Repository objects
- Message Boxes for user confirmation and error handling
- Calls to external methods for List sorting
- Event handling for Data Grid View
- Calls to external methods for saving
- Calls to XML generate methods

Contained in the Data Layer (WSSC.DL) the DataRepository.cs class contains code for:

- The default directory path
- Class constructors
- Stream Writer to write to file
- Stream Reader to read from file
- Message Boxes and Try Catch for error handling
- LINQ methods for data sorting

- Methods for Item creation and List<Item> creation

Contained in the Data Layer (WSSC.DL) the Item.cs class contains code for:

- Properties
- String override

Contained in the Data Layer (WSSC.DL) the Validator.cs class contains code for:

- Data validation

Contained in the Data Layer (WSSC.DL) the XMLGenerator class contains code for:

- Saving contents of grid view to an XML file
- Reading from XML file

Further Details

Upon loading, the software populates the Data Table with data from C:/StockFile/stocklist.cvs (as per the requirement) however because this path is hardcoded into the software it may potentially pose a problem in the future if the user wishes to keep the stock list in a different location.

An update to the software has seen that the user can now select a file to load (csv or xml) however the default file upon loading remains as "C:\StockFile\stocklist.csv". Depending on how the users' want to operate the software this may be desirable, however it may end up causing an inefficiency if a different default directory, file or file format ends up being used.