CVB815 Winter 2014 Assignment 1 - Database.dll Readme.txt

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

-- SUMMARY --

The Database.dll is a back end VB.NET dynamic link library (DLL) that can be

used to store, retrieve, add, remove and modify customer, product and order

information from CSV and SOAP serialized files. In addition to providing

entity classes to hold these records, the library also provides an abstract

record structure that can be extended to hold other entity classes with

similar attributes.

The individual customer, product and order objects are stored in type

specific book classes that extend the Abstract Book class. The Book class

uses Generics and can be used to hold records of any type the user of

Database.dll creates.

-- REQUIREMENTS --

The Database.dll library was developed with the .NET Framework version 4.5.1,

and has not been tested for use with any prior versions.

-- INSTALLATION --

No special installation is required. Simply unzip the Zip file containing the

Database.dll file and store it in a local directory. Then create a new project

in visual studio. Right click the project and select Add->Reference. Browse to

the local directory where you stored the Database.dll file and select it.

Now at the top of your .NET source file, type the appropriate 'Import Database'

or 'using Database' statement depending on the .NET programming language

being used.

-- CUSTOMIZATION --

The abstract record and book classes can be extended to hold entity classes

of other types that have similar needs. The reading and writing of the data

to the CSV files is handled generically with only the specific entity classes

knowing the details of the fields being stored. The SOAP serialization is

handled from the book classes, and just requires that any new classes that

are created to extend book and record are created as 'Serializable'.

-- TROUBLESHOOTING --

At this time, the client application must pick 1 of the 2 methods, CSV or SOAP,

to load the books from. On termination of the client, if the user chooses to

Save their data, then the content of the books as currently held in memory is

output to both the corresponding CSV and SOAP files and the 2 files are

synchronized at that time. If the end user chooses to manually modify the files,

then they should choose to load from the modified file the next time they

launch the client, otherwise the manual changes would be lost.

-- FAQ --

Q: How do I use Database.Dll in my application?

1. Unzip the Zip file containing the Database.dll file and store it in a local

directory.

2. Create a new project in visual studio.

3. Right click the project and select Add->Reference. Browse to the local

directory where you stored the Database.dll file and select it.

4. At the top of your .NET source file, type the appropriate 'Import Database'

or 'using Database' statement depending on the .NET programming language

being used.

5. Create a new book using:

Dim productbook As New ProductBook()

6. Load the book from file specifying the location of the csv or SOAP XML file:

productbook = productbook.Load(PRODUCTS\_PATH)

7. Now you can add/remove entries from the book, using the Add/Remove API.

8. Before closing your application, the book can be saved using:

productbook.SaveCSV(PRODUCTS\_CSV\_PATH) AND/OR

productbook.SaveXML(PRODUCTS\_XML\_PATH)

Q: I have a different kind of record I want to use in my application. Can I still

use Database.dll

Yes! A new class can be defined to extend the 'Record' class. Declare your new

class to be <Serializable> and Inherit from Record, and implement IRecord. Add

any new attributes needed to your new class. Now create a corresponding 'Book'

class that is also <Serializable> and extends 'Book'.

Q: My application is loading from CSV but not from XML.

You are likely running productbook.Load but not assigning the return variable to

anything; it is being properly loaded in a new book, but then discarded. Capture

it and you should be fine!

-- INTERACTION BETWEEN BOOKS --

Books can refer to other books; in order to keep each book ignorant of the

structure of other books, the following compromise was reached:

1: A record might store a reference to a record in another book (or its own,

for customized versions of the Book/Record classes).

2: For our implementation, this is stored in the child record, referencing the

parent. This resolves complexities with a many-to-one structure: each child

book is given a public method to return a collection going the other way.

3: In order to have an active reference rather than an id stored, each child

record fires an event that can be handled by the client application, which

allows the client to make a link if it wishes. You can certainly use the book

without this link -- getByID exists as a standard method for a reason -- but

using the exceptions in this manner can give you improved transactional

performance.

See the NewAddress and NewOrderItem methods in the sample file to see how

this can happen.

-- OUR EXPERIENCES --

In the process of developing our Database.dll library, we have learned many

new things about developing libraries and applications for the .NET Framework.

Our initial code for our library started out with Lab 3 where we learned to

read and parse CSV files in order to perform searches. In Lab 4 we extended

this functionality and learned how to create a separate C# library and use

it to provide an interface for our field validation. In lab 4 we also learned

how to raise events in our library and handle events in our console client.

In Lab 5, when we identified the need for a second entity class with similar

characteristics to the existing Address Class we defined an Abstract base

class to hold the common fields and methods that our 2 entity classes would

use. In addition to this, we added the Abstract Book class to implement the

Generic Collection Handling, as well as taking over responsibility for the

Serialization and CSV reading/writing.

We had our initial project design meeting on Feb 6th. We discussed an initial

version of the class diagram, as well as initial UI screens. On February 13th,

we had another meeting to discuss design changes required based on the new

.NET features we learned from Lab 4 and Lab 5, including the SOAP

serialization.

At the meeting on February 13th, we discussed the difficulties we were having

regarding keeping our code synchronized with each other and not duplicating

work. After some research, we decided we would use the free web-based Git

repository service called 'GitHub' to manage our source code. This tool is

fully integrated into Visual Studio 2013 and made our lives much easier for

continuing to progress on the project. On February 19th, our GitHub

repository was created, and all our source code changes since that date are

stored in the repository. To see all of our source code revisions, our

repository can be viewed at: https://github.com/trevel/vbas1.

Most of our project communication was done in person at school or via text

messaging. One of the major changes our library underwent was on February 22nd

after an email discussion, we decided to move the specific CSV line parsing

out of the Book classes and into the specific Record classes. This allows

the Book classes to be unaware of the structure of the records in the CSV

file. On February 26th we demonstrated our project during class time and

subsequently made a few small user interface changes to improve our order,

product and customer display.

-- CONTACT --

Current maintainers:

Laurie Shields (llshields@myseneca.ca)

Mark Lindan (melindan@myseneca.ca)