

Address Book Application Version 1.0

Technical Specification Document

[Document History 2](#_Toc126579965)

[Outstanding Issues 2](#_Toc126579966)

[Overview 2](#_Toc126579967)

[Architecture approach 2](#_Toc126579968)

[ER Diagram 3](#_Toc126579969)

[Database Layer 3](#_Toc126579970)

[“addressBookDB” constructor 3](#_Toc126579971)

[“GetConnectionObject” function 4](#_Toc126579972)

[GetCommand Function 5](#_Toc126579973)

[GetAddresses Function 5](#_Toc126579974)

[GetAddresses (int intAddressid) 6](#_Toc126579975)

[deleteAddress(int intAddressid) 7](#_Toc126579976)

[addAddress 8](#_Toc126579977)

[Business Layer 8](#_Toc126579978)

[AddressBook class 9](#_Toc126579979)

[addAddress() 10](#_Toc126579980)

[deleteAddress() 10](#_Toc126579981)

[AddressBooks ( inherits from System.Collections.Collectionbase) 10](#_Toc126579982)

[LoadAddress 10](#_Toc126579983)

[LoadAddress(int intAddressid) 11](#_Toc126579984)

[Presentation Layer 12](#_Toc126579985)

[FrmAddressBook.cs 12](#_Toc126579986)

[GUI components 13](#_Toc126579987)

[setValueFromUI 13](#_Toc126579988)

[ClearUI 13](#_Toc126579989)

[LoadAddress 14](#_Toc126579990)

[LoadAddressinUI 14](#_Toc126579991)

[Method call flow and action for address book 15](#_Toc126579992)

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Changed By | Comments |
| 1.0 | 2006.1.1 | Shivprasad Koirala | Initial Draft |

# Outstanding Issues

|  |  |  |  |
| --- | --- | --- | --- |
| Issue | Resolution | Date Resolved | Owner |
|  |  |  |  |

# Overview

This document will decide in detail the technical specification for the address book application according to the Use case and the initial scope provided.

# Architecture approach

This is a simple desktop application which will be used by limited users. We will be using three tier architecture with access database. Access database was recommended because very less data growth is expected.

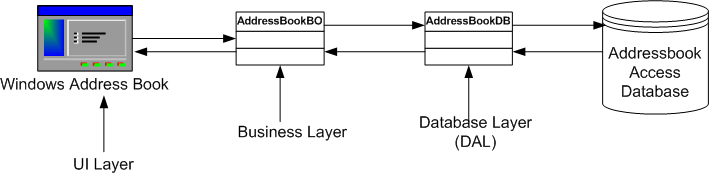


Figure: - Architecture road map for address book application

# ER Diagram

Below is the ER diagram for address database. “AddressId” is the primary key.

|  |  |
| --- | --- |
| **Field Name** | **DataType** |
| AddressId (Primary Key) | AutoNumber |
| Name | Text |
| Address | Memo |
| PhoneNumber | Text |

# Database Layer

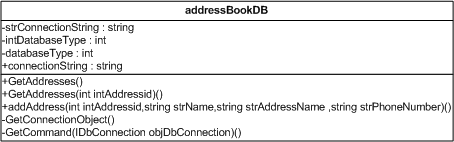


Figure: - Database access layer diagram

## “addressBookDB” constructor

Connection string and Database type is initialized in the constructor of addressBookDB class. Both the values will be stored in “App.config” file in “appSettings” section with “ConnectionString” and “DatabaseType” key. Below is how the configuration file section will look like

*<appSettings>*

*<add key="ConnectionString" value="here goes the connection string"/>*

*<add key="DatabaseType" value="1"/>*

*</appSettings>*

Database type will any of the following type :-

* Access intDatabaseType = 1
* SQL Server intDatabaseType = 2
* Oracle intDatabaseType = 3

Pseudo-code for the constructor will be as below:-

*public addressBookDB()*

*{*

*// create the appsettingReader object*

*AppSettingsReader objAppsettingreader = new AppSettingsReader();*

*// Get the Connectionstring using the “ConnectionString” key*

*strConnectionString =objAppsettingreader.GetValue("ConnectionString");*

*// Get the DatabaseType using the “DatabaseType” key.*

*intDatabaseType = objAppsettingreader.GetValue("DatabaseType");*

*}*

## “GetConnectionObject” function

This function return connection object with IDbConnection interface. This function will use connection string loaded in the “strConnectionString” property. This function will do the following things:-

* If there is no “databasetype” specified it will raise illegal provider error.
* Depending on databasetype it will create the connection object. For phase one we will be only providing service to access database. So the “OleDBConnection” class object will created and assigned to the interface IDBConnection.

Note: - “IDBConnection” interface is used as we need to keep every thing generalized so that we can support multiple database types like Access, SQL Server etc. Below goes the pseudo-code for the same.

*public IDbConnection GetConnectionObject()*

*{*

*IDbConnection objConnection = null;*

*if (intDatabaseType==0)*

*{*

*Throwexception that illegal database provider;*

*}*

*// If its access then use the OLeDBConnection class to create object.*

*// Other database like oracle and sql server will be considered for the second phase.*

*if (intDatabaseType == 1)*

*{*

*objConnection = new OleDbConnection();*

*}*

*// set the connection string*

*objConnection.ConnectionString = strConnectionString;*

*return objConnection;*

*}*

## GetCommand Function

This function will return a command object interface depending on the connection object passed. This function also returns “IDBCommand” so that we can operate with generalized command object for all database types (like Access, SQL Server etc).

*public IDbCommand GetCommand(IDbConnection objDbConnection)*

*{*

*IDbCommand objCommand = objDbConnection.CreateCommand();*

*return objCommand;*

*}*

## GetAddresses Function

This function will return addresses from address database as “IDataReader” interface. Which the clients can loop and use the same. Below will be the pseudo-code for “GetAddresses” function:-

*Public function IDataReader GetAddresses()*

*{*

*// Get the connection object using GetConnectionObject() function*

*objConnection = GetConnectionObject();*

*// open the connection object*

*objConnection.Open();*

*// Using the GetCommand function and objConnection get the command object*

*objCommand = GetCommand(objConnection);*

*// current this DAL component will only support simple SQL*

*// and not Stored procedures*

*objCommand.CommandType = CommandType.Text;*

*// We need all the records so us the Select \* from SQL*

*objCommand.CommandText = "Select \* from Address";*

*// finally execute the reader*

*objDataReader = objCommand.ExecuteReader();*

*// and then return the reader to the function*

*Return objDataReader;*

*}*

## GetAddresses (int intAddressid)

This is an over loaded function for “GetAddresses” which takes addressid as input. There is no such difference from the previous function and the SQL statement will change.

*Public function IDataReader GetAddresses(int intAddressid)*

*{*

*// Get the connection object using GetConnectionObject() function*

*objConnection = GetConnectionObject();*

*// open the connection object*

*objConnection.Open();*

*// Using the GetCommand function and objConnection get the command object*

*objCommand = GetCommand(objConnection);*

*// current this DAL component will only support simple SQL*

*// and not Stored procedures*

*objCommand.CommandType = CommandType.Text;*

*// We need to select only one address with the specific addressid*

*objCommand.CommandText = "Select \* from Address where addressid=" + intAddressid;*

*// finally execute the reader*

*objDataReader = objCommand.ExecuteReader();*

*// and then return the reader to the function*

*Return objDataReader;*

*}*

## deleteAddress(int intAddressid)

This function deletes the specific address record specified

*public void deleteAddress(int intAddressid)*

*{*

*// First get the connection*

*objConnection = GetConnectionObject();*

*// open the connection*

*objConnection.Open();*

*// get the command object using this connection object*

*objCommand = GetCommand(objConnection);*

*// current this DAL component will only support simple SQL*

*// and not Stored procedures*

*objCommand.CommandType = CommandType.Text;*

*// check first if there is any addressid passed*

*if (intAddressid != 0)*

*{*

*// then delete the addressid from the address table*

*objCommand.CommandText = "delete from Address where addressid=" + intAddressid.ToString();*

*}*

*// execute the command*

*objCommand.ExecuteNonQuery();*

*// close the objec*

*objCommand.Connection.Close();*

*}*

## addAddress

This method adds and updates address records to the address table. The dotted lines are the same code as defined for the previous “Getaddresses” and “deleteaddress” sub routines. Dotted lines will have the same code

* Get the connection object
* Get the command object.
* Set the commandtext
* And finally execute the command object.

*Public void addAddress(int intAddressid,string strName,string strAddressName,string strPhoneNumber)*

*{*

*…*

*…*

*..*

*..*

*if (intAddressid==0)*

*{*

*objCommand.CommandText = "insert into Address(Name,Address,Phonenumber) values('" + strName + "','" + strAddressName + "','" + strPhoneNumber + "')" ;*

*}*

*else*

*{*

*objCommand.CommandText = "update Address set name='" + strName + "', Address='" + strAddressName + "',phonenumber='" + strPhoneNumber + "' where addressid=" + intAddressid.ToString();*

*}*

*..*

*..*

*Objcommand.executenonquery();*

*}*

# Business Layer

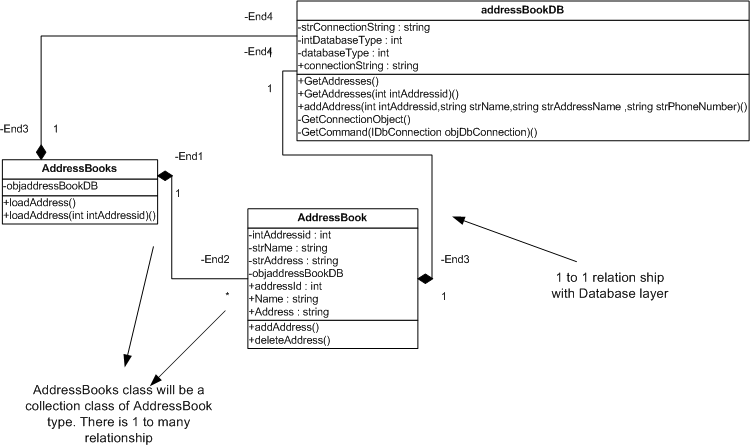


Figure: - Business object diagram

Above is the business object diagram for the address application. There are two main business objects:-

## AddressBook class

This class will have the validations and two basic operational functionalities adding new addresses and second delete addresses. “AddressBook” class represents a single entity class.

All the business validations will be written in the set property of the class. For instance for the name property you can see that if there is no name provided it will throw error to the client.

*Public string Name*

*{*

*Set*

*{*

*// if the strName is not provided then throw a exception*

*If value.length == 0*

*{*

*Throw new exception(“Name is a compulsory field”);*

*}*

*// if all the validations are ok then set the private value*

*strName = value;*

*}*

*Get*

*{*

*return strName;*

*}*

*}*

Same type of business validation will follow for address property.

## addAddress()

This method will call the “addAddress” method of the database class “addressBookDB” and add new address record in the database.

Pseudo-code for calling the database object will be something as below.

*addressBookDB.addAddress (intAddressid, strName, strAddress, strPhoneNumber)*

Also note the private variables passed to the address database objects,

## deleteAddress()

This method will call the “deleteAddress” method of the “addressBookDB” class and delete a particular address from the address database.

Pseudo-code for calling the database object will be something as below.

*addressBookDB.deleteAddress(intAddressid);*

## AddressBooks ( inherits from System.Collections.Collectionbase)

This class will be having collection of “AddressBook” type objects and it inherits from the System.collections.Collectionbase class. “AddressBooks” represents a collection. It has the following methods and properties:-

## LoadAddress

“Loadaddress” method loads the address in to collection base. Below is the pseudo-code for the same.

*public void loadAddress()*

*{*

*// before we load all the address clear the list collection*

*List.Clear();*

*// call the addressbookdb and and the get the address datareader*

*objDatareader = objaddressBookDB.GetAddresses();*

*// loop through the datareader and load the*

*// collection of addressbooks*

*while (objDatareader.Read())*

*{*

*AddressBook objAddressBook ;*

*objAddressBook = new AddressBook();*

*objAddressBook.addressId = objDatareader["Addressid"];*

*objAddressBook.Address = objDatareader["Address"].ToString();*

*objAddressBook.Name = objDatareader["Name"].ToString();*

*objAddressBook.PhoneNumber = objDatareader["PhoneNumber"].ToString();*

*// after the values are set add it to the list collection*

*List.Add(objAddressBook);*

*}*

*// finally close the object*

*objDatareader.Close();*

*}*

## LoadAddress(int intAddressid)

This is a overloaded method it will have the same logic as defined previously for “loadAddress” method without parameters. Only that it will call

*objaddressBookDB.GetAddresses(intAddressid);*

# Presentation Layer

## FrmAddressBook.cs

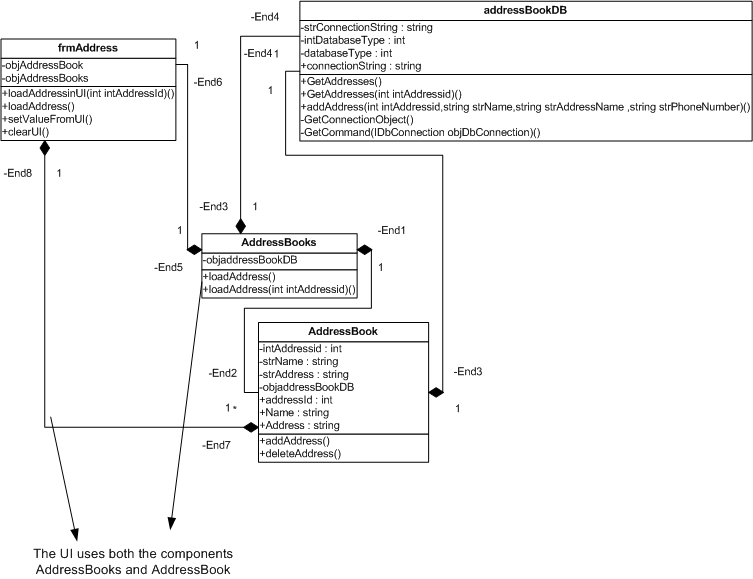


Figure: - UI component Diagram

Presentation layer define the way “addressbook” and “addressbooks” object will flow and interact.

Following are the various methods and properties in the presentation layer:-

AddressBook and AddressBooks object will define a private variable in the presentation layer.

*private AddressBook objAddressBook = new AddressBook();*

*private AddressBooks objAddressBooks = new AddressBooks();*

## GUI components

|  |  |  |
| --- | --- | --- |
| **GUI Object Name** | **Type** | **Description** |
| txtName | System.Windows.Forms.TextBox | Textbox for taking Name which is given by user. |
| txtAddress; | System.Windows.Forms.TextBox | Textbox for taking address data which is given by user. |
| txtPhoneNumber | System.Windows.Forms.TextBox | Will be used to take data for Phone number. |
| lblName , lblAddress and lblPhoneNumber | System.Windows.Forms.Label | Label to display the caption for the three entities. |
| cmdUpdate, cmdCancel and cmdDelete | System.Windows.Forms.Button | Command buttons for “Adding” , “deleting” address data. Cancel will clear what ever is typed by user. |
| dgAddresses | System.Windows.Forms.DataGrid | Grid to display address data. |

## setValueFromUI

This will take the value from the UI components and set it to the address object. Pseudo-code is as shown below:-

*private void setValueFromUI()*

*{*

*objAddressBook.Name = txtName.Text;*

*objAddressBook.PhoneNumber = txtPhoneNumber.Text;*

*objAddressBook.Address = txtAddress.Text;*

*}*

## ClearUI

This sub routine clears all the user interface values.

*private void clearUI()*

*{  
txtName.Text = "";*

*txtPhoneNumber.Text = "";*

*txtAddress.Text = "";  
}*

## LoadAddress

This subroutine will bind the address collection object that is objAddressBooks to the datagrid.

*private void loadAddress()*

*{*

*// call the load method of the addressbooks object*

*objAddressBooks.loadAddress();*

*// clear all the previous records in the grid*

*dgAddresses.DataSource = null;*

*//rebind the fresh new data in t he addressbooks collection*

*dgAddresses.DataSource = objAddressBooks;*

*// refresh the grid*

*dgAddresses.refresh();*

*}*

## LoadAddressinUI

“LoadAddressinUI” does the exact opposite of what “SetValueFromUI” does.

*public void loadAddressinUI(int intAddressId)*

*{*

*// Load the addressbooks object with the addressid*

*objAddressBooks.loadAddress(intAddressId);*

*// check if there are any address object loaded*

*if (objAddressBooks.Count > 0)*

*{*

*// if yes then set the current addressbook object to the first object of the*

*// addressbooks collection object*

*objAddressBook = objAddressBooks[0];*

*}*

*// finally load all the object value in to the address book UI*

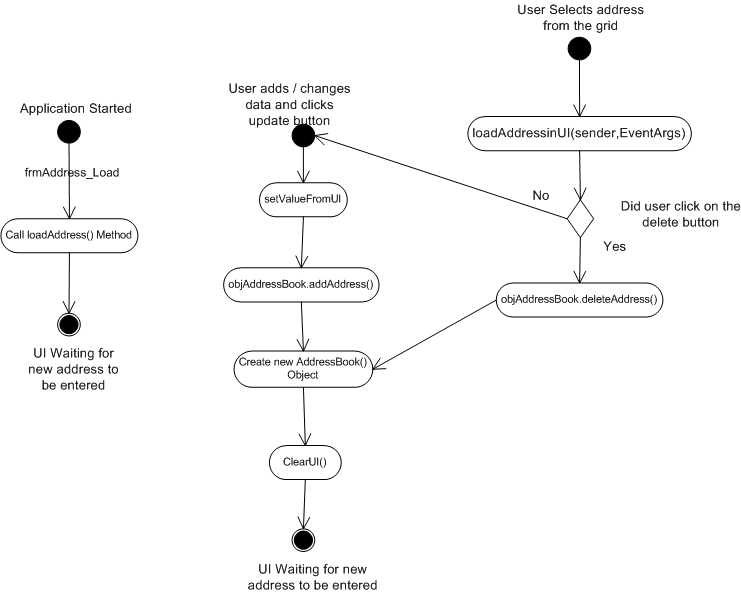
*txtName.Text = objAddressBook.Name;*

*txtAddress.Text = objAddressBook.Address;*

*txtPhoneNumber.Text = objAddressBook.PhoneNumber;*

*}*

## Method call flow and action for address book



**Figure: - Method flow in UI**

Above shows how the methods will interact among themselves to accomplish the task.