Technical manual

// Programmers :Alrick Visagie : 214086402

// Shahbaaz Sheikh : 214066614

// Moeketsi Betana : 214110370

// Watlinton Moholo : 214030377

Presentation layer

Loginwindow.xaml.cs

/// <summary>

/// Interaction logic for LoginWindow.xaml

/// </summary>

public partial class LoginWindow

{

private static string Path = System.IO.Path.Combine(Environment.GetFolderPath(

Environment.SpecialFolder.ApplicationData), "StorageUnitManagementDB.db");

private string \_conStr;

private SQLiteConnection \_sqlCon;

private UBL \_ubl;

public List<User> User

{

//

//Property Name : Automatic property List<SalaryEmployee> SEObjects

//Purpose : Automatic Public property containing all the SalaryEmployee objects

//Re-use : none

//Input : List<SalaryEmployee>

// - generic list containing all the SalaryEmployee objects

//Output : List<SalaryEmployee>

// - generic list containing all the SalaryEmployee objects

//

get;

set;

} // end property

public LoginWindow()

{

InitializeComponent();

CreateDatabase.CreateDb();

}//end method

private void Hyperlink\_Register(object sender, RequestNavigateEventArgs e)

{

Process.Start(new ProcessStartInfo(e.Uri.AbsoluteUri));

e.Handled = true;

}//end method

private void Hyperlink\_ForgotPassword(object sender, RequestNavigateEventArgs e)

{

Process.Start(new ProcessStartInfo(e.Uri.AbsoluteUri));

e.Handled = true;

}//end method

private void btnLogin\_Click(object sender, RoutedEventArgs e)

{

\_ubl = new UBL("UserSQLiteProvider");

User = \_ubl.SelectAll();

MainWindow window = new MainWindow();

foreach (User user in User)

{

if (textBox.Text.ToString() == user.UserName.ToString() && textBox1.Text.ToString() == user.Password.ToString())

{

window.Show();

MahApps.Metro.Controls.Dialogs.DialogManager.ShowMessageAsync(window, "Logging In", "Successfull Press OK to continue");

this.Close();

window.TextBlock1.Text = user.UserName;

}//end if

else

{

MahApps.Metro.Controls.Dialogs.DialogManager.ShowMessageAsync(this, "Logging In", "Unsuccessfull, Password or Username Incorrect");

break;

}//end else

}//end foreach

}//end method

private void lettersOnlyTextBox\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (Char.IsLetter((char)e.Key)) e.Handled = true;

}//end method

}//end class

MainWindow.xaml.cs

namespace StorageUnitManagementSystem

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow

{

Dictionary<string, string> dictionary = new Dictionary<string, string>();

public Client MyClient { get; set; }

private SUBL \_subl;

private CBL \_cbl;

private LUBL \_lubl;

private UBL \_ubl;

public Client ClientObj { get; set; }

public List<LeaseUnits> LeaseUnits { get; set; }

public List<StorageUnit> StorageUnits { get; set; }

public List<Client> Clients { get; set; }

public List<string> Data { get; } = new List<string> {"Client ID", "Name", "Surname", "City", "Province"};

public PopUp PopUp = new PopUp();

public List<string> cb\_UnitListSearchItems { get; } = new List<string>

{

"Vacant Units",

"Occupied Units",

"In Arrears",

"Up-To-Date",

"In Advance"

};

public List<string> cb\_UnitListSearchItemsCopy { get; } = new List<string>

{

"Vacant Units",

"Occupied Units",

"In Arrears",

"Up-To-Date",

"In Advance",

"ID"

};

private GridViewColumnHeader \_listViewSortCol = null;

private SortAdorner \_listViewSortAdorner = null;

private GridViewColumnHeader \_listViewSortColUnits = null;

private SortAdorner \_listViewSortAdornerUnits = null;

private List<StorageUnit> \_suObjects;

private StorageUnit \_insertStorageUnit;

int count = 0;

public MainWindow()

{

InitializeComponent();

LoginWindow login = new LoginWindow();

\_cbl = new CBL("ClientSQLiteProvider");

\_subl = new SUBL("StorageUnitSQLiteProvider");

\_lubl = new LUBL("LeaseUnitsSQLiteProvider");

\_ubl = new UBL("UserSQLiteProvider");

DataContext = new StorageUnit();

}//end method

private void Hyperlink\_RequestNavigate(object sender, RequestNavigateEventArgs e)

{

Process.Start(new ProcessStartInfo(e.Uri.AbsoluteUri))

e.Handled = true

}//end method

private void button3\_Click(object sender, RoutedEventArgs e)

{

//TEST COMMITTTTTTTTTTTTTTTTTTTTTTTTTTTT!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

LoginWindow login = new LoginWindow();

login.Show();

this.Close();

}//end method

//TO:DO COPY PASTE VALIDATION

private void lettersOnlyTextBox\_KeyDown(object sender, KeyEventArgs e)

{

try

{

if ((e.Key < Key.A) || (e.Key > Key.Z))

{

e.Handled = true;

}//end if

}//end try

catch (Exception ex)

{

this.ShowMessageAsync(ex.Message, "Main Window : lettersOnlyTextBox\_KeyDown");

}//end catch

}//end method

private void btnAddClient\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

Client clientObj = new Client();

try

{

if (TxtBoxId.Text.Equals("") && TxtBoxName.Text.Equals("") &&

TxtBoxSurname.Text.Equals("") && TxtBoxDateOfBirth.Text.Equals("") &&

TxtBoxCellPhone.Text.Equals("") && TxtBoxTelephone.Text.Equals("") &&

TxtBoxEmail.Text.Equals("") && TxtBoxAddress.Text.Equals("") &&

TxtBoxLine2.Text.Equals("") && TxtBoxCity.Text.Equals("") &&

TxtBoxProvince.Text.Equals("") && TxtBoxCode.Text.Equals(""))

{

this.ShowMessageAsync("Please Enter All Required Fields!", "");

}//end if

else

{

clientObj.idNumber = TxtBoxId.Text;

clientObj.FirstName = TxtBoxName.Text;

clientObj.LastName = TxtBoxSurname.Text;

clientObj.DateOfBirth = TxtBoxDateOfBirth.Text;

clientObj.Cellphone = TxtBoxCellPhone.Text;

clientObj.Telephone = TxtBoxTelephone.Text;

clientObj.EMailAddress = TxtBoxEmail.Text;

clientObj.Address.Line1 = TxtBoxAddress.Text;

clientObj.Address.Line2 = TxtBoxLine2.Text;

clientObj.Address.City = TxtBoxCity.Text;

clientObj.Address.Province = TxtBoxProvince.Text;

clientObj.Address.PostalCode = TxtBoxCode.Text;

rc = \_cbl.Insert(clientObj);

if (rc == 0)

{

this.ShowMessageAsync(

"Client: " + clientObj.FirstName + " " + clientObj.LastName + " Successfully Added!",

"Client Added");

TxtBoxId.Clear();

TxtBoxName.Clear();

TxtBoxSurname.Clear();

TxtBoxDateOfBirth.Clear();

TxtBoxCellPhone.Clear();

TxtBoxTelephone.Clear();

TxtBoxEmail.Clear();

TxtBoxAddress.Clear();

TxtBoxLine2.Clear();

TxtBoxCity.Clear();

TxtBoxProvince.Clear();

TxtBoxCode.Clear();

} // end if

else

{

this.ShowMessageAsync("Duplicate Client exists. Please try again.", "Client Not Added");

} // end else

}

} // end try

catch (Exception ex)

{

this.ShowMessageAsync(ex.Message, "Add Client: btnSubmit\_Click");

} // end catch

}//end method

private void bntRemoveSearch\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

Client clientObj = new Client();

LblClientName.Visibility = Visibility.Visible;

LblClientSurname.Visibility = Visibility.Visible;

LblClientCellPhone.Visibility = Visibility.Visible;

LblClientDateOfBirth.Visibility = Visibility.Visible;

LblClientTelephone.Visibility = Visibility.Visible;

LblClientAddress.Visibility = Visibility.Visible;

LblClientEmail.Visibility = Visibility.Visible;

TxtBoxRemoveClientName.Visibility = Visibility.Visible;

TxtBoxRemoveClientSurname.Visibility = Visibility.Visible;

TxtBoxRemoveClientCellPhone.Visibility = Visibility.Visible;

TxtBoxRemoveClientDateOfBirth.Visibility = Visibility.Visible;

TxtBoxRemoveClientTelephone.Visibility = Visibility.Visible;

TxtBoxRemoveClientAddress.Visibility = Visibility.Visible;

TxtBoxRemoveClientEmail.Visibility = Visibility.Visible;

BtnRemoveClient.Visibility = Visibility.Visible;

try

{

if (TxtBoxRemoveClientId.Text == "")

{

LblClientName.Visibility = Visibility.Hidden;

LblClientSurname.Visibility = Visibility.Hidden;

LblClientCellPhone.Visibility = Visibility.Hidden;

LblClientDateOfBirth.Visibility = Visibility.Hidden;

LblClientTelephone.Visibility = Visibility.Hidden;

LblClientAddress.Visibility = Visibility.Hidden;

LblClientEmail.Visibility = Visibility.Hidden;

TxtBoxRemoveClientName.Visibility = Visibility.Hidden;

TxtBoxRemoveClientSurname.Visibility = Visibility.Hidden;

TxtBoxRemoveClientCellPhone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientDateOfBirth.Visibility = Visibility.Hidden;

TxtBoxRemoveClientTelephone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientAddress.Visibility = Visibility.Hidden;

TxtBoxRemoveClientEmail.Visibility = Visibility.Hidden;

BtnRemoveClient.Visibility = Visibility.Hidden;

this.ShowMessageAsync("Please enter Client ID!", "");

}//end if

else

{

rc = \_cbl.SelectClient(TxtBoxRemoveClientId.Text, ref clientObj);

if (clientObj.Archived == Convert.ToBoolean(1))

{

LblClientName.Visibility = Visibility.Hidden;

LblClientSurname.Visibility = Visibility.Hidden;

LblClientCellPhone.Visibility = Visibility.Hidden;

LblClientDateOfBirth.Visibility = Visibility.Hidden;

LblClientTelephone.Visibility = Visibility.Hidden;

LblClientAddress.Visibility = Visibility.Hidden;

LblClientEmail.Visibility = Visibility.Hidden;

TxtBoxRemoveClientName.Visibility = Visibility.Hidden;

TxtBoxRemoveClientSurname.Visibility = Visibility.Hidden;

TxtBoxRemoveClientCellPhone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientDateOfBirth.Visibility = Visibility.Hidden;

TxtBoxRemoveClientTelephone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientAddress.Visibility = Visibility.Hidden;

TxtBoxRemoveClientEmail.Visibility = Visibility.Hidden;

BtnRemoveClient.Visibility = Visibility.Hidden;

this.ShowMessageAsync("Client Not Found!", "");

}//end if

else

{

if (rc == 0)

{

TxtBoxRemoveClientName.Text = clientObj.FirstName;

TxtBoxRemoveClientSurname.Text = clientObj.LastName;

TxtBoxRemoveClientCellPhone.Text = clientObj.Cellphone;

TxtBoxRemoveClientDateOfBirth.Text = clientObj.DateOfBirth;

TxtBoxRemoveClientTelephone.Text = clientObj.Telephone;

TxtBoxRemoveClientAddress.Text = clientObj.Address.Line1 + "\n" +

clientObj.Address.Line2 + "\n" +

clientObj.Address.City + "\n" +

clientObj.Address.Province + "\n" +

clientObj.Address.PostalCode;

TxtBoxRemoveClientEmail.Text = clientObj.EMailAddress;

}//end if

else

{

LblClientName.Visibility = Visibility.Hidden;

LblClientSurname.Visibility = Visibility.Hidden;

LblClientCellPhone.Visibility = Visibility.Hidden;

LblClientDateOfBirth.Visibility = Visibility.Hidden;

LblClientTelephone.Visibility = Visibility.Hidden;

LblClientAddress.Visibility = Visibility.Hidden;

LblClientEmail.Visibility = Visibility.Hidden;

TxtBoxRemoveClientName.Visibility = Visibility.Hidden;

TxtBoxRemoveClientSurname.Visibility = Visibility.Hidden;

TxtBoxRemoveClientCellPhone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientDateOfBirth.Visibility = Visibility.Hidden;

TxtBoxRemoveClientTelephone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientAddress.Visibility = Visibility.Hidden;

TxtBoxRemoveClientEmail.Visibility = Visibility.Hidden;

BtnRemoveClient.Visibility = Visibility.Hidden;

this.ShowMessageAsync("Client Not Found!", "");

}//end else

}//end else

}//end else

}//end try

catch (Exception ex)

{

this.ShowMessageAsync(ex.Message, "Remove Client: bntRemoveSearch\_Click");

}//end catch

}//end method

private void btnRemoveClient\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

Client clientObj = new Client();

try

{

rc = \_cbl.SelectClient(TxtBoxRemoveClientId.Text, ref clientObj);

if (rc == 0)

{

clientObj.Archived = Convert.ToBoolean(1);

rc = \_cbl.Update(clientObj);

this.ShowMessageAsync("Client Successfully Removed!", "");

TxtBoxRemoveClientName.Clear();

TxtBoxRemoveClientSurname.Clear();

TxtBoxRemoveClientCellPhone.Clear();

TxtBoxRemoveClientDateOfBirth.Clear();

TxtBoxRemoveClientTelephone.Clear();

TxtBoxRemoveClientAddress.Clear();

TxtBoxRemoveClientEmail.Clear();

LblClientName.Visibility = Visibility.Hidden;

LblClientSurname.Visibility = Visibility.Hidden;

LblClientCellPhone.Visibility = Visibility.Hidden;

LblClientDateOfBirth.Visibility = Visibility.Hidden;

LblClientTelephone.Visibility = Visibility.Hidden;

LblClientAddress.Visibility = Visibility.Hidden;

LblClientEmail.Visibility = Visibility.Hidden;

TxtBoxRemoveClientName.Visibility = Visibility.Hidden;

TxtBoxRemoveClientSurname.Visibility = Visibility.Hidden;

TxtBoxRemoveClientCellPhone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientDateOfBirth.Visibility = Visibility.Hidden;

TxtBoxRemoveClientTelephone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientAddress.Visibility = Visibility.Hidden;

TxtBoxRemoveClientEmail.Visibility = Visibility.Hidden;

BtnRemoveClient.Visibility = Visibility.Hidden;

}//end if

else

{

this.ShowMessageAsync("Client Not Found!", "");

}//end else

}//end try

catch (Exception ex)

{

this.ShowMessageAsync(ex.Message, "Remove Client: btnRemoveClient\_Click");

}//end catch

}//end method

private void txtBoxRemoveClientID\_TextChanged(object sender, TextChangedEventArgs e)

{

TxtBoxRemoveClientId.MaxLength = 13;

if (TxtBoxRemoveClientId.Text == "")

{

TxtBoxRemoveClientName.Clear();

TxtBoxRemoveClientSurname.Clear();

TxtBoxRemoveClientCellPhone.Clear();

TxtBoxRemoveClientDateOfBirth.Clear();

TxtBoxRemoveClientTelephone.Clear();

TxtBoxRemoveClientAddress.Clear();

TxtBoxRemoveClientEmail.Clear();

LblClientName.Visibility = Visibility.Hidden;

LblClientSurname.Visibility = Visibility.Hidden;

LblClientCellPhone.Visibility = Visibility.Hidden;

LblClientDateOfBirth.Visibility = Visibility.Hidden;

LblClientTelephone.Visibility = Visibility.Hidden;

LblClientAddress.Visibility = Visibility.Hidden;

LblClientEmail.Visibility = Visibility.Hidden;

TxtBoxRemoveClientName.Visibility = Visibility.Hidden;

TxtBoxRemoveClientSurname.Visibility = Visibility.Hidden;

TxtBoxRemoveClientCellPhone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientDateOfBirth.Visibility = Visibility.Hidden;

TxtBoxRemoveClientTelephone.Visibility = Visibility.Hidden;

TxtBoxRemoveClientAddress.Visibility = Visibility.Hidden;

TxtBoxRemoveClientEmail.Visibility = Visibility.Hidden;

BtnRemoveClient.Visibility = Visibility.Hidden;

}//end if

}//end method

private void lvRestoreClient\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

List<Client> clientObjects = new List<Client>();

clientObjects = \_cbl.SelectAll();

if (clientObjects.Count > 0)

{

foreach (Client temp in clientObjects)

{

if (temp.Archived == Convert.ToBoolean(1))

{

TxtBoxRestoreClientId.Text = temp.idNumber;

}//end if

}//end foreach

}//end if

else

{

this.ShowMessageAsync("There are no Clients to list", "No Clients");

}//end else

}//end method

private void cb\_addClass\_DropDownOpened(object sender, EventArgs e)

{

cb\_addClass.Items.Clear();

\_suObjects = \_subl.SelectAll();

List<string> classArray = new List<string>();

foreach (StorageUnit unit in \_suObjects)

{

classArray.Add(unit.UnitClassification);

}//end foreach

// You can convert it back to an array if you would like to

string[] classStrings = classArray.ToArray();

classStrings = classStrings.Distinct().ToArray();

for (int x = 0; x < classStrings.Length; x++)

{

cb\_addClass.Items.Add(classStrings[x]);

}//end for

cb\_addClass.SelectedIndex = 0;

}//end method

private void btnRestoreSearch\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

Client clientObj = new Client();

try

{

if (TxtBoxRestoreClientId.Text == "")

{

this.ShowMessageAsync("Please Enter Client ID", "");

}//end if

else

{

rc = \_cbl.SelectClient(TxtBoxRestoreClientId.Text, ref clientObj);

if (rc == 0)

{

if (clientObj.Archived == Convert.ToBoolean(1))

{

LvRestoreClient.Items.Clear();

LvRestoreClient.Items.Add(clientObj);

}//end if

else

{

this.ShowMessageAsync("Client Not Found!", "");

}//end else

}//end else

else

{

this.ShowMessageAsync("Client Not Found!", "");

}//end else

} //end else

}//end try

catch (Exception ex)

{

this.ShowMessageAsync(ex.Message, "Restore Client: btnRestoreSearch\_Click");

}//end catch

}//end method

private void txtBoxRestoreClientID\_TextChanged(object sender, TextChangedEventArgs e)

{

TxtBoxRestoreClientId.MaxLength = 13;

if (TxtBoxRestoreClientId.Text == "")

{

LvRestoreClient.Items.Clear();

TxtBoxRestoreClientId.Clear();

}//end if

}//end else

private void btnRestoreClient\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

Client clientObj = new Client();

try

{

if (TxtBoxRestoreClientId.Text == "")

{

this.ShowMessageAsync("Please Enter Client ID", "");

TxtBoxRestoreClientId.Clear();

}//end if

else

{

rc = \_cbl.SelectClient(TxtBoxRestoreClientId.Text, ref clientObj);

if (rc == 0 && clientObj.Archived != Convert.ToBoolean(0))

{

clientObj.Archived = Convert.ToBoolean(0);

rc = \_cbl.Update(clientObj);

LvRestoreClient.Items.Clear();

TxtBoxRestoreClientId.Clear();

this.ShowMessageAsync("Client Restored Successfully!", "");

}//end if

else

{

this.ShowMessageAsync("Client Not Found!", "");

}//end else

}//end else

}//end try

catch (Exception ex)

{

this.ShowMessageAsync(ex.Message, "Restore Client: btnRestoreSearch\_Click");

}//end catch

}//end method

private void lvClientsColumnHeader\_Click(object sender, RoutedEventArgs e)

{

GridViewColumnHeader column = (sender as GridViewColumnHeader);

string sortBy = column.Tag.ToString();

if (\_listViewSortCol != null)

{

AdornerLayer.GetAdornerLayer(\_listViewSortCol).Remove(\_listViewSortAdorner);

LvRestoreClient.Items.SortDescriptions.Clear();

}//end if

ListSortDirection newDir = ListSortDirection.Ascending;

if (\_listViewSortCol == column && \_listViewSortAdorner.Direction == newDir)

newDir = ListSortDirection.Descending;

\_listViewSortCol = column;

\_listViewSortAdorner = new SortAdorner(\_listViewSortCol, newDir);

AdornerLayer.GetAdornerLayer(\_listViewSortCol).Add(\_listViewSortAdorner);

LvRestoreClient.Items.SortDescriptions.Add(new SortDescription(sortBy, newDir));

}//end method

private void lvListClientsColumnHeader\_Click(object sender, RoutedEventArgs e)

{

GridViewColumnHeader column = (sender as GridViewColumnHeader);

string sortBy = column.Tag.ToString();

if (\_listViewSortCol != null)

{

AdornerLayer.GetAdornerLayer(\_listViewSortCol).Remove(\_listViewSortAdorner);

LvListClient.Items.SortDescriptions.Clear();

}//end if

ListSortDirection newDir = ListSortDirection.Ascending;

if (\_listViewSortCol == column && \_listViewSortAdorner.Direction == newDir)

newDir = ListSortDirection.Descending;

\_listViewSortCol = column;

\_listViewSortAdorner = new SortAdorner(\_listViewSortCol, newDir);

AdornerLayer.GetAdornerLayer(\_listViewSortCol).Add(\_listViewSortAdorner);

LvListClient.Items.SortDescriptions.Add(new SortDescription(sortBy, newDir));

}//end if

private void lvLeaseUnitsClientsColumnHeader\_Click(object sender, RoutedEventArgs e)

{

GridViewColumnHeader column = (sender as GridViewColumnHeader);

string sortBy = column.Tag.ToString();

if (\_listViewSortCol != null)

{

AdornerLayer.GetAdornerLayer(\_listViewSortCol).Remove(\_listViewSortAdorner);

LvLeaseUnits.Items.SortDescriptions.Clear();

}//end if

ListSortDirection newDir = ListSortDirection.Ascending;

if (\_listViewSortCol == column && \_listViewSortAdorner.Direction == newDir)

newDir = ListSortDirection.Descending;

\_listViewSortCol = column;

\_listViewSortAdorner = new SortAdorner(\_listViewSortCol, newDir);

AdornerLayer.GetAdornerLayer(\_listViewSortCol).Add(\_listViewSortAdorner);

LvLeaseUnits.Items.SortDescriptions.Add(new SortDescription(sortBy, newDir));

}//end method

private void btnRestoreListAll\_Click(object sender, RoutedEventArgs e)

{

List<Client> clientObjects = new List<Client>();

clientObjects = \_cbl.SelectAll();

LvRestoreClient.Items.Clear();

if (clientObjects.Count > 0)

{

LvRestoreClient.Items.Clear();

foreach (Client temp in clientObjects)

{

if (temp.Archived == Convert.ToBoolean(1))

LvRestoreClient.Items.Add(temp);

}//end if

}//end foreach

else

{

this.ShowMessageAsync("There are no Clients to list", "No Clients");

}//end else

}//end method

private void imgRefresh\_MouseDown(object sender, MouseButtonEventArgs e)

{

List<Client> clientObjects = new List<Client>();

clientObjects = \_cbl.SelectAll();

LvListClient.Items.Clear();

if (clientObjects.Count > 0)

{

LvListClient.Items.Clear();

foreach (Client temp in clientObjects)

{

if (temp.Archived == Convert.ToBoolean(0))

LvListClient.Items.Add(temp);

}//end foreach

}//end if

else

{

this.ShowMessageAsync("There are no Clients to list", "No Clients");

}//end else

}//end method

private void imgRefreshUnits\_MouseDown(object sender, MouseButtonEventArgs e)

{

List<StorageUnit> suObjects = new List<StorageUnit>();

suObjects = \_subl.SelectAll();

lv\_Units.Items.Clear();

if (suObjects.Count > 0)

{

lv\_Units.Items.Clear();

foreach (StorageUnit temp in suObjects)

{

lv\_Units.Items.Add(temp);

}//end foreach

}//end if

else

{

this.ShowMessageAsync("There are no Storage Units to list", "No Units");

}//end else

}//end method

private void imgRefreshUnitsSearch\_MouseDown(object sender, MouseButtonEventArgs e)

{

List<StorageUnit> suObjects = new List<StorageUnit>();

suObjects = \_subl.SelectAll();

lv\_Units\_Search.Items.Clear();

if (suObjects.Count > 0)

{

lv\_Units\_Search.Items.Clear();

foreach (StorageUnit temp in suObjects)

{

lv\_Units\_Search.Items.Add(temp);

}//end foreach

}//end if

else

{

this.ShowMessageAsync("There are no Storage Units to list", "No Units");

}//end else

}//end method

private void cboListSearch\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

try

{

ComboBoxItem typeItem = (ComboBoxItem) CboListSearch.SelectedItem;

string value = typeItem.Content.ToString();

}//end try

catch (Exception)

{

// ignored

}//end catch

}//end method

private void cboListSearch\_DropDownOpened(object sender, EventArgs e)

{

try

{

CboListSearch.Items.Clear();

CboListSearch.ItemsSource = Data;

}

catch (Exception)

{

// ignored

}//end catch

}//end method

private void cb\_UnitListSearch\_DropDownOpened(object sender, EventArgs e)

{

try

{

cb\_UnitListSearch.Items.Clear();

cb\_UnitListSearch.ItemsSource = cb\_UnitListSearchItems;

cb\_UnitListSearch.SelectedIndex = 0;

}

catch (Exception)

{

// Go Home WPF

}

}

private void btnListSearch\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

List<Client> clients = new List<Client>();

clients = \_cbl.SelectAll();

Client clientObj = new Client();

if (CboListSearch.SelectedItem.ToString() == "Client ID")

{

LvListClient.Items.Clear();

rc = \_cbl.SelectClient(TxtBoxListSearch.Text, ref clientObj);

if (rc == 0)

{

LvListClient.Items.Add(clientObj);

}

}

else if (CboListSearch.SelectedItem.ToString() == "Name")

{

LvListClient.Items.Clear();

foreach (Client client in clients)

{

if (TxtBoxListSearch.Text.Equals(client.FirstName))

{

LvListClient.Items.Add(client);

}

}

}

else if (CboListSearch.SelectedItem.ToString() == "Surname")

{

LvListClient.Items.Clear();

foreach (Client client in clients)

{

if (TxtBoxListSearch.Text.Equals(client.LastName))

{

LvListClient.Items.Add(client);

}

}

}

else if (CboListSearch.SelectedItem.ToString() == "City")

{

LvListClient.Items.Clear();

foreach (Client client in clients)

{

if (TxtBoxListSearch.Text.Equals(client.Address.City))

{

LvListClient.Items.Add(client);

}

}

}

else if (CboListSearch.SelectedItem.ToString() == "Province")

{

LvListClient.Items.Clear();

foreach (Client client in clients)

{

if (TxtBoxListSearch.Text.Equals(client.Address.Province))

{

LvListClient.Items.Add(client);

}

}

}

else

{

this.ShowMessageAsync("Client Does Not Exist", "No Client");

}

}

private void btn\_UnitListSearch\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

List<StorageUnit> suObjects = new List<StorageUnit>();

StorageUnit storageUnit = new StorageUnit();

switch (cb\_UnitListSearch.SelectedItem.ToString())

{

case "Sort By:":

this.ShowMessageAsync("Error", "Please Choose an Option from the Drop Down Box");

break;

case "Vacant Units":

lv\_Units.Items.Clear();

suObjects = \_subl.SelectAll();

foreach (StorageUnit unit in suObjects)

{

if (unit.UnitOccupied.Equals(false))

{

lv\_Units.Items.Add(unit);

rc = 1;

}

}

if (rc == 0)

{

this.ShowMessageAsync("Error", "No Vacant Units Found");

}

break;

case "Occupied Units":

lv\_Units.Items.Clear();

suObjects = \_subl.SelectAll();

foreach (StorageUnit unit in suObjects)

{

if (unit.UnitOccupied.Equals(true))

{

lv\_Units.Items.Add(unit);

rc = 1;

}

}

if (rc == 0)

{

this.ShowMessageAsync("Error", "No Occupied Units Found");

}

break;

case "In Arrears":

lv\_Units.Items.Clear();

suObjects = \_subl.SelectAll();

foreach (StorageUnit unit in suObjects)

{

if (unit.UnitArrears.Equals(true))

{

lv\_Units.Items.Add(unit);

rc = 1;

}

}

if (rc == 0)

{

this.ShowMessageAsync("Error", "No Units in Arrears Found");

}

break;

case "Up-To-Date":

lv\_Units.Items.Clear();

suObjects = \_subl.SelectAll();

foreach (StorageUnit unit in suObjects)

{

if (unit.UnitUpToDate.Equals(true))

{

lv\_Units.Items.Add(unit);

rc = 1;

}

}

if (rc == 0)

{

this.ShowMessageAsync("Error", "No Up-To-Date Units Found");

}

break;

case "In Advance":

lv\_Units.Items.Clear();

suObjects = \_subl.SelectAll();

foreach (StorageUnit unit in suObjects)

{

if (unit.UnitInAdvance.Equals(true))

{

lv\_Units.Items.Add(unit);

rc = 1;

}

}

if (rc == 0)

{

this.ShowMessageAsync("Error", "No Units Paid for in Advance Found");

}

break;

default:

this.ShowMessageAsync("Unit Does Not Exist", "No Unit");

break;

}

}

private void AddClientTextBoxValidation(object sender, TextChangedEventArgs e)

{

if (TxtBoxId.Text.Length == 0)

{

//txtBoxId.Foreground = System.Windows.Media.Brushes.Red;

TxtBoxId.SetValue(TextBoxHelper.WatermarkProperty, "ID number cannot be empty!");

}

else if (TxtBoxId.Text.Length > 12)

{

TxtBoxId.MaxLength = 13;

}

}

private void UIElement\_OnPreviewTextInput(object sender, TextCompositionEventArgs e)

{

e.Handled = !IsTextAllowed(e.Text);

}

private static bool IsTextAllowed(string text)

{

Regex regex = new Regex("[^0-9]+"); //regex that matches disallowed text

return !regex.IsMatch(text);

}

// Use the DataObject.Pasting Handler

private void TextBoxPasting(object sender, DataObjectPastingEventArgs e)

{

if (e.DataObject.GetDataPresent(typeof (String)))

{

String text = (String) e.DataObject.GetData(typeof (String));

if (!IsTextAllowed(text))

{

e.CancelCommand();

}

}

else

{

e.CancelCommand();

}

}

private void txtBoxCellPhone\_TextChanged(object sender, TextChangedEventArgs e)

{

TxtBoxCellPhone.MaxLength = 10;

}

private void txtBoxTelephone\_TextChanged(object sender, TextChangedEventArgs e)

{

TxtBoxTelephone.MaxLength = 10;

}

private void CbLeaseSelectClass\_DropDownOpened(object sender, EventArgs e)

{

CbLeaseSelectClass.Items.Clear();

//MessageBox.Show(cb\_addClass.SelectedItem.ToString());

StorageUnits = \_subl.SelectAll();

List<string> classArray = new List<string>();

foreach (StorageUnit unit in StorageUnits)

{

classArray.Add(unit.UnitClassification);

}

// You can convert it back to an array if you would like to

string[] classStrings = classArray.ToArray();

classStrings = classStrings.Distinct().ToArray();

for (int x = 0; x < classStrings.Length; x++)

{

CbLeaseSelectClass.Items.Add(classStrings[x]);

}

CbLeaseSelectClass.SelectedIndex = 0;

}

private int CountAvailableUnits(string unitClass) //returns number of available units

{

int availableUnits = 0;

foreach (StorageUnit unit in StorageUnits)

{

if (unit.UnitOccupied == Convert.ToBoolean(0) && unit.UnitClassification.Equals(unitClass))

{

availableUnits ++;

//COME BACK TO THIS

}

}

return availableUnits;

}

private void cb\_addClass\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

try

{

foreach (StorageUnit unit in \_suObjects)

{

if (unit.UnitClassification == cb\_addClass.SelectedValue.ToString())

{

lb\_currentPrice.Content = "R" + unit.UnitPrice;

char[] charSize = unit.UnitSize.ToCharArray();

lb\_currentDimensions.Content = "Width : " + charSize[0] + "m ; "

+ "Length : " + charSize[2] + "m ; "

+ "Height : " + charSize[4] + "m ; ";

\_insertStorageUnit = new StorageUnit();

\_insertStorageUnit.UnitSize = unit.UnitSize;

\_insertStorageUnit.UnitPrice = unit.UnitPrice;

\_insertStorageUnit.UnitArrears = Convert.ToBoolean(0);

\_insertStorageUnit.UnitUpToDate = Convert.ToBoolean(0);

\_insertStorageUnit.UnitInAdvance = Convert.ToBoolean(0);

\_insertStorageUnit.UnitOccupied = Convert.ToBoolean(0);

\_insertStorageUnit.UnitOwnerId = null;

break;

}

}

}

catch (Exception)

{

//Go Home WPF , You're Drunk

}

}

private int CountOccupiedUnits(string unitClass) //returns number of occupied units

{

int occupiedUnits = 0;

foreach (StorageUnit unit in StorageUnits)

{

if (unit.UnitOccupied == Convert.ToBoolean(1) && unit.UnitClassification.Equals(unitClass))

{

occupiedUnits++;

}

}

return occupiedUnits;

}

private void CbLeaseSelectClass\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

try

{

foreach (StorageUnit unit in StorageUnits)

{

if (unit.UnitClassification == CbLeaseSelectClass.SelectedValue.ToString())

{

LbCurrentPrice.Content = "R" + unit.UnitPrice;

char[] charSize = unit.UnitSize.ToCharArray();

LbCurrentDimensions.Content = "Width : " + charSize[0] + "m ; "

+ "Length : " + charSize[2] + "m ; "

+ "Height : " + charSize[4] + "m ; ";

LblAvailableUnits.Content = CountAvailableUnits(CbLeaseSelectClass.SelectedValue.ToString());

break;

}

}

}

catch (Exception)

{

//Go Home WPF , You're Drunk

}

}

private void btn\_addNewUnits\_Click(object sender, RoutedEventArgs e)

{

int rc = 0;

if (!string.IsNullOrEmpty(tb\_noOfNewUnits.Text))

{

try

{

StorageUnit suObject = new StorageUnit();

suObject.UnitClassification = cb\_addClass.SelectedValue.ToString();

suObject.UnitSize = \_insertStorageUnit.UnitSize;

suObject.UnitPrice = \_insertStorageUnit.UnitPrice;

suObject.UnitArrears = \_insertStorageUnit.UnitArrears;

suObject.UnitUpToDate = \_insertStorageUnit.UnitUpToDate;

suObject.UnitInAdvance = \_insertStorageUnit.UnitInAdvance;

suObject.UnitOccupied = \_insertStorageUnit.UnitOccupied;

suObject.UnitOwnerId = "0";

for (int x = 0; x < Convert.ToInt16(tb\_noOfNewUnits.Text); x++)

{

\_suObjects.Clear();

\_suObjects = \_subl.SelectAll();

int max = 0;

foreach (StorageUnit temp in \_suObjects)

{

if (Convert.ToInt16(temp.UnitId) >= max)

{

max = Convert.ToInt16(temp.UnitId);

}

}

suObject.UnitId = Convert.ToString(max + 1);

rc = \_subl.Insert(suObject);

}

if (rc == 0)

{

this.ShowMessageAsync("Success", "Added New Unit/s ");

}

else

{

this.ShowMessageAsync("Error", "Could not Add New Unit");

}

}

catch (Exception exception)

{

MessageBox.Show(exception.ToString());

}

}

else

{

this.ShowMessageAsync("Warning", "Please Enter all options");

}

}

private void BtnLeaseSubmit\_Click(object sender, RoutedEventArgs e)

{

string key = "";

string value = "";

string unitSize = "";

string unitId = "";

string unitClass = "";

List<StorageUnit> storageUnits = \_subl.SelectAll();

Clients = \_cbl.SelectAll();

var lastItemInIndex = storageUnits[storageUnits.Count - 1]; //gets the last item in the list

LeaseUnits leaseUnit = new LeaseUnits();

string lblCurrentPrice = LbCurrentPrice.Content.ToString();

char[] ch = lblCurrentPrice.ToCharArray();

ch[0] = ' '; // index starts at 0! --->> Remove the 'R' character at the 1st position

string newLblCurrentPrice = new string(ch);

//double currentPrice = double.Parse(newLblCurrentPrice);

double currentPrice;

if (double.TryParse(newLblCurrentPrice, out currentPrice))

{

}

else

{

//Do Nothing!!!

}

int rc = 0;

if (!(TxtBoxLeaseId.Text.Equals("") && LeaseName.Text.Equals("") && LeaseSurname.Text.Equals("")))

{

try

{

if (\_cbl.DoesExist(TxtBoxLeaseId.Text))

{

foreach (StorageUnit storageUnit in storageUnits)

{

if (storageUnit.UnitClassification == CbLeaseSelectClass.SelectedValue.ToString() &&

storageUnit.UnitOccupied == Convert.ToBoolean(0))

{

if (!(\_lubl.DoesExist(storageUnit.UnitId)))

{

unitId = storageUnit.UnitId;

unitSize = storageUnit.UnitSize;

unitClass = storageUnit.UnitClassification;

break;

}

}

}

leaseUnit.LeaseID = unitId;

leaseUnit.StorageUnit.UnitId = unitId;

leaseUnit.Client.idNumber = TxtBoxLeaseId.Text;

leaseUnit.Client.FirstName = LeaseName.Text;

leaseUnit.Client.LastName = LeaseSurname.Text;

leaseUnit.StorageUnit.UnitPrice = currentPrice;

leaseUnit.StorageUnit.UnitSize = unitSize;

leaseUnit.StorageUnit.UnitClassification = unitClass;

leaseUnit.TotalUnitPrice = LblTotal.Content.ToString();

leaseUnit.NoOfUnits = int.Parse(TbNoOfNewUnits.Text);

leaseUnit.ClientAdded = Convert.ToBoolean(1);

if (CbLeaseSelectClass.SelectedValue.ToString() == "A")

{

leaseUnit.StorageUnit.UnitClassification = CbLeaseSelectClass.SelectedValue.ToString();

if (int.Parse(TbNoOfNewUnits.Text) >

CountAvailableUnits(CbLeaseSelectClass.SelectedValue.ToString()))

{

this.ShowMessageAsync("Units Occupied", "Units Not Available,Please Enter Less Units");

}

}

else if (CbLeaseSelectClass.SelectedValue.ToString() == "B")

{

leaseUnit.StorageUnit.UnitClassification = CbLeaseSelectClass.SelectedValue.ToString();

if (int.Parse(TbNoOfNewUnits.Text) >

CountAvailableUnits(CbLeaseSelectClass.SelectedValue.ToString()))

{

this.ShowMessageAsync("Units Occupied", "Units Not Available,Please Enter Less Units");

}

}

else if (CbLeaseSelectClass.SelectedValue.ToString() == "C")

{

leaseUnit.StorageUnit.UnitClassification = CbLeaseSelectClass.SelectedValue.ToString();

if (int.Parse(TbNoOfNewUnits.Text) >

CountAvailableUnits(CbLeaseSelectClass.SelectedValue.ToString()))

{

this.ShowMessageAsync("Units Occupied", "Units Not Available,Please Enter Less Units");

}

}

else if (CbLeaseSelectClass.SelectedValue.ToString() == "D")

{

leaseUnit.StorageUnit.UnitClassification = CbLeaseSelectClass.SelectedValue.ToString();

if (int.Parse(TbNoOfNewUnits.Text) >

CountAvailableUnits(CbLeaseSelectClass.SelectedValue.ToString()))

{

this.ShowMessageAsync("Units Occupied", "Units Not Available,Please Enter Less Units");

}

}

else if (CbLeaseSelectClass.SelectedValue.ToString() == "E")

{

leaseUnit.StorageUnit.UnitClassification = CbLeaseSelectClass.SelectedValue.ToString();

if (int.Parse(TbNoOfNewUnits.Text) >

CountAvailableUnits(CbLeaseSelectClass.SelectedValue.ToString()))

{

this.ShowMessageAsync("Units Occupied", "Units Not Available,Please Enter Less Units");

}

}

else if (CbLeaseSelectClass.SelectedValue.ToString() == "E")

{

leaseUnit.StorageUnit.UnitClassification = CbLeaseSelectClass.SelectedValue.ToString();

if (int.Parse(TbNoOfNewUnits.Text) >

CountAvailableUnits(CbLeaseSelectClass.SelectedValue.ToString()))

{

this.ShowMessageAsync("Units Occupied", "Units Not Available,Please Enter Less Units");

}

}

rc = \_lubl.Insert(leaseUnit);

if (rc == 0)

{

this.ShowMessageAsync("Contract Successfully Created", "Contract Will Be Sent To Client!");

TxtBoxLeaseId.Clear();

LeaseName.Clear();

LeaseSurname.Clear();

TbNoOfNewUnits.Clear();

LbCurrentDimensions.Content = ".....";

LbCurrentPrice.Content = ".....";

LblAvailableUnits.Content = ".....";

LblTotal.Content = ".....";

foreach (Client client in Clients)

{

if (client.idNumber.Equals(leaseUnit.Client.idNumber))

{

string fileTemplate = CreateFile("Quotation.docx", "Quotations");

string outputFileName = string.Format(fileTemplate);

//SendEmail(client.EMailAddress, "Please find the attached document!");

DocX letter = GetTemplate(leaseUnit.Client.idNumber);

foreach (KeyValuePair<string, string> pair in dictionary)

{

if (pair.Value == leaseUnit.Client.idNumber)

letter.ReplaceText("ClientID", leaseUnit.Client.idNumber);

else if (pair.Value == leaseUnit.Client.FirstName)

letter.ReplaceText("ClientName", leaseUnit.Client.FirstName);

else if (pair.Value == leaseUnit.Client.LastName)

letter.ReplaceText("ClientSurname", leaseUnit.Client.LastName);

else if (pair.Value == leaseUnit.StorageUnit.UnitClassification)

letter.ReplaceText("UnitClass", leaseUnit.StorageUnit.UnitClassification);

else if (pair.Value == leaseUnit.StorageUnit.UnitSize)

letter.ReplaceText("UnitSize", leaseUnit.StorageUnit.UnitSize);

else if (pair.Value ==

leaseUnit.StorageUnit.UnitPrice.ToString(CultureInfo.InvariantCulture))

letter.ReplaceText("UnitPrice",

leaseUnit.StorageUnit.UnitPrice.ToString(

CultureInfo.InvariantCulture));

else if (pair.Value == leaseUnit.NoOfUnits.ToString())

letter.ReplaceText("NoOfUnits", leaseUnit.NoOfUnits.ToString());

else if (pair.Value == leaseUnit.TotalUnitPrice)

letter.ReplaceText("TotalUnitPrice", leaseUnit.TotalUnitPrice);

}

letter.ReplaceText("ClientID", leaseUnit.Client.idNumber);

letter.ReplaceText("ClientName", leaseUnit.Client.FirstName);

letter.ReplaceText("ClientSurname", leaseUnit.Client.LastName);

letter.ReplaceText("UnitClass", leaseUnit.StorageUnit.UnitClassification);

letter.ReplaceText("UnitSize", leaseUnit.StorageUnit.UnitSize);

letter.ReplaceText("UnitPrice",

leaseUnit.StorageUnit.UnitPrice.ToString(CultureInfo.InvariantCulture));

letter.ReplaceText("NoOfUnits", leaseUnit.NoOfUnits.ToString());

letter.ReplaceText("TotalUnitPrice", leaseUnit.TotalUnitPrice);

letter.SaveAs(outputFileName);

// Open in word:

Process.Start("WINWORD.EXE", "\"" + outputFileName + "\""); //For Debugging Purposes

break;

}

}

}

else

{

this.ShowMessageAsync("Insert Failed", "Record Not Inserted Into Database!");

}

}

else

{

this.ShowMessageAsync("Client Does No Exist", "Please Add a Client!!!");

TxtBoxLeaseId.Clear();

LeaseName.Clear();

LeaseSurname.Clear();

TbNoOfNewUnits.Clear();

LbCurrentDimensions.Content = ".....";

LbCurrentPrice.Content = ".....";

LblAvailableUnits.Content = ".....";

LblTotal.Content = ".....";

}

}

catch (Exception ex)

{

this.ShowMessageAsync("Error", ex.Message);

}

}

else

{

this.ShowMessageAsync("Empty Fields", "Fields Cannot Be Empty!");

}

}

private void SendEmail(string to,string body)

{

string directory = CreateFile("Quotation.docx", "Quotations");

ProgressRingLeaseUnits.IsActive = true;

SmtpMail oMail = new SmtpMail("TryIt");

SmtpClient oSmtp = new SmtpClient();

// Set sender email address, please change it to yours

oMail.From = "onesandzeroesmail@gmail.com";

// Set recipient email address, please change it to yours

oMail.To = to;

// Set email subject

oMail.Subject = "RE: Contract to be reviewed";

// Set Html body

//oMail.HtmlBody = "<font size=\"5\">This is</font> <font color=\"red\"><b>a test</b></font>";

oMail.HtmlBody = body;

// Your SMTP server address

SmtpServer oServer = new SmtpServer("smtp.gmail.com");

// User and password for ESMTP authentication, if your server doesn't require

// User authentication, please remove the following codes.

oServer.User = "onesandzeroesmail@gmail.com";

oServer.Password = "Onesandzeroes.";

// If your smtp server requires SSL connection, please add this line

oServer.ConnectType = SmtpConnectType.ConnectSSLAuto;

try

{

oSmtp.SendMail(oServer, oMail);

ProgressRingLeaseUnits.IsActive = false;

}

catch (Exception ep)

{

this.ShowMessageAsync("Sending Email Failed", ep.Message);

}

}

private void lvUnitsColumnHeader\_Click(object sender, RoutedEventArgs e)

{

GridViewColumnHeader column = (sender as GridViewColumnHeader);

string sortBy = column.Tag.ToString();

if (\_listViewSortColUnits != null)

{

AdornerLayer.GetAdornerLayer(\_listViewSortColUnits).Remove(\_listViewSortAdornerUnits);

lv\_Units.Items.SortDescriptions.Clear();

}

ListSortDirection newDir = ListSortDirection.Ascending;

if (\_listViewSortColUnits == column && \_listViewSortAdornerUnits.Direction == newDir)

newDir = ListSortDirection.Descending;

\_listViewSortColUnits = column;

\_listViewSortAdornerUnits = new SortAdorner(\_listViewSortColUnits, newDir);

AdornerLayer.GetAdornerLayer(\_listViewSortColUnits).Add(\_listViewSortAdornerUnits);

lv\_Units.Items.SortDescriptions.Add(new SortDescription(sortBy, newDir));

}

private void lvUnitsSearchColumnHeader\_Click(object sender, RoutedEventArgs e)

{

GridViewColumnHeader column = (sender as GridViewColumnHeader);

string sortBy = column.Tag.ToString();

if (\_listViewSortColUnits != null)

{

AdornerLayer.GetAdornerLayer(\_listViewSortColUnits).Remove(\_listViewSortAdornerUnits);

lv\_Units.Items.SortDescriptions.Clear();

}

ListSortDirection newDir = ListSortDirection.Ascending;

if (\_listViewSortColUnits == column && \_listViewSortAdornerUnits.Direction == newDir)

newDir = ListSortDirection.Descending;

\_listViewSortColUnits = column;

\_listViewSortAdornerUnits = new SortAdorner(\_listViewSortColUnits, newDir);

AdornerLayer.GetAdornerLayer(\_listViewSortColUnits).Add(\_listViewSortAdornerUnits);

lv\_Units.Items.SortDescriptions.Add(new SortDescription(sortBy, newDir));

}

private void TbNoOfNewUnits\_TextChanged(object sender, TextChangedEventArgs e)

//clears the textbox when there is no input

{

if (TbNoOfNewUnits.Text.Equals(""))

{

LblTotal.Content = ".....";

}

}

private void LeaseId\_TextChanged(object sender, TextChangedEventArgs e)

{

Clients = \_cbl.SelectAll();

foreach (Client temp in Clients)

{

if (TxtBoxLeaseId.Text.Equals(temp.idNumber) && temp.Archived.Equals(Convert.ToBoolean(0)))

{

LeaseName.Text = temp.FirstName;

LeaseSurname.Text = temp.LastName;

break;

}

}

if (TxtBoxLeaseId.Text.Equals(""))

{

LeaseName.Clear();

LeaseSurname.Clear();

}

}

private void TxtBoxListSearch\_TextChanged(object sender, TextChangedEventArgs e)

{

if (TxtBoxListSearch.Text.Equals(""))

{

LvListClient.Items.Clear();

}

}

private string CreateFile(string fileName, string folderName)

{

string directory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\OnesAndZeroes" + "\\" + folderName;

if (!Directory.Exists(directory))

Directory.CreateDirectory(directory);

string path = Path.Combine(directory, fileName);

return path;

}

private DocX GetTemplate(string txtBox)

{

string address = "";

//string fileName = @"C:\\Users\\Watlinton\Documents\\DocXExample.docx";

string fileName = CreateFile("Template.docx","Template");

var doc = DocX.Create(fileName);

LeaseUnits = \_lubl.SelectAll();

Clients = \_cbl.SelectAll();

// Add a Table to this document.

Table table = doc.AddTable(2, 5);

// Specify some properties for this Table.

table.Alignment = Alignment.center;

table.Design = TableDesign.LightGridAccent2;

foreach (Client client in Clients)

{

if (client.idNumber.Equals(txtBox))

{

address = client.Address.Line1 + "\n" +

client.Address.Line2 + "\n" +

client.Address.City + "\n" +

client.Address.Province + "\n" +

client.Address.PostalCode + "\n";

if (!dictionary.ContainsKey("Address"))

{

dictionary.Add("Address", address);

}

break;

}

}

foreach (LeaseUnits leaseUnit in LeaseUnits)

{

if (leaseUnit.Client.idNumber.Equals(txtBox))

{

// Set up our paragraph contents:

string headerText = "Unit Quotation";

string date = "Date: " + DateTime.Now.ToShortDateString();

string time = "Time: " + DateTime.Now.ToShortTimeString();

string clientId = "Client ID:",

clientName = "Client Name: ",

clientSurname = "Client Surname: ";

string clientDetaills = clientId + "\t" + "\t" + leaseUnit.Client.idNumber + "\n"

+ clientName + "\t" + "\t" + leaseUnit.Client.FirstName + "\n"

+ clientSurname + "\t" + leaseUnit.Client.LastName

+ Environment.NewLine;

// Title Formatting:

var titleFormat = new Formatting();

titleFormat.FontFamily = new System.Drawing.FontFamily("Arial Black");

titleFormat.Size = 18D;

titleFormat.FontColor = Color.Black;

titleFormat.Position = 12;

// Body Formatting

var paraFormat = new Formatting();

paraFormat.FontFamily = new System.Drawing.FontFamily("Calibri");

paraFormat.Size = 12D;

titleFormat.Position = 12;

// Variable Formatting

var variableFormat = new Formatting();

variableFormat.FontFamily = new System.Drawing.FontFamily("Tahoma");

variableFormat.Size = 12D;

variableFormat.Position = 12;

variableFormat.Bold = variableFormat.Bold;

// Insert each prargraph, with appropriate spacing and alignment:

Paragraph title = doc.InsertParagraph(headerText, false, titleFormat);

title.Alignment = Alignment.center;

doc.InsertParagraph(Environment.NewLine);

Paragraph dateP = doc.InsertParagraph(date, false, paraFormat);

dateP.Alignment = Alignment.right;

Paragraph timeP = doc.InsertParagraph(time, false, paraFormat);

timeP.Alignment = Alignment.right;

Paragraph addressP = doc.InsertParagraph("1 Park Road" + "\n" +

"Westdene" + "\n" +

"Bloemfontein" + "\n" +

"9301" + "\n" +

"South Africa", false, paraFormat);

addressP.Alignment = Alignment.left;

doc.InsertParagraph(Environment.NewLine);

Paragraph para = doc.InsertParagraph("Thank you for your enquiry and for giving us the opportunity " +

"to quote on the following items:", false, paraFormat);

para.Alignment = Alignment.center;

table.Rows[0].Cells[0].Paragraphs.First().Append("Unit Class").Bold();

table.Rows[0].Cells[1].Paragraphs.First().Append("Unit Size").Bold();

table.Rows[0].Cells[2].Paragraphs.First().Append("Unit Price").Bold();

table.Rows[0].Cells[3].Paragraphs.First().Append("Number of Units").Bold();

table.Rows[0].Cells[4].Paragraphs.First().Append("Total Unit Price (R)").Bold();

table.Rows[1].Cells[0].Paragraphs.First().Append(leaseUnit.StorageUnit.UnitClassification);

table.Rows[1].Cells[1].Paragraphs.First().Append(leaseUnit.StorageUnit.UnitSize);

table.Rows[1].Cells[2].Paragraphs.First().Append(leaseUnit.StorageUnit.UnitPrice.ToString(CultureInfo.InvariantCulture));

table.Rows[1].Cells[3].Paragraphs.First().Append(leaseUnit.NoOfUnits.ToString());

table.Rows[1].Cells[4].Paragraphs.First().Append(leaseUnit.TotalUnitPrice);

doc.InsertTable(table);

doc.InsertParagraph(Environment.NewLine);

Paragraph clientDetails = doc.InsertParagraph("Client Details:", false, paraFormat);

clientDetails.Alignment = Alignment.left;

clientDetails.UnderlineStyle(UnderlineStyle.singleLine);

clientDetails.Bold();

Paragraph clientD = doc.InsertParagraph(clientDetaills, false, paraFormat);

clientD.Alignment = Alignment.left;

Paragraph addressTitle = doc.InsertParagraph("Client Address:", false, paraFormat);

addressTitle.Alignment = Alignment.left;

addressTitle.UnderlineStyle(UnderlineStyle.singleLine);

addressTitle.Bold();

Paragraph addr = doc.InsertParagraph(address, false, paraFormat);

addr.Alignment = Alignment.left;

doc.InsertParagraph(Environment.NewLine);

Paragraph sign = doc.InsertParagraph("Client Signature:", false, paraFormat);

sign.Alignment = Alignment.left;

sign.FontSize(14);

sign.Bold();

if (!(dictionary.ContainsKey("ClientID") || dictionary.ContainsKey("ClientName") || dictionary.ContainsKey("ClientSurname")

|| dictionary.ContainsKey("UnitClass") || dictionary.ContainsKey("UnitSize") || dictionary.ContainsKey("UnitPrice")

|| dictionary.ContainsKey("NoOfUnits") || dictionary.ContainsKey("TotalUnitPrice")))

{

dictionary.Add("ClientID", leaseUnit.Client.idNumber);

dictionary.Add("ClientName", leaseUnit.Client.FirstName);

dictionary.Add("ClientSurname", leaseUnit.Client.LastName);

dictionary.Add("UnitClass", leaseUnit.StorageUnit.UnitClassification);

dictionary.Add("UnitSize", leaseUnit.StorageUnit.UnitSize);

dictionary.Add("UnitPrice", leaseUnit.StorageUnit.UnitPrice.ToString(CultureInfo.InvariantCulture));

dictionary.Add("NoOfUnits", leaseUnit.NoOfUnits.ToString());

dictionary.Add("TotalUnitPrice", leaseUnit.TotalUnitPrice);

}

break;

}

}

return doc;

}

private void BtnCheck\_Click(object sender, RoutedEventArgs e)

{

double total = 0;

string LblCurrentPrice = LbCurrentPrice.Content.ToString();

char[] ch = LblCurrentPrice.ToCharArray();

ch[0] = ' '; // index starts at 0! --->> Remove the 'R' character at the 1st position

string newLblCurrentPrice = new string(ch);

double unitPrice = 0;

double noOfUnits = 0;

string noOfUnits\_ = TbNoOfNewUnits.GetLineText(0);

try

{

if (!TbNoOfNewUnits.Text.Equals(""))

{

foreach (StorageUnit units in StorageUnits)

{

if (units.UnitOccupied == Convert.ToBoolean(0))

{

if (units.UnitPrice == double.Parse(newLblCurrentPrice))

{

unitPrice = units.UnitPrice;

break;

}

}

}

if (double.TryParse(noOfUnits\_, out noOfUnits))

{

LblTotal.Content = unitPrice\*Convert.ToDouble(noOfUnits);

}

else

{

LblTotal.Content = ".....";

}

}

else

{

this.ShowMessageAsync("Field Cannot Be Empty!","Please Enter a Value");

}

}

catch (Exception ex)

{

//this.ShowMessageAsync("Error", ex.Message);

this.ShowMessageAsync("Error!", ex.Message);

LblTotal.Content = ".....";

TbNoOfNewUnits.Clear();

}

}

private void ImgRefreshLease\_MouseDown(object sender, MouseButtonEventArgs e)

{

LeaseUnits = \_lubl.SelectAll();

LvLeaseUnits.Items.Clear();

if (LeaseUnits.Count > 0)

{

LvLeaseUnits.Items.Clear();

foreach (LeaseUnits temp in LeaseUnits)

{

LvLeaseUnits.Items.Add(temp);

}

}

else

{

this.ShowMessageAsync("There are no Clients to list", "No Clients");

}

}

private void BtnEditLeaseUnit\_Click(object sender, RoutedEventArgs e)

{

if (LvLeaseUnits.SelectedIndex >= 0)

{

var unitObj = LvLeaseUnits.SelectedItem as LeaseUnits;

PopUp.LeaseIDTxtBox.Text = unitObj.LeaseID;

PopUp.ClientIDTxtBox.Text = unitObj.Client.idNumber;

PopUp.LeaseNameTxtBox.Text = unitObj.Client.FirstName;

PopUp.LeaseSurnameTxtBox.Text = unitObj.Client.LastName;

PopUp.LeaseOwedTxtBox.Text = unitObj.AmountOwed;

PopUp.LeasePaidTxtBox.Text = unitObj.AmountPaid;

PopUp.LeaseDateTxtBox.Text = unitObj.DateOfPayment;

PopUp.LeaseUnitTxtBox.Text = unitObj.UnitLeased.ToString();

PopUp. LeaseIDTxtBox.IsEnabled = false;

PopUp. ClientIDTxtBox.IsEnabled = false;

PopUp.ShowDialog();

}

else

{

this.ShowMessageAsync("Item Not Selected!", "Please Select an item");

}

}

private void Btn\_ClearLessee\_OnClick(object sender, RoutedEventArgs e)

{

int rc = 0;

StorageUnit selectedUnit = new StorageUnit();

if (lv\_Units.SelectedIndex >= 0)

{

//Get Selected Item as a SU Object , possible because of class binding

var unitObj = lv\_Units.SelectedItem as StorageUnit;

string selectedID = unitObj.UnitId;

rc = \_subl.SelectStorageUnit(selectedID, ref selectedUnit);

if (rc != 0)

{

this.ShowMessageAsync("Error", "Could Not Find Storage Unit ... \n Please Refresh Unit List ");

}

else

{

selectedUnit.UnitOccupied = false;

selectedUnit.UnitOwnerId = "0";

rc = \_subl.Update(selectedUnit);

if (rc != 0)

{

this.ShowMessageAsync("Error", "Could not Remove Client from Unit");

}

else

{

LeaseUnits = \_lubl.SelectAll();

foreach (LeaseUnits leaseUnit in LeaseUnits)

{

if (leaseUnit.StorageUnit.UnitId.Equals(selectedUnit.UnitId))

{

rc = \_lubl.Delete(leaseUnit.LeaseID);

if (rc != 0)

{

this.ShowMessageAsync("Error", "Could not Delete Lease Information");

}

else

{

this.ShowMessageAsync("Success", "Removed Leasing Information for selected Unit");

}

}

}

}

}

}

else

{

this.ShowMessageAsync("Warning", "Please Choose a Unit in the List");

}

}

private void btn\_UnitListSearch\_Copy\_OnClick(object sender, RoutedEventArgs e)

{

int rc = 0;

StorageUnit unitObject = new StorageUnit();

lv\_Units\_Search.Items.Clear();

rc = \_subl.SelectStorageUnit(tb\_SearchUnit.Text, ref unitObject);

if (rc == 0)

{

lv\_Units\_Search.Items.Add(unitObject);

}

else

{

this.ShowMessageAsync("Error", "No Matching Unit ID Found");

} //end else

} //end if

}//end method

}//end class

Business Access layer

Classes folder

Address Class

namespace StorageUnitManagementSystem.BL.Classes

{

public class Address

{

private string \_line1; //'x'x All correct according to UML/specs

private string \_line2;

private string \_city;

private string \_province;

private string \_postalCode;

public string Line1 //'x Line1 according to UML/specs

{

//

//Property Name : property string Line1

//Purpose : Public property to give access to \_line1 instance variable

//Re-use : none

//Input : string value

// - the user supplied \_line1

//Output : string

// - value stored in \_line1 instance variable

//

get { return \_line1; } // end get

set { \_line1 = value; } // end set

} // end property

public string Line2 //'x Line2 according to UML/specs

{

//

//Property Name : property string Line2

//Purpose : Public property to give access to \_line2 instance variable

//Re-use : none

//Input : string value

// - the user supplied \_line2

//Output : string

// - value stored in \_line2 instance variable

//

get { return \_line2; } // end get

set { \_line2 = value; } // end set

} // end property

public string City //'x City according to UML/specs

{

//

//Property Name : property string City

//Purpose : Public property to give access to \_city instance variable

//Re-use : none

//Input : string value

// - the user supplied \_city

//Output : string

// - value stored in \_city instance variable

//

get { return \_city; } // end get

set { \_city = value; } // end set

} // end property

public string Province //'x Province according to UML/specs

{

//

//Property Name : property string Province

//Purpose : Public property to give access to \_province instance variable

//Re-use : none

//Input : string value

// - the user supplied \_province

//Output : string

// - value stored in \_province instance variable

//

get { return \_province; } // end get

set { \_province = value; } // end set

} // end property

public string PostalCode //'x PostalCode according to UML/specs

{

//

//Property Name : property string PostalCode

//Purpose : Public property to give access to \_postalCode instance variable

//Re-use : none

//Input : string value

// - the user supplied \_postalCode

//Output : string

// - value stored in \_postalCode instance variable

//

get { return \_postalCode; } // end get

set { \_postalCode = value; } // end set

} // end property

public Address() //'x Correct empty default const according to UML/specs

{

//

//Method Name : Address()

//Purpose : Empty default constructor

//Re-use : none

//Input : None

//Output : None

this.Line1 = Line1;

this.Line2 = Line2;

this.City = City;

this.Province = Province;

this.PostalCode = PostalCode;

//

} // end method

//'x 1st oc: header according to UML/specs

public Address(string Line1, string Line2, string City, string Province, string PostalCode)

{

//

//Method Name : Address(string Line1, string Line2, string Province, string PostalCode)

//Purpose : Update instance variables with user supplied values (method parameters)

//Re-use : none

//Input : - string Line1

// - user supplied value to update instance variable

// - string Line2

// - user supplied value to update instance variable

// - string City

// - user supplied value to update instance variable

// - string Province

// - user supplied value to update instance variable

// - string PostalCode

// - user supplied value to update instance variable

//Output : None

//

this.Line1 = Line1;

this.Line2 = Line2;

this.City = City;

this.Province = Province;

this.PostalCode = PostalCode;

} // end method

public Address(string Line1, string Line2, string City)

: this(Line1, Line2, City, "FS", "9301")

{

//

//Method Name : Address(string Line1, string Line2)

//Purpose : Update instance variables with user supplied values (method parameters);

// assign default values to \_province and \_postalCode

//Re-use : Address(string Line1, string Line2, string Province, string PostalCode)

//Input : - string Line1

// - user supplied value to update instance variable

// - string Line2

// - user supplied value to update instance variable

// - string City

// - user supplied value to update instance variable

//Output : None

//

} // end method

public override string ToString()

{

//

//Method Name : string ToString()

//Purpose : Returns a string containing all the Address info

//Re-use : none

//Input : none

//Output : string

// - meaningful string containing all the Address info

//

return \_line1 + ", " + \_city + ", " + \_province + ", " + \_postalCode;//deleted line 2"

} // end method

}

}

Client Class

namespace StorageUnitManagementSystem.BL.Classes

{

public class Client

{

private string \_idNumber;

private string \_firstName;

private string \_lastName;

private string \_dateOfBirth;

private string \_cellphone;

private string \_telephone;

private bool \_archived;

private string \_eMailAddress;

private Address \_address;

public Client()

{

\_address = new Address();

}

public string idNumber

{

get { return \_idNumber; }

set{ \_idNumber = value;}

}

public string FirstName

{

get { return \_firstName; }

set { \_firstName = value; }

}

public string LastName

{

get { return \_lastName; }

set { \_lastName = value; }

}

public string DateOfBirth

{

get { return \_dateOfBirth; }

set { \_dateOfBirth = value; }

}

public string Cellphone

{

get { return \_cellphone; }

set { \_cellphone = value; }

}

public string Telephone

{

get { return \_telephone; }

set { \_telephone = value; }

}

public Address Address

{

get { return \_address; }

set { \_address = value; }

}

public string EMailAddress

{

get { return \_eMailAddress; }

set { \_eMailAddress = value; }

}

public bool Archived

{

get { return \_archived; }

set { \_archived = value; }

}

}

}

Create database class

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SQLite;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace StorageUnitManagementSystem.BL.Classes

{

public class CreateDatabase

{

public CreateDatabase()

{

CreateDb();

}

public static string Dir { get; } = System.IO.Path.Combine(Environment.GetFolderPath(

Environment.SpecialFolder.ApplicationData), "StorageUnitManagementDB.db");

public static string ConStr { get; private set; } = "Data Source=" + Dir + ";Version=3;";

public static SQLiteConnection SqlCon { get; private set; }

public static void CreateTableClients()

{

try

{

SqlCon = new SQLiteConnection(ConStr);

SqlCon.Open();

string query = "CREATE TABLE Clients (clientID TEXT PRIMARY KEY NOT NULL,"

+ "clientFirstNames TEXT,"

+ "clientLastName TEXT,"

+ "clientDateOfBirth TEXT,"

+ "clientCellPhone TEXT,"

+ "clientEmail TEXT,"

+ "clientTelephone TEXT,"

+ "clientALine1 TEXT,"

+ "clientALine2 TEXT,"

+ "clientACity TEXT,"

+ "clientAProvince TEXT,"

+ "clientPostalCode TEXT ,"

+ "clientArchived INTEGER DEFAULT 0 ,"

+ "unitId TEXT); ";

SQLiteCommand sqlCommand = new SQLiteCommand(query, SqlCon); // setup command

sqlCommand.ExecuteNonQuery();

}

catch (Exception ex)

{

throw ex;

}

finally

{

SqlCon.Close(); // Close connection

} // end finally

}

public static void CreateTableLeaseUnits()

{

try

{

SqlCon = new SQLiteConnection(ConStr);

SqlCon.Open();

string query = "CREATE TABLE LeaseUnits (LeaseID TEXT PRIMARY KEY NOT NULL,"

+ "ClientID TEXT,"

+ "ClientName TEXT,"

+ "ClientSurname TEXT,"

+ "UnitID TEXT,"

+ "UnitClass TEXT,"

+ "UnitPrice NUMERIC DEFAULT 0,"

+ "NoOfUnits INT DEFAULT 0,"

+ "ClientWaitingList INT DEFAULT 0,"

+ "AvailableUnits TEXT,"

+ "TypeOfPayment TEXT,"

+ "DatePaid TEXT,"

+ "DateOfContractStart TEXT,"

+ "DateOfContractEnd TEXT,"

+ "AmountDeposited TEXT,"

+ "AmountOwed TEXT,"

+ "AmountPaid TEXT,"

+ "ClientCurrentTotal TEXT,"

+ "UnitLeased INT DEFAULT 0,"

+ "ClientAdded INT DEFAULT 0,"

+ "TotalUnitPrice TEXT,"

+ "UnitSize TEXT); ";

SQLiteCommand sqlCommand = new SQLiteCommand(query, SqlCon); // setup command

sqlCommand.ExecuteNonQuery();

}

catch (Exception ex)

{

throw ex;

}

finally

{

SqlCon.Close(); // Close connection

} // end finally

}

public static void CreateTableStorageUnits()

{

try

{

SqlCon = new SQLiteConnection(ConStr);

SqlCon.Open();

string query = "CREATE TABLE StorageUnits (suID TEXT PRIMARY KEY NOT NULL,"

+ "suClassification TEXT,"

+ "suPrice NUMERIC,"

+ "suSize TEXT,"

+ "suArrears INT DEFAULT 0,"

+ "suOccupied INT DEFAULT 0,"

+ "suAdvance INT DEFAULT 0,"

+ "suUpToDate INT DEFAULT 0,"

+ "suOwnerID TEXT DEFAULT '0'); ";

SQLiteCommand sqlCommand = new SQLiteCommand(query, SqlCon); // setup command

sqlCommand.ExecuteNonQuery();

}

catch (Exception ex)

{

throw ex;

}

finally

{

SqlCon.Close(); // Close connection

} // end finally

}

public static void CreateTableUsers()

{

try

{

SqlCon = new SQLiteConnection(ConStr);

SqlCon.Open();

string query = "CREATE TABLE Users (UId TEXT DEFAULT '0' PRIMARY KEY NOT NULL,"

+ "UName TEXT DEFAULT 'ADMIN',"

+ "USurname TEXT DEFAULT 'ADMIN',"

+ "UPassword TEXT DEFAULT 'ADMIN',"

+ "UPosition TEXT DEFAULT 'ADMIN');";

SQLiteCommand sqlCommand = new SQLiteCommand(query, SqlCon); // setup command

sqlCommand.ExecuteNonQuery();

}

catch (Exception ex)

{

throw ex;

}

finally

{

SqlCon.Close(); // Close connection

} // end finally

}

public static void CreateDb()

{

ConStr = "Data Source=" + Dir + ";Version=3;";

if (!File.Exists(Dir))

{

SQLiteConnection.CreateFile(Dir);

if (TableExists("Clients") == false && TableExists("StorageUnits") == false &&

TableExists("LeaseUnits") == false && TableExists("Users") == false)

{

CreateTableClients();

CreateTableStorageUnits();

CreateTableLeaseUnits();

CreateTableUsers();

InsertUsers();

for (int i = 1; i < 47; i++)

{

Insert(i.ToString(), "A", 650, "3,3,3");

}

for (int i = 47; i < 68; i++)

{

Insert(i.ToString(), "B", 750, "3,5,3");

}

for (int i = 68; i < 84; i++)

{

Insert(i.ToString(), "C", 950, "3,5,5");

}

for (int i = 84; i < 95; i++)

{

Insert(i.ToString(), "D", 1150, "3,7,3");

}

for (int i = 95; i < 106; i++)

{

Insert(i.ToString(), "E", 1250, "3,7,5");

}

for (int i = 106; i < 111; i++)

{

Insert(i.ToString(), "F", 400, "5,6,4");

}

}

}

}

public static void Insert(string unitID, string unitClass, double unitPrice, string unitSize)

{

int rc = 0;

try

{

int rowsAffected = 0;

//TO:DO

SqlCon = new SQLiteConnection(ConStr); // new connection

SqlCon.Open(); // open connection

string insertQuery = "INSERT INTO StorageUnits([suID], [suClassification], [suPrice], " +

"[suSize]) VALUES(" +

"@suID, @suClassification, @suPrice, @suSize)";

SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, SqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@suID", DbType.String),

new SQLiteParameter("@suClassification", DbType.String),

new SQLiteParameter("@suPrice", DbType.VarNumeric),

new SQLiteParameter("@suSize", DbType.String),

};

sqlParams[0].Value = unitID; // Populate SQLiteParameters from StorageUnit

sqlParams[1].Value = unitClass;

sqlParams[2].Value = unitPrice;

sqlParams[3].Value = unitSize;

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 1) // Test rowsAffected

{

// 1 row affected, thus 1 row added to datastore, thus success

rc = 0;

}

else

{

rc = -1;

} // end if

}

catch (Exception ex)

{

throw ex;

}

finally

{

SqlCon.Close();

}

}

public static void InsertUsers()

{

int rc = 0;

try

{

int rowsAffected = 0;

//TO:DO

SqlCon = new SQLiteConnection(ConStr); // new connection

SqlCon.Open(); // open connection

string insertQuery = "INSERT INTO Users([UId], [UName],[USurname], [UPassword], " +

"[UPosition]) VALUES(" +

"@UId, @UName, @USurname,@UPassword, @UPosition)";

SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, SqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@UId", DbType.String),

new SQLiteParameter("@UName", DbType.String),

new SQLiteParameter("@USurname", DbType.String),

new SQLiteParameter("@UPassword", DbType.String),

new SQLiteParameter("@UPosition", DbType.String)

};

sqlParams[0].Value = "ADMIN"; // Populate SQLiteParameters from StorageUnit

sqlParams[1].Value = "ADMIN";

sqlParams[2].Value = "ADMIN";

sqlParams[3].Value = "ADMIN";

sqlParams[4].Value = "ADMIN";

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 1) // Test rowsAffected

{

// 1 row affected, thus 1 row added to datastore, thus success

rc = 0;

}

else

{

rc = -1;

} // end if

}

catch (Exception ex)

{

throw ex;

}

finally

{

SqlCon.Close();

}

}

public static bool TableExists(String tableName)

{

bool rc = false;

try

{

SqlCon = new SQLiteConnection(ConStr);

SqlCon.Open();

using (SQLiteCommand cmd = new SQLiteCommand())

{

cmd.CommandType = CommandType.Text;

cmd.Connection = SqlCon;

cmd.CommandText = "SELECT \* FROM sqlite\_master WHERE type = 'table' AND name = @name";

cmd.Parameters.AddWithValue("@name", tableName);

using (SQLiteDataReader sqlDataReader = cmd.ExecuteReader())

{

if (sqlDataReader.Read())

rc = true;

else

rc = false;

}

}

}

catch (Exception ex)

{

throw ex;

}

finally

{

SqlCon.Close(); // Close connection

}

return rc;

}

}//end method

}//end class

Lease Unit

namespace StorageUnitManagementSystem.BL.Classes

{

public class LeaseUnits

{

public LeaseUnits()

{

Client = new Client();

StorageUnit = new StorageUnit();

}

public Client Client { get; set; }

public StorageUnit StorageUnit { get; set; }

public string LeaseID { get; set; }

public int NoOfUnits { get; set; }

public string AvailableUnits { get; set; }

public string TypeOfPayment { get; set; }

public string DateOfPayment { get; set; }

public string DateOfContractStart { get; set; }

public string DateOfContractEnd { get; set; }

public string AmountPaid { get; set; }

public string AmountOwed { get; set; }

public string AmountDeposited { get; set; }

public string ClientCurrentTotal { get; set; }

public bool ClientWaitingList { get; set; }

public bool UnitLeased { get; set; }

public bool ClientAdded { get; set; }

public string TotalUnitPrice { get; set; }

}

}//end class

Sort Adoner Class

using System.ComponentModel;

using System.Windows;

using System.Windows.Documents;

using System.Windows.Media;

namespace StorageUnitManagementSystem.BL.Classes

{//

public class

SortAdorner : Adorner

{

private static Geometry \_ascGeometry =

Geometry.Parse("M 0 4 L 3.5 0 L 7 4 Z");

private static Geometry \_descGeometry =

Geometry.Parse("M 0 0 L 3.5 4 L 7 0 Z");

public ListSortDirection Direction { get; private set; }

public SortAdorner(UIElement element, ListSortDirection dir)

: base(element)

{

this.Direction = dir;

}

protected override void OnRender(DrawingContext drawingContext)

{

base.OnRender(drawingContext);

if (AdornedElement.RenderSize.Width < 20)

return;

TranslateTransform transform = new TranslateTransform

(

AdornedElement.RenderSize.Width - 15,

(AdornedElement.RenderSize.Height - 5) / 2

);

drawingContext.PushTransform(transform);

Geometry geometry = \_ascGeometry;

if (this.Direction == ListSortDirection.Descending)

geometry = \_descGeometry;

drawingContext.DrawGeometry(Brushes.Black, null, geometry);

drawingContext.Pop();

}

}

}//end class

Storage Unit.cs

namespace StorageUnitManagementSystem.BL.Classes

{

public class StorageUnit

{

private string \_unitClassification;

private string \_unitSize;

private double \_unitPrice;

private bool \_unitArrears;

private bool \_unitUpToDate;

private bool \_unitInAdvance;

private bool \_unitOccupied;

private string \_unitOwnerId;

private string \_unitId;

public StorageUnit()

{

}

public bool UnitOccupied

{

get { return \_unitOccupied; }

set { \_unitOccupied = value; }

}

public bool UnitArrears

{

get { return \_unitArrears; }

set { \_unitArrears = value; }

}

public bool UnitInAdvance

{

get { return \_unitInAdvance; }

set { \_unitInAdvance = value; }

}

public bool UnitUpToDate

{

get { return \_unitUpToDate; }

set { \_unitUpToDate = value; }

}

public string UnitOwnerId

{

get { return \_unitOwnerId; }

set { \_unitOwnerId = value; }

}

public string UnitId

{

get { return \_unitId; }

set { \_unitId = value; }

}

public string UnitClassification

{

get { return \_unitClassification; }

set { \_unitClassification = value; }

}

public string UnitSize

{

get { return \_unitSize; }

set { \_unitSize = value; }

}

public double UnitPrice

{

get { return \_unitPrice; }

set { \_unitPrice = value; }

}

}

}//end class

User.cs

namespace StorageUnitManagementSystem.BL.Classes

{

public class User

{

private string \_id;

private string userName;

private string password;

private string role;

public string UserName

{

get

{

return userName;

}

set

{

userName = value;

}

}

public string Password

{

get

{

return password;

}

set

{

password = value;

}

}

public string Role

{

get

{

return role;

}

set

{

role = value;

}

}

public string Id

{

get

{

return \_id;

}

set

{

\_id = value;

}

}

}

}//end class

LabelBinding.cs

namespace StorageUnitManagementSystem.BL.Binding

{

class LabelBinding : INotifyPropertyChanged

{

#region INotifyPropertyChanged implementation

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

}

#endregion

private string \_myContent;

public string MyContent

{

get

{

return \_myContent;

}

set

{

if (\_myContent == value)

return;

\_myContent = value;

OnPropertyChanged("MyContent");

}

}

}

}//end class

CBL.cs

namespace StorageUnitManagementSystem.BL

{

class CBL

{

private ClientProviderBase providerBase;

public CBL(string Provider)

{

//

//Method name : SUBL(string Provider)

//Purpose : Overloaded constructor; invoke \_SetupProviderBase to setup data provider

//Re-use : \_SetupProviderBase()

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

\_SetupProviderBase(Provider);

} // end method

/// <summary>

/// This method gets the list of all the business objects from the Client datastore.

/// It returns the list of business objects

/// </summary>

public List<Client> SelectAll()

{

return providerBase.SelectAll();

} // end method

public List<Client> SelectAll(string name)//new Method for returning record using the name field

{

return providerBase.SelectAll();

} // end method

/// <summary>

/// This method gets a single Client object from the Client datastore.

/// </summary>

/// <param name="ID">The Client ID of the Client to load from the datastore.</param>

/// <param name="Client">The Client object loaded from the datastore.</param>

public int SelectClient(string ID, ref Client Client)

{

return providerBase.SelectClient(ID, ref Client);

} // end method

public int SelectClientName(string Name, ref Client Client)

{

return providerBase.SelectClient(Name, ref Client);

} // end method

/// <summary>

/// This method inserts a row in the Client datastore

/// </summary>

/// <param name="Client">The Client object to add to the Client datastore.</param>

public int Insert(Client Client)

{

return providerBase.Insert(Client);

} // end method

/// <summary>

/// This method updates a row in the Client datastore.

/// </summary>

/// <param name="Client">The new Client data for the row in the Client datastore.</param>

public int Update(Client Client)

{

return providerBase.Update(Client);

} // end method

/// <summary>

/// This method deletes a row in the Client datastore.

/// </summary>

/// <param name="ID">The Client ID of the Client to delete in the Client datastore.</param>

public int Delete(string ID)

{

return providerBase.Delete(ID);

} // end method

/// <summary>

/// This method determines if a given Client exists in the Client datastore.

/// It returns true to indicate the Client was foundy, or

/// false to indicate the Client was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to search in the Client datastore.</param>

public bool DoesExist(string ID)

{

return providerBase.DoesExist(ID);

} // end method

private void \_SetupProviderBase(string Provider)

{

//

//Method name : void \_SetupProviderBase()

//Purpose : Helper method to select the correct data provider

//Re-use : None

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

if (Provider == "ClientSQLiteProvider")

{

providerBase = new ClientSQLiteProvider();

} // end if

} // end method

}

}//end namespace

LUBL.cs

namespace StorageUnitManagementSystem.BL

{

class LUBL

{

private LeaseUnitsProviderBase providerBase;

public LUBL(string Provider)

{

//

//Method Name : SUBL(string Provider)

//Purpose : Overloaded constructor; invoke \_SetupProviderBase to setup data provider

//Re-use : \_SetupProviderBase()

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

\_SetupProviderBase(Provider);

} // end method

/// <summary>

/// This method gets the list of all the business objects from the StorageUnit datastore.

/// It returns the list of business objects

/// </summary>

public List<LeaseUnits> SelectAll()

{

return providerBase.SelectAll();

} // end method

/// <summary>

/// This method gets a single StorageUnit object from the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was loaded from the datastore, or

/// -1 to indicate that no StorageUnit was loaded from the datastore.

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to load from the datastore.</param>

/// <param name="StorageUnit">The StorageUnit object loaded from the datastore.</param>

public int SelectLeaseUnit(string ID, ref LeaseUnits LeaseUnit)

{

return providerBase.SelectLeaseUnit(ID, ref LeaseUnit);

} // end method

/// <summary>

/// This method inserts a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was inserted into datastore, or

/// -1 to indicate the StorageUnit was not inserted because a duplicate was found

/// </summary>

/// <param name="StorageUnit">The StorageUnit object to add to the StorageUnit datastore.</param>

public int Insert(LeaseUnits LeaseUnit)

{

return providerBase.Insert(LeaseUnit);

} // end method

/// <summary>

/// This method updates a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was found and updated successfully, or

/// -1 to indicate the StorageUnit was not updated because the record was not found

/// </summary>

/// <param name="StorageUnit">The new StorageUnit data for the row in the StorageUnit datastore.</param>

public int Update(LeaseUnits LeaseUnit)

{

return providerBase.Update(LeaseUnit);

} // end method

public int UpdatePopUp(LeaseUnits LeaseUnit)

{

return providerBase.Update(LeaseUnit);

} // end method

/// <summary>

/// This method deletes a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was found and deleted successfully, or

/// -1 to indicate the StorageUnit was not deleted because the record was not found

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to delete in the StorageUnit datastore.</param>

public int Delete(string ID)

{

return providerBase.Delete(ID);

} // end method

/// <summary>

/// This method determines if a given StorageUnit exists in the StorageUnit datastore.

/// It returns true to indicate the StorageUnit was foundy, or

/// false to indicate the StorageUnit was not found

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to search in the StorageUnit datastore.</param>

public bool DoesExist(string ID)

{

return providerBase.DoesExist(ID);

} // end method

private void \_SetupProviderBase(string Provider)

{

//

//Method Name : void \_SetupProviderBase()

//Purpose : Helper method to select the correct data provider

//Re-use : None

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

if (Provider == "LeaseUnitsSQLiteProvider")

{

providerBase = new LeaseUnitsSQLiteProvider();

} // end if

else

{

//if (Provider == "StorageUnitXMLProvider")

//{

// providerBase = new StorageUnitXMLProvider();

//} // end if

// else

// {

// if (Provider == "StorageUnitCSVProvider")

// {

// providerBase = new StorageUnitCSVProvider();

// } // end if

// } // end else

} // end else

} // end method

}

}

SUBL.cs

namespace StorageUnitManagementSystem.BL

{

public class SUBL

{

private StorageUnitManagementProviderBase providerBase;

public SUBL(string Provider)

{

//

//Method Name : SUBL(string Provider)

//Purpose : Overloaded constructor; invoke \_SetupProviderBase to setup data provider

//Re-use : \_SetupProviderBase()

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

\_SetupProviderBase(Provider);

} // end method

/// <summary>

/// This method gets the list of all the business objects from the StorageUnit datastore.

/// It returns the list of business objects

/// </summary>

public List<StorageUnit> SelectAll()

{

return providerBase.SelectAll();

} // end method

/// <summary>

/// This method gets a single StorageUnit object from the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was loaded from the datastore, or

/// -1 to indicate that no StorageUnit was loaded from the datastore.

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to load from the datastore.</param>

/// <param name="StorageUnit">The StorageUnit object loaded from the datastore.</param>

public int SelectStorageUnit(string ID, ref StorageUnit StorageUnit)

{

return providerBase.SelectStorageUnit(ID, ref StorageUnit);

} // end method

/// <summary>

/// This method inserts a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was inserted into datastore, or

/// -1 to indicate the StorageUnit was not inserted because a duplicate was found

/// </summary>

/// <param name="StorageUnit">The StorageUnit object to add to the StorageUnit datastore.</param>

public int Insert(StorageUnit StorageUnit)

{

return providerBase.Insert(StorageUnit);

} // end method

/// <summary>

/// This method updates a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was found and updated successfully, or

/// -1 to indicate the StorageUnit was not updated because the record was not found

/// </summary>

/// <param name="StorageUnit">The new StorageUnit data for the row in the StorageUnit datastore.</param>

public int Update(StorageUnit StorageUnit)

{

return providerBase.Update(StorageUnit);

} // end method

/// <summary>

/// This method deletes a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was found and deleted successfully, or

/// -1 to indicate the StorageUnit was not deleted because the record was not found

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to delete in the StorageUnit datastore.</param>

public int Delete(string ID)

{

return providerBase.Delete(ID);

} // end method

/// <summary>

/// This method determines if a given StorageUnit exists in the StorageUnit datastore.

/// It returns true to indicate the StorageUnit was foundy, or

/// false to indicate the StorageUnit was not found

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to search in the StorageUnit datastore.</param>

public bool DoesExist(string ID)

{

return providerBase.DoesExist(ID);

} // end method

private void \_SetupProviderBase(string Provider)

{

//

//Method Name : void \_SetupProviderBase()

//Purpose : Helper method to select the correct data provider

//Re-use : None

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

if (Provider == "StorageUnitSQLiteProvider")

{

providerBase = new StroageUnitManagementSQLiteProvider();

} // end if

else

{

//if (Provider == "StorageUnitXMLProvider")

//{

// providerBase = new StorageUnitXMLProvider();

//} // end if

// else

// {

// if (Provider == "StorageUnitCSVProvider")

// {

// providerBase = new StorageUnitCSVProvider();

// } // end if

// } // end else

} // end else

} // end method

}//end class

}//end namespace

UBL.cs

using StorageUnitManagementSystem.DAL;

using StorageUnitManagementSystem.BL.Classes;

using System.Collections.Generic;

namespace StorageUnitManagementSystem.BL

{

class UBL

{

private UserProviderBase providerBase;

public UBL(string Provider)

{

//

//Method name : SUBL(string Provider)

//Purpose : Overloaded constructor; invoke \_SetupProviderBase to setup data provider

//Re-use : \_SetupProviderBase()

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

\_SetupProviderBase(Provider);

} // end method

/// <summary>

/// This method gets the list of all the business objects from the Client datastore.

/// It returns the list of business objects

/// </summary>

public List<User> SelectAll()

{

return providerBase.SelectAll();

} // end method

public List<User> SelectAll(string name)//new Method for returning record using the name field

{

return providerBase.SelectAll();

} // end method

/// <summary>

/// This method gets a single Client object from the Client datastore.

/// </summary>

/// <param name="ID">The Client ID of the Client to load from the datastore.</param>

/// <param name="Client">The Client object loaded from the datastore.</param>

public int SelectClient(string ID, ref User User)

{

return providerBase.SelectUser(ID, ref User);

} // end method

public int SelectClientName(string Name, ref User User)

{

return providerBase.SelectUser(Name, ref User);

} // end method

/// <summary>

/// This method inserts a row in the Client datastore

/// </summary>

/// <param name="Client">The Client object to add to the Client datastore.</param>

public int Insert(User User)

{

return providerBase.Insert(User);

} // end method

/// <summary>

/// This method updates a row in the Client datastore.

/// </summary>

/// <param name="Client">The new Client data for the row in the Client datastore.</param>

public int Update(User User)

{

return providerBase.Update(User);

} // end method

/// <summary>

/// This method deletes a row in the Client datastore.

/// </summary>

/// <param name="ID">The Client ID of the Client to delete in the Client datastore.</param>

public int Delete(string ID)

{

return providerBase.Delete(ID);

} // end method

/// <summary>

/// This method determines if a given Client exists in the Client datastore.

/// It returns true to indicate the Client was foundy, or

/// false to indicate the Client was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to search in the Client datastore.</param>

public bool DoesExist(string ID)

{

return providerBase.DoesExist(ID);

} // end method

private void \_SetupProviderBase(string Provider)

{

//

//Method name : void \_SetupProviderBase()

//Purpose : Helper method to select the correct data provider

//Re-use : None

//Input : string Provider

// - The name of the data provider to use

//Output : None

//

if (Provider == "UserSQLiteProvider")

{

providerBase = new UserSQLiteProvider();

} // end if

} // end method

}//end class

}//end namespace

ClientProviderBase.cs

namespace ClientManagementSystem.DAL

{

public abstract class ClientProviderBase

{

/// <summary>

/// This method gets the list of all the business objects from the Client datastore.

/// It returns the list of business objects

/// </summary>

public abstract List<Client> SelectAll();

/// <summary>

/// This method gets a single Client object from the Client datastore.

/// It returns 0 to indicate the Client was loaded from datastore, or

/// -1 to indicate that no Client was loaded from the datastore (not found).

/// </summary>

/// <param name="ID">The ID of the Client to load from the datastore.</param>

/// <param name="Client">The Client object loaded from the datastore.</param>

public abstract int SelectClient(string ID, ref Client Client);

/// <summary>

/// This method inserts a row in the Client datastore.

/// It returns 0 to indicate the Client was inserted into datastore, or

/// -1 to indicate the Client was not inserted because a duplicate was found

/// </summary>

/// <param name="Client">The Client object to add to the Client datastore.</param>

public abstract int Insert(Client Client);

/// <summary>

/// This method updates a row in the Client datastore.

/// It returns 0 to indicate the Client was found and updated successfully, or

/// -1 to indicate the Client was not updated because the record was not found

/// </summary>

/// <param name="Client">The new Client data for the row in the Client datastore.</param>

public abstract int Update(Client Client);

/// <summary>

/// This method deletes a row in the Client datastore.

/// It returns 0 to indicate the Client was found and deleted successfully, or

/// -1 to indicate the Client was not deleted because the record was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to delete in the Client datastore.</param>

public abstract int Delete(string ID);

/// <summary>

/// This method determines if a given Client exists in the Client datastore.

/// It returns true to indicate the Client was foundy, or

/// false to indicate the Client was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to search in the Client datastore.</param>

public abstract bool DoesExist(string ID);

}//end class

}//end namespace

clientSQLProvider

namespace ClientManagementSystem.DAL

{

public class ClientSQLiteProvider : ClientProviderBase

{

private string \_conStr = CreateDatabase.ConStr;

private SQLiteConnection \_sqlCon;

public override List<Client> SelectAll()

{

//

//Method Name : List<Client> SelectAll()

//Purpose : Try to get all the Client objects from the datastore

//Re-use : None

//Input : None

//Output : - ref List<Client>

// - the list that will contain the Client objects loaded from datastore

//

List<Client> Clients; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

Clients = new List<Client>(); // this ensures that if there are no records,

// the returned list will not be null, but

// it will be empty (Count = 0)

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM Clients";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read(); // Priming read (must have 2nd read in loop)

while (bRead == true) // false indicates no more rows/records4\4

{

Client Client = new Client();

//unit.Address = new Address();

//unit.Phone = new Phone();

Client.Address = new Address();

Client.idNumber = Convert.ToString(sdr["clientID"]);

Client.FirstName = Convert.ToString(sdr["clientFirstNames"]);

Client.LastName = Convert.ToString(sdr["clientLastName"]);

Client.DateOfBirth = Convert.ToString(sdr["clientDateOfBirth"]);

Client.Cellphone = Convert.ToString(sdr["clientCellphone"]);

Client.EMailAddress = Convert.ToString(sdr["clientEmail"]);

Client.Telephone = Convert.ToString(sdr["clientTelephone"]);

Client.Address.Line1 = Convert.ToString(sdr["clientALine1"]);

Client.Address.Line2 = Convert.ToString(sdr["clientALine2"]);

Client.Address.City = Convert.ToString(sdr["clientACity"]);

Client.Address.Province = Convert.ToString(sdr["clientAProvince"]);

Client.Address.PostalCode = Convert.ToString(sdr["clientPostalCode"]);

Client.Archived = (Convert.ToInt16(sdr["clientArchived"]) == 1) ? true : false;//First Converts Object to Int16 and then Int16 to Boolean Value

Clients.Add(Client);

bRead = sdr.Read(); // Priming read (must have 1st read before loop)

} // end while

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return Clients; // Single return

} // end method

public override int SelectClient(string ID, ref Client Client)

{

//

//Method Name : int SelectSalaryEmployee(string ID, ref Client Client)

//Purpose : Try to get a single Client object from the Client datastore

//Re-use :

//Input : string ID

// - The ID of the Client to load from the datastore

// ref Client Client

// - The Client object loaded from the datastore

//Output : - int

// 0 : Client loaded from datastore

// -1 : no Client was loaded from the datastore (not found)

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

Client = new Client();

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM Clients WHERE [clientID] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

Client.idNumber = Convert.ToString(sdr["clientID"]);

Client.FirstName = Convert.ToString(sdr["clientFirstNames"]);

Client.LastName = Convert.ToString(sdr["clientLastName"]);

Client.DateOfBirth = Convert.ToString(sdr["clientDateOfBirth"]);

Client.Cellphone = Convert.ToString(sdr["clientCellphone"]);

Client.EMailAddress = Convert.ToString(sdr["clientEmail"]);

Client.Telephone = Convert.ToString(sdr["clientTelephone"]);

Client.Address.Line1 = Convert.ToString(sdr["clientALine1"]);

Client.Address.Line2 = Convert.ToString(sdr["clientALine2"]);

Client.Address.City = Convert.ToString(sdr["clientACity"]);

Client.Address.Province = Convert.ToString(sdr["clientAProvince"]);

Client.Address.PostalCode = Convert.ToString(sdr["clientPostalCode"]);

Client.Archived = (Convert.ToInt16(sdr["clientArchived"]) == 1) ? true : false;//First Converts Object to Int16 and then Int16 to Boolean Value

rc = 0;

} // end if

else

{

rc = -1;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int Insert(Client Client)

{

//

//Method Name : int Insert(Client Client)

//Purpose : Try to insert a row in the Client datastore

//Re-use : DoesExist()

//Input : Client Client

// - The Client object to add to the Client datastore

//Output : - int

// 0 : Client inserted into datastore

// -1 : Client not inserted because a duplicate was found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

bool doesExist = false;

int rowsAffected = 0;

doesExist = DoesExist(Client.idNumber);

if (doesExist == false)

{

//TO:DO

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

\_sqlCon.Open(); // open connection

string insertQuery = "INSERT INTO Clients([clientID], [clientFirstNames], [clientLastName], " +

"[clientDateOfBirth],[clientCellphone],[clientEmail],[clientTelephone]," +

"[clientALine1],[clientALine2],[clientACity],[clientAProvince]," +

"[clientPostalCode],[clientArchived] ) VALUES(" +

"@clientID, @clientFirstNames, @clientLastName, @clientDateOfBirth,@clientCellphone,@clientEmail," +

"@clientTelephone,@clientALine1,@clientALine2,@clientACity,@clientAProvince," +

"@clientPostalCode,@clientArchived)";

SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@clientID",DbType.String),

new SQLiteParameter("@clientFirstNames",DbType.String),

new SQLiteParameter("@clientLastName", DbType.String),

new SQLiteParameter("@clientDateOfBirth", DbType.String),

new SQLiteParameter("@clientCellphone",DbType.String),

new SQLiteParameter("@clientEmail",DbType.String),

new SQLiteParameter("@clientTelephone",DbType.String),

new SQLiteParameter("@clientALine1",DbType.String),

new SQLiteParameter("@clientALine2",DbType.String),

new SQLiteParameter("@clientACity",DbType.String),

new SQLiteParameter("@clientAProvince",DbType.String),

new SQLiteParameter("@clientPostalCode",DbType.String),

new SQLiteParameter("@clientArchived",DbType.Int16)

};

sqlParams[0].Value = Client.idNumber; // Populate SQLiteParameters from Client

sqlParams[1].Value = Client.FirstName;

sqlParams[2].Value = Client.LastName;

sqlParams[3].Value = Client.DateOfBirth;

sqlParams[4].Value = Client.Cellphone;

sqlParams[5].Value = Client.EMailAddress;

sqlParams[6].Value = Client.Telephone;

sqlParams[7].Value = Client.Address.Line1;

sqlParams[8].Value = Client.Address.Line2;

sqlParams[9].Value = Client.Address.City;

sqlParams[10].Value = Client.Address.Province;

sqlParams[11].Value = Client.Address.PostalCode;

sqlParams[12].Value = Client.Archived ? 1 : 0;// ? 1 : 0 Converts Boolean to Int16 for Database Storage

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 1) // Test rowsAffected

{

// 1 row affected, thus 1 row added to datastore, thus success

rc = 0;

} // end if

} // end if

else

{

rc = -1;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override int Update(Client Client)

{

//

//Method Name : int Update(Client Client)

//Purpose : Try to update a row in the Client datastore

//Re-use : None

//Input : Client Client

// - The new Client data for the row in the Client datastore

//Output : - int

// 0 : Client found and updated successfully

// -1 : Client not updated because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // open connection

//

// REMEMBER: DO NOT update primary key (ID)!!!

//

//

// A better option would be to only update the fields that actually changed

//

string updateQuery = string.Format("UPDATE Clients SET [clientID]=@clientID, [clientFirstNames]=@clientFirstNames, " +

"[clientLastName]=@clientLastName,[clientDateOfBirth]=@clientDateOfBirth," +

"[clientCellphone]=@clientCellphone,[clientEmail]=@clientEmail," +

"[clientTelephone]=@clientTelephone,[clientALine1]=@clientALine1," +

"[clientALine2]=@clientALine2,[clientACity]=@clientACity," +

"[clientAProvince]=@clientAProvince,[clientPostalCode]=@clientPostalCode," +

"[clientArchived]=@clientArchived WHERE " +

"[clientID] = '{0}'", Client.idNumber);

SQLiteCommand sqlCommand = new SQLiteCommand(updateQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@clientID",DbType.String),

new SQLiteParameter("@clientFirstNames",DbType.String),

new SQLiteParameter("@clientLastName", DbType.String),

new SQLiteParameter("@clientDateOfBirth", DbType.String),

new SQLiteParameter("@clientCellphone",DbType.String),

new SQLiteParameter("@clientEmail",DbType.String),

new SQLiteParameter("@clientTelephone",DbType.String),

new SQLiteParameter("@clientALine1",DbType.String),

new SQLiteParameter("@clientALine2",DbType.String),

new SQLiteParameter("@clientACity",DbType.String),

new SQLiteParameter("@clientAProvince",DbType.String),

new SQLiteParameter("@clientPostalCode",DbType.String),

new SQLiteParameter("@clientArchived",DbType.Int16)

};

sqlParams[0].Value = Client.idNumber; // Populate SQLiteParameters from Client

sqlParams[1].Value = Client.FirstName;

sqlParams[2].Value = Client.LastName;

sqlParams[3].Value = Client.DateOfBirth;

sqlParams[4].Value = Client.Cellphone;

sqlParams[5].Value = Client.EMailAddress;

sqlParams[6].Value = Client.Telephone;

sqlParams[7].Value = Client.Address.Line1;

sqlParams[8].Value = Client.Address.Line2;

sqlParams[9].Value = Client.Address.City;

sqlParams[10].Value = Client.Address.Province;

sqlParams[11].Value = Client.Address.PostalCode;

sqlParams[12].Value = Client.Archived ? 1 : 0;// ? 1 : 0 Converts Boolean to Int16 for Database Storage

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int Delete(string ID)

{

//

//Method Name : int Delete(string ID)

//Purpose : Try to delete a row from the Client datastore

//Re-use : None

//Input : string ID

// - the ID of the Client to delete in the Client datastore

//Output : - int

// 0 : Client found and deleted successfully

// -1 : Client not deleted because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // Open connection

string deleteQuery = string.Format("DELETE FROM Clients WHERE [clientID] = '{0}'", ID);

SQLiteCommand sqlCommand = new SQLiteCommand(deleteQuery, \_sqlCon); // setup command

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override bool DoesExist(string ID)

{

//

//Method Name : bool DoesExist(string ID)

//Purpose : Determines if a given Client exists in the Client datastore.

//Re-use : None

//Input : string ID

// - the ID of the Client to search in the Client datastore

//Output : - bool

// true : Client found

// false : Client not found

//

bool rc = false; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM Clients WHERE [clientID] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

rc = true;

} // end if

else

{

rc = false;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

}//end class

}//end namespace

leaseUnitProviderBase

namespace StorageUnitManagementSystem.DAL

{

public abstract class LeaseUnitsProviderBase

{

/// <summary>

/// This method gets the list of all the business objects from the Client datastore.

/// It returns the list of business objects

/// </summary>

public abstract List<LeaseUnits> SelectAll();

/// <summary>

/// This method gets a single Client object from the Client datastore.

/// It returns 0 to indicate the Client was loaded from datastore, or

/// -1 to indicate that no Client was loaded from the datastore (not found).

/// </summary>

/// <param name="ID">The ID of the Client to load from the datastore.</param>

/// <param name="Client">The Client object loaded from the datastore.</param>

public abstract int SelectLeaseUnit(string ID, ref LeaseUnits Client);

/// <summary>

/// This method inserts a row in the Client datastore.

/// It returns 0 to indicate the Client was inserted into datastore, or

/// -1 to indicate the Client was not inserted because a duplicate was found

/// </summary>

/// <param name="Client">The Client object to add to the Client datastore.</param>

public abstract int Insert(LeaseUnits LeaseUnits);

/// <summary>

/// This method updates a row in the Client datastore.

/// It returns 0 to indicate the Client was found and updated successfully, or

/// -1 to indicate the Client was not updated because the record was not found

/// </summary>

/// <param name="Client">The new Client data for the row in the Client datastore.</param>

public abstract int Update(LeaseUnits LeaseUnits);

public abstract int UpdatePopUp(LeaseUnits LeaseUnits);

/// <summary>

/// This method deletes a row in the Client datastore.

/// It returns 0 to indicate the Client was found and deleted successfully, or

/// -1 to indicate the Client was not deleted because the record was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to delete in the Client datastore.</param>

public abstract int Delete(string ID);

/// <summary>

/// This method determines if a given Client exists in the Client datastore.

/// It returns true to indicate the Client was foundy, or

/// false to indicate the Client was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to search in the Client datastore.</param>

public abstract bool DoesExist(string ID);

}//end class

}//end namespace

LeaseUnitSQLiteProvider

namespace StorageUnitManagementSystem.DAL

{

public class LeaseUnitsSQLiteProvider : LeaseUnitsProviderBase

{

private SQLiteConnection \_sqlCon;

private string \_conStr = CreateDatabase.ConStr;

public override List<LeaseUnits> SelectAll()

{

//

//Method Name : List<Client> SelectAll()

//Purpose : Try to get all the Client objects from the datastore

//Re-use : None

//Input : None

//Output : - ref List<Client>

// - the list that will contain the Client objects loaded from datastore

//

List<LeaseUnits> LeaseUnits; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

LeaseUnits = new List<LeaseUnits>(); // this ensures that if there are no records,

// the returned list will not be null, but

// it will be empty (Count = 0)

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM LeaseUnits";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read(); // Priming read (must have 2nd read in loop)

while (bRead == true) // false indicates no more rows/records4\4

{

LeaseUnits LeaseUnit = new LeaseUnits();

//unit.Address = new Address();

//unit.Phone = new Phone();

LeaseUnit.Client = new Client();

LeaseUnit.StorageUnit = new StorageUnit();

LeaseUnit.LeaseID = Convert.ToString(sdr["LeaseID"]);

LeaseUnit.Client.idNumber = Convert.ToString(sdr["ClientID"]);

LeaseUnit.Client.FirstName = Convert.ToString(sdr["ClientName"]);

LeaseUnit.Client.LastName = Convert.ToString(sdr["ClientSurName"]);

LeaseUnit.StorageUnit.UnitId = Convert.ToString(sdr["UnitID"]);

LeaseUnit.StorageUnit.UnitClassification = Convert.ToString(sdr["UnitClass"]);

LeaseUnit.StorageUnit.UnitPrice = Convert.ToDouble(sdr["UnitPrice"]);

LeaseUnit.NoOfUnits = Convert.ToInt32(sdr["NoOfUnits"]);

LeaseUnit.ClientWaitingList = (Convert.ToInt16(sdr["ClientWaitingList"]) == 1) ? true : false;

LeaseUnit.AvailableUnits = Convert.ToString(sdr["AvailableUnits"]);

LeaseUnit.TypeOfPayment = Convert.ToString(sdr["TypeOfPayment"]);

LeaseUnit.DateOfPayment = Convert.ToString(sdr["DatePaid"]);

LeaseUnit.DateOfContractStart = Convert.ToString(sdr["DateOfContractStart"]);

LeaseUnit.DateOfContractEnd = Convert.ToString(sdr["DateOfContractEnd"]);

LeaseUnit.AmountDeposited = Convert.ToString(sdr["AmountDeposited"]);

LeaseUnit.AmountOwed = Convert.ToString(sdr["AmountOwed"]);

LeaseUnit.AmountPaid = Convert.ToString(sdr["AmountPaid"]);

LeaseUnit.ClientCurrentTotal = Convert.ToString(sdr["ClientCurrentTotal"]);

LeaseUnit.UnitLeased = (Convert.ToInt16(sdr["UnitLeased"]) == 1) ? true : false;

LeaseUnit.ClientAdded = (Convert.ToInt16(sdr["ClientAdded"]) == 1) ? true : false;

LeaseUnit.TotalUnitPrice = Convert.ToString(sdr["TotalUnitPrice"]);

LeaseUnit.StorageUnit.UnitSize = Convert.ToString(sdr["UnitSize"]);

LeaseUnits.Add(LeaseUnit);

bRead = sdr.Read(); // Priming read (must have 1st read before loop)

} // end while

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return LeaseUnits; // Single return

} // end method

public override int SelectLeaseUnit(string ID, ref LeaseUnits LeaseUnit)

{

//

//Method Name : int SelectSalaryEmployee(string ID, ref Client Client)

//Purpose : Try to get a single Client object from the Client datastore

//Re-use :

//Input : string ID

// - The ID of the Client to load from the datastore

// ref Client Client

// - The Client object loaded from the datastore

//Output : - int

// 0 : Client loaded from datastore

// -1 : no Client was loaded from the datastore (not found)

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

LeaseUnit = new LeaseUnits();

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM LeaseUnits WHERE [LeaseID] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

LeaseUnit.LeaseID = Convert.ToString(sdr["LeaseID"]);

LeaseUnit.Client.idNumber = Convert.ToString(sdr["ClientID"]);

LeaseUnit.Client.FirstName = Convert.ToString(sdr["ClientName"]);

LeaseUnit.Client.LastName = Convert.ToString(sdr["ClientSurName"]);

LeaseUnit.StorageUnit.UnitId = Convert.ToString(sdr["UnitID"]);

LeaseUnit.StorageUnit.UnitClassification = Convert.ToString(sdr["UnitClass"]);

LeaseUnit.StorageUnit.UnitPrice = Convert.ToDouble(sdr["UnitPrice"]);

LeaseUnit.NoOfUnits = Convert.ToInt32(sdr["NoOfUnits"]);

LeaseUnit.ClientWaitingList = (Convert.ToInt16(sdr["ClientWaitingList"]) == 1) ? true : false;

LeaseUnit.AvailableUnits = Convert.ToString(sdr["AvailableUnits"]);

LeaseUnit.TypeOfPayment = Convert.ToString(sdr["TypeOfPayment"]);

LeaseUnit.DateOfPayment = Convert.ToString(sdr["DatePaid"]);

LeaseUnit.DateOfContractStart = Convert.ToString(sdr["DateOfContractStart"]);

LeaseUnit.DateOfContractEnd = Convert.ToString(sdr["DateOfContractEnd"]);

LeaseUnit.AmountDeposited = Convert.ToString(sdr["AmountDeposited"]);

LeaseUnit.AmountOwed = Convert.ToString(sdr["AmountOwed"]);

LeaseUnit.AmountPaid = Convert.ToString(sdr["AmountPaid"]);

LeaseUnit.ClientCurrentTotal = Convert.ToString(sdr["ClientCurrentTotal"]);

LeaseUnit.UnitLeased = (Convert.ToInt16(sdr["UnitLeased"]) == 1) ? true : false;

LeaseUnit.ClientAdded = (Convert.ToInt16(sdr["ClientAdded"]) == 1) ? true : false;

LeaseUnit.TotalUnitPrice = Convert.ToString(sdr["TotalUnitPrice"]);

LeaseUnit.StorageUnit.UnitSize = Convert.ToString(sdr["UnitSize"]);

rc = 0;

} // end if

else

{

rc = -1;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int Insert(LeaseUnits LeaseUnit)

{

//

//Method Name : int Insert(Client Client)

//Purpose : Try to insert a row in the Client datastore

//Re-use : DoesExist()

//Input : Client Client

// - The Client object to add to the Client datastore

//Output : - int

// 0 : Client inserted into datastore

// -1 : Client not inserted because a duplicate was found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

bool doesExist = false;

int rowsAffected = 0;

{

//TO:DO

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

\_sqlCon.Open(); // open connection

string insertQuery = "INSERT INTO LeaseUnits([LeaseID],[ClientID], [ClientName], [ClientSurname], " +

"[UnitID],[UnitClass],[UnitPrice],[NoOfUnits]," +

"[ClientWaitingList],[AvailableUnits],[TypeOfPayment],[DatePaid]," +

"[DateOfContractStart],[DateOfContractEnd],[AmountDeposited],[AmountOwed]," +

"[AmountPaid],[ClientCurrentTotal],[UnitLeased],[ClientAdded],[TotalUnitPrice],[UnitSize] )" +

" VALUES(" +

"@LeaseID,@ClientID, @ClientName, @ClientSurname, @UnitID,@UnitClass,@UnitPrice," +

"@NoOfUnits,@ClientWaitingList,@AvailableUnits,@TypeOfPayment,@DatePaid," +

"@DateOfContractStart,@DateOfContractEnd,@AmountDeposited,@AmountOwed,@AmountPaid," +

"@ClientCurrentTotal,@UnitLeased,@ClientAdded,@TotalUnitPrice,@UnitSize)";

SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@LeaseID",DbType.String),

new SQLiteParameter("@ClientID",DbType.String),

new SQLiteParameter("@ClientName",DbType.String),

new SQLiteParameter("@ClientSurname", DbType.String),

new SQLiteParameter("@UnitID", DbType.String),

new SQLiteParameter("@UnitClass",DbType.String),

new SQLiteParameter("@UnitPrice",DbType.Double),

new SQLiteParameter("@NoOfUnits",DbType.String),

new SQLiteParameter("@ClientWaitingList",DbType.String),

new SQLiteParameter("@AvailableUnits",DbType.String),

new SQLiteParameter("@TypeOfPayment",DbType.String),

new SQLiteParameter("@DatePaid",DbType.String),

new SQLiteParameter("@DateOfContractStart",DbType.String),

new SQLiteParameter("@DateOfContractEnd",DbType.String),

new SQLiteParameter("@AmountDeposited",DbType.String),

new SQLiteParameter("@AmountOwed", DbType.String),

new SQLiteParameter("@AmountPaid", DbType.String),

new SQLiteParameter("@ClientCurrentTotal", DbType.String),

new SQLiteParameter("@UnitLeased", DbType.Int16),

new SQLiteParameter("@ClientAdded", DbType.Int16),

new SQLiteParameter("@TotalUnitPrice", DbType.String),

new SQLiteParameter("@UnitSize", DbType.String)

};

sqlParams[0].Value = LeaseUnit.LeaseID;

sqlParams[1].Value = LeaseUnit.Client.idNumber; // Populate SQLiteParameters from Client

sqlParams[2].Value = LeaseUnit.Client.FirstName;

sqlParams[3].Value = LeaseUnit.Client.LastName;

sqlParams[4].Value = LeaseUnit.StorageUnit.UnitId;

sqlParams[5].Value = LeaseUnit.StorageUnit.UnitClassification;

sqlParams[6].Value = LeaseUnit.StorageUnit.UnitPrice;

sqlParams[7].Value = LeaseUnit.NoOfUnits;

sqlParams[8].Value = LeaseUnit.ClientWaitingList;

sqlParams[9].Value = LeaseUnit.AvailableUnits;

sqlParams[10].Value = LeaseUnit.TypeOfPayment;

sqlParams[11].Value = LeaseUnit.DateOfPayment;

sqlParams[12].Value = LeaseUnit.DateOfContractStart;

sqlParams[13].Value = LeaseUnit.DateOfContractEnd;

sqlParams[14].Value = LeaseUnit.AmountDeposited;

sqlParams[15].Value = LeaseUnit.AmountOwed;

sqlParams[16].Value = LeaseUnit.AmountPaid;

sqlParams[17].Value = LeaseUnit.ClientCurrentTotal;

sqlParams[18].Value = LeaseUnit.UnitLeased ? 1 : 0;// ? 1 : 0 Converts Boolean to Int16 for Database Storage

sqlParams[19].Value = LeaseUnit.ClientAdded ? 1 : 0;

sqlParams[20].Value = LeaseUnit.TotalUnitPrice;

sqlParams[21].Value = LeaseUnit.StorageUnit.UnitSize;

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 1) // Test rowsAffected

{

// 1 row affected, thus 1 row added to datastore, thus success

rc = 0;

} // end if

} // end if

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override int Update(LeaseUnits LeaseUnit)

{

//

//Method Name : int Update(Client Client)

//Purpose : Try to update a row in the Client datastore

//Re-use : None

//Input : Client Client

// - The new Client data for the row in the Client datastore

//Output : - int

// 0 : Client found and updated successfully

// -1 : Client not updated because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // open connection

//

// REMEMBER: DO NOT update primary key (ID)!!!

//

//

// A better option would be to only update the fields that actually changed

//

string updateQuery = "UPDATE LeaseUnits SET [ClientID]=@ClientID,[ClientName]=@ClientName," +

"[ClientSurname]=@ClientSurname,[UnitID]=@UnitID," +

"[UnitClass]=@UnitClass,[UnitPrice]=@UnitPrice," +

"[NoOfUnits]=@NoOfUnits,[ClientWaitingList]=@ClientWaitingList," +

"[AvailableUnits]=@AvailableUnits,[TypeOfPayment]=@TypeOfPayment," +

"[DatePaid]=@DatePaid,[DateOfContractStart]=@DateOfContractStart," +

"[DateOfContractEnd]=@DateOfContractEnd,[AmountDeposited]=@AmountDeposited," +

"[AmountOwed]=@AmountOwed,[AmountPaid]=@AmountPaid," +

"[ClientCurrentTotal]=@ClientCurrentTotal," +

"[UnitLeased]=@UnitLeased,[ClientAdded]=@ClientAdded," +

"[TotalUnitPrice]=@TotalUnitPrice,[UnitSize]=@UnitSize WHERE " +

$"[LeaseID] = '{LeaseUnit.LeaseID}'";

SQLiteCommand sqlCommand = new SQLiteCommand(updateQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = {

new SQLiteParameter("@LeaseID",DbType.String),

new SQLiteParameter("@ClientID",DbType.String),

new SQLiteParameter("@ClientName",DbType.String),

new SQLiteParameter("@ClientSurname", DbType.String),

new SQLiteParameter("@UnitID", DbType.String),

new SQLiteParameter("@UnitClass",DbType.String),

new SQLiteParameter("@UnitPrice",DbType.Double),

new SQLiteParameter("@NoOfUnits",DbType.Int16),

new SQLiteParameter("@ClientWaitingList",DbType.Int16),

new SQLiteParameter("@AvailableUnits",DbType.String),

new SQLiteParameter("@TypeOfPayment",DbType.String),

new SQLiteParameter("@DatePaid",DbType.String),

new SQLiteParameter("@DateOfContractStart",DbType.String),

new SQLiteParameter("@DateOfContractEnd",DbType.String),

new SQLiteParameter("@AmountDeposited",DbType.String),

new SQLiteParameter("@AmountOwed", DbType.String),

new SQLiteParameter("@AmountPaid", DbType.String),

new SQLiteParameter("@ClientCurrentTotal", DbType.String),

new SQLiteParameter("@UnitLeased", DbType.Int16),

new SQLiteParameter("@ClientAdded", DbType.Int16),

new SQLiteParameter("@TotalUnitPrice", DbType.String),

new SQLiteParameter("@UnitSize", DbType.String)

};

// Populate SQLiteParameters from Client

sqlParams[0].Value = LeaseUnit.LeaseID;

sqlParams[1].Value = LeaseUnit.Client.idNumber;

sqlParams[2].Value = LeaseUnit.Client.FirstName;

sqlParams[3].Value = LeaseUnit.Client.LastName;

sqlParams[4].Value = LeaseUnit.StorageUnit.UnitId;

sqlParams[5].Value = LeaseUnit.StorageUnit.UnitClassification;

sqlParams[6].Value = LeaseUnit.StorageUnit.UnitPrice;

sqlParams[7].Value = LeaseUnit.NoOfUnits;

sqlParams[8].Value = LeaseUnit.ClientWaitingList;

sqlParams[9].Value = LeaseUnit.AvailableUnits;

sqlParams[10].Value = LeaseUnit.TypeOfPayment;

sqlParams[11].Value = LeaseUnit.DateOfPayment;

sqlParams[12].Value = LeaseUnit.DateOfContractStart;

sqlParams[13].Value = LeaseUnit.DateOfContractEnd;

sqlParams[14].Value = LeaseUnit.AmountDeposited;

sqlParams[15].Value = LeaseUnit.AmountOwed;

sqlParams[16].Value = LeaseUnit.AmountPaid;

sqlParams[17].Value = LeaseUnit.ClientCurrentTotal;

sqlParams[18].Value = LeaseUnit.UnitLeased ? 1 : 0;// ? 1 : 0 Converts Boolean to Int16 for Database Storage

sqlParams[19].Value = LeaseUnit.ClientAdded ? 1 : 0;

sqlParams[20].Value = LeaseUnit.TotalUnitPrice;

sqlParams[21].Value = LeaseUnit.StorageUnit.UnitSize;

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int UpdatePopUp(LeaseUnits LeaseUnit)

{

//

//Method Name : int Update(Client Client)

//Purpose : Try to update a row in the Client datastore

//Re-use : None

//Input : Client Client

// - The new Client data for the row in the Client datastore

//Output : - int

// 0 : Client found and updated successfully

// -1 : Client not updated because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // open connection

//

// REMEMBER: DO NOT update primary key (ID)!!!

//

//

// A better option would be to only update the fields that actually changed

//

string updateQuery = "UPDATE LeaseUnits SET [ClientName]=@ClientName," +

"[ClientSurname]=@ClientSurname,[DatePaid]=@DatePaid," +

"[AmountOwed]=@AmountOwed,[AmountPaid]=@AmountPaid," +

$"[UnitLeased]=@UnitLeased WHERE [LeaseID] = '{LeaseUnit.LeaseID}'";

SQLiteCommand sqlCommand = new SQLiteCommand(updateQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{ new SQLiteParameter("@LeaseID",DbType.String),

new SQLiteParameter("@ClientName",DbType.String),

new SQLiteParameter("@ClientSurname", DbType.String),

new SQLiteParameter("@DatePaid",DbType.String),

new SQLiteParameter("@AmountOwed", DbType.String),

new SQLiteParameter("@AmountPaid", DbType.String),

new SQLiteParameter("@UnitLeased", DbType.Int16),

};

// Populate SQLiteParameters from Client

sqlParams[0].Value = LeaseUnit.LeaseID;

sqlParams[1].Value = LeaseUnit.Client.FirstName;

sqlParams[2].Value = LeaseUnit.Client.LastName;

sqlParams[3].Value = LeaseUnit.DateOfPayment;

sqlParams[4].Value = LeaseUnit.AmountOwed;

sqlParams[5].Value = LeaseUnit.AmountPaid;

sqlParams[6].Value = LeaseUnit.UnitLeased ? 1 : 0;// ? 1 : 0 Converts Boolean to Int16 for Database Storage

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int Delete(string ID)

{

//

//Method Name : int Delete(string ID)

//Purpose : Try to delete a row from the Client datastore

//Re-use : None

//Input : string ID

// - the ID of the Client to delete in the Client datastore

//Output : - int

// 0 : Client found and deleted successfully

// -1 : Client not deleted because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // Open connection

string deleteQuery = $"DELETE FROM LeaseUnits WHERE [LeaseID] = '{ID}'";

SQLiteCommand sqlCommand = new SQLiteCommand(deleteQuery, \_sqlCon); // setup command

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override bool DoesExist(string ID)

{

//

//Method Name : bool DoesExist(string ID)

//Purpose : Determines if a given Client exists in the Client datastore.

//Re-use : None

//Input : string ID

// - the ID of the Client to search in the Client datastore

//Output : - bool

// true : Client found

// false : Client not found

//

bool rc = false; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM LeaseUnits WHERE [LeaseID] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

rc = true;

} // end if

else

{

rc = false;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

}//end class

}//end namespace

StorageUnitManagementProviderBase

using System.Collections.Generic;

using StorageUnitManagementSystem.BL.Classes;

//

namespace StorageUnitManagementSystem.DAL

{

public abstract class StorageUnitManagementProviderBase

{

/// <summary>

/// This method gets the list of all the business objects from the StorageUnit datastore.

/// It returns the list of business objects

/// </summary>

public abstract List<StorageUnit> SelectAll();

/// <summary>

/// This method gets a single StorageUnit object from the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was loaded from datastore, or

/// -1 to indicate that no StorageUnit was loaded from the datastore (not found).

/// </summary>

/// <param name="ID">The ID of the StorageUnit to load from the datastore.</param>

/// <param name="StorageUnit">The StorageUnit object loaded from the datastore.</param>

public abstract int SelectStorageUnit(string ID, ref StorageUnit StorageUnit);

/// <summary>

/// This method inserts a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was inserted into datastore, or

/// -1 to indicate the StorageUnit was not inserted because a duplicate was found

/// </summary>

/// <param name="StorageUnit">The StorageUnit object to add to the StorageUnit datastore.</param>

public abstract int Insert(StorageUnit StorageUnit);

/// <summary>

/// This method updates a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was found and updated successfully, or

/// -1 to indicate the StorageUnit was not updated because the record was not found

/// </summary>

/// <param name="StorageUnit">The new StorageUnit data for the row in the StorageUnit datastore.</param>

public abstract int Update(StorageUnit StorageUnit);

/// <summary>

/// This method deletes a row in the StorageUnit datastore.

/// It returns 0 to indicate the StorageUnit was found and deleted successfully, or

/// -1 to indicate the StorageUnit was not deleted because the record was not found

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to delete in the StorageUnit datastore.</param>

public abstract int Delete(string ID);

/// <summary>

/// This method determines if a given StorageUnit exists in the StorageUnit datastore.

/// It returns true to indicate the StorageUnit was foundy, or

/// false to indicate the StorageUnit was not found

/// </summary>

/// <param name="ID">The StorageUnit ID of the StorageUnit to search in the StorageUnit datastore.</param>

public abstract bool DoesExist(string ID);

}//end class

}//end namespace

StorageUnitManagementSQLProvider

namespace StorageUnitManagementSystem.DAL

{

class StroageUnitManagementSQLiteProvider : StorageUnitManagementProviderBase

{

private SQLiteConnection \_sqlCon;

private string \_conStr = CreateDatabase.ConStr;

public override List<StorageUnit> SelectAll()

{

//

//Method Name : List<StorageUnit> SelectAll()

//Purpose : Try to get all the StorageUnit objects from the datastore

//Re-use : None

//Input : None

//Output : - ref List<StorageUnit>

// - the list that will contain the StorageUnit objects loaded from datastore

//

List<StorageUnit> StorageUnits; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

StorageUnits = new List<StorageUnit>(); // this ensures that if there are no records,

// the returned list will not be null, but

// it will be empty (Count = 0)

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM StorageUnits";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read(); // Priming read (must have 2nd read in loop)

while (bRead == true) // false indicates no more rows/records4\4

{

StorageUnit StorageUnit = new StorageUnit();

//unit.Address = new Address();

//unit.Phone = new Phone();

StorageUnit.UnitId = Convert.ToString(sdr["suID"]);

StorageUnit.UnitClassification = Convert.ToString(sdr["suClassification"]);

StorageUnit.UnitPrice = Convert.ToDouble(sdr["suPrice"]);

StorageUnit.UnitSize = Convert.ToString(sdr["suSize"]);

StorageUnit.UnitArrears = (Convert.ToInt16(sdr["suArrears"]) == 1) ? true : false;//First Converts Object to Int16 and then Int16 to Boolean Value

StorageUnit.UnitOccupied = (Convert.ToInt16(sdr["suOccupied"]) == 1) ? true : false;

StorageUnit.UnitInAdvance = (Convert.ToInt16(sdr["suAdvance"]) == 1) ? true : false;

StorageUnit.UnitUpToDate = (Convert.ToInt16(sdr["suUpToDate"]) == 1) ? true : false;

StorageUnit.UnitOwnerId = Convert.ToString(sdr["suOwnerID"]);

StorageUnits.Add(StorageUnit);

bRead = sdr.Read(); // Priming read (must have 1st read before loop)

} // end while

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return StorageUnits; // Single return

} // end method

public override int SelectStorageUnit(string ID, ref StorageUnit StorageUnit)

{

//

//Method Name : int SelectStorageUnit(string ID, ref StorageUnit StorageUnit)

//Purpose : Try to get a single StorageUnit object from the StorageUnit datastore

//Re-use :

//Input : string ID

// - The ID of the StorageUnit to load from the datastore

// ref StorageUnit StorageUnit

// - The StorageUnit object loaded from the datastore

//Output : - int

// 0 : StorageUnit loaded from datastore

// -1 : no StorageUnit was loaded from the datastore (not found)

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

StorageUnit = new StorageUnit();

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM StorageUnits WHERE [suID] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

StorageUnit.UnitId = Convert.ToString(sdr["suID"]);

StorageUnit.UnitClassification = Convert.ToString(sdr["suClassification"]);

StorageUnit.UnitPrice = Convert.ToDouble(sdr["suPrice"]);

StorageUnit.UnitSize = Convert.ToString(sdr["suSize"]);

StorageUnit.UnitOwnerId = Convert.ToString(sdr["suOwner"]);

StorageUnit.UnitArrears = (Convert.ToInt16(sdr["suArrears"]) == 1) ? true : false; //First Converts Object to Int16 and then Int16 to Boolean Value

StorageUnit.UnitOccupied = (Convert.ToInt16(sdr["suOccupied"]) == 1) ? true : false;

StorageUnit.UnitInAdvance = (Convert.ToInt16(sdr["suAdvance"]) == 1) ? true : false;

StorageUnit.UnitUpToDate = (Convert.ToInt16(sdr["suUpToDate"]) == 1) ? true : false;

rc = 0;

} // end if

else

{

rc = -1;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

//public override int Insert(StorageUnit StorageUnit)

//{

// //

// //Method Name : int Insert(StorageUnit StorageUnit)

// //Purpose : Try to insert a row in the StorageUnit datastore

// //Re-use : DoesExist()

// //Input : StorageUnit StorageUnit

// // - The StorageUnit object to add to the StorageUnit datastore

// //Output : - int

// // 0 : StorageUnit inserted into datastore

// // -1 : StorageUnit not inserted because a duplicate was found

// //

// int rc = 0; // will be returned, thus can not be declared in try block

// try

// {

// bool doesExist = false;

// int rowsAffected = 0;

// doesExist = DoesExist(StorageUnit.UnitId);

// if (doesExist == false)

// {

// //TO:DO

// \_sqlCon = new SQLiteConnection(\_conStr); // new connection

// \_sqlCon.Open(); // open connection

// string insertQuery = "INSERT INTO StorageUnits([suID], [suClassification], [suPrice], " +

// "[suSize],[suArrears],[suOccupied],[suAdvance],[suUpToDate] ) VALUES(" +

// "@seID, @suClassification, @suPrice, @suSize,@suArrears,@suOccupied," +

// "@suAdvance,@suUpToDate)";

// SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, \_sqlCon); // setup command

// SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

// {

// new SQLiteParameter("@suID",DbType.String),

// new SQLiteParameter("@suClassification",DbType.String),

// new SQLiteParameter("@suPrice", DbType.VarNumeric),

// new SQLiteParameter("@suSize", DbType.String),

// new SQLiteParameter("@suArrears",DbType.Int16),

// new SQLiteParameter("@suOccupied",DbType.Int16),

// new SQLiteParameter("@suAdvance",DbType.Int16),

// new SQLiteParameter("@suUpToDate",DbType.Int16)

// };

// sqlParams[0].Value = StorageUnit.UnitId; // Populate SQLiteParameters from StorageUnit

// sqlParams[1].Value = StorageUnit.UnitClassification;

// sqlParams[2].Value = StorageUnit.UnitPrice;

// sqlParams[3].Value = StorageUnit.UnitSize;

// sqlParams[4].Value = StorageUnit.UnitArrears ? 1 : 0; // ? 1 : 0 Converts Boolean to Int16 for Database Storage

// sqlParams[5].Value = StorageUnit.UnitOccupied ? 1 : 0;

// sqlParams[6].Value = StorageUnit.UnitInAdvance ? 1 : 0;

// sqlParams[7].Value = StorageUnit.UnitUpToDate ? 1 : 0;

// sqlCommand.Parameters.AddRange(sqlParams);

// rowsAffected = sqlCommand.ExecuteNonQuery();

// if (rowsAffected == 1) // Test rowsAffected

// {

// // 1 row affected, thus 1 row added to datastore, thus success

// rc = 0;

// } // end if

// } // end if

// else

// {

// rc = -1;

// } // end else

// } // end try

// catch (Exception ex)

// {

// throw ex;

// } // end catch

// finally

// {

// \_sqlCon.Close(); // Close connection

// } // end finally

// return rc; // Single return

//} // end method

//public override int Insert(StorageUnit StorageUnit)

//{

// //

// //Method Name : int Insert(StorageUnit StorageUnit)

// //Purpose : Try to insert a row in the StorageUnit datastore

// //Re-use : DoesExist()

// //Input : StorageUnit StorageUnit

// // - The StorageUnit object to add to the StorageUnit datastore

// //Output : - int

// // 0 : StorageUnit inserted into datastore

// // -1 : StorageUnit not inserted because a duplicate was found

// //

// int rc = 0; // will be returned, thus can not be declared in try block

// try

// {

// bool doesExist = false;

// int rowsAffected = 0;

// doesExist = DoesExist(StorageUnit.UnitId);

// if (doesExist == false)

// {

// //TO:DO

// \_sqlCon = new SQLiteConnection(\_conStr); // new connection

// \_sqlCon.Open(); // open connection

// string insertQuery = "INSERT INTO StorageUnits([suID], [suClassification], [suPrice], " +

// "[suSize],[suArrears],[suOccupied],[suAdvance],[suUpToDate] ) VALUES(" +

// "@seID, @suClassification, @suPrice, @suSize,@suArrears,@suOccupied," +

// "@suAdvance,@suUpToDate)";

// SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, \_sqlCon); // setup command

// SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

// {

// new SQLiteParameter("@suID",DbType.String),

// new SQLiteParameter("@suClassification",DbType.String),

// new SQLiteParameter("@suPrice", DbType.VarNumeric),

// new SQLiteParameter("@suSize", DbType.String),

// new SQLiteParameter("@suArrears",DbType.Int16),

// new SQLiteParameter("@suOccupied",DbType.Int16),

// new SQLiteParameter("@suAdvance",DbType.Int16),

// new SQLiteParameter("@suUpToDate",DbType.Int16)

// };

// sqlParams[0].Value = StorageUnit.UnitId; // Populate SQLiteParameters from StorageUnit

// sqlParams[1].Value = StorageUnit.UnitClassification;

// sqlParams[2].Value = StorageUnit.UnitPrice;

// sqlParams[3].Value = StorageUnit.UnitSize;

// sqlParams[4].Value = StorageUnit.UnitArrears ? 1 : 0; // ? 1 : 0 Converts Boolean to Int16 for Database Storage

// sqlParams[5].Value = StorageUnit.UnitOccupied ? 1 : 0;

// sqlParams[6].Value = StorageUnit.UnitInAdvance ? 1 : 0;

// sqlParams[7].Value = StorageUnit.UnitUpToDate ? 1 : 0;

// sqlCommand.Parameters.AddRange(sqlParams);

// rowsAffected = sqlCommand.ExecuteNonQuery();

// if (rowsAffected == 1) // Test rowsAffected

// {

// // 1 row affected, thus 1 row added to datastore, thus success

// rc = 0;

// } // end if

// } // end if

// else

// {

// rc = -1;

// } // end else

// } // end try

// catch (Exception ex)

// {

// throw ex;

// } // end catch

// finally

// {

// \_sqlCon.Close(); // Close connection

// } // end finally

// return rc; // Single return

//} // end method

public override int Insert(StorageUnit StorageUnit)

{

//

//Method Name : int Insert(StorageUnit StorageUnit)

//Purpose : Try to insert a row in the StorageUnit datastore

//Re-use : DoesExist()

//Input : StorageUnit StorageUnit

// - The StorageUnit object to add to the StorageUnit datastore

//Output : - int

// 0 : StorageUnit inserted into datastore

// -1 : StorageUnit not inserted because a duplicate was found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

bool doesExist = false;

int rowsAffected = 0;

doesExist = DoesExist(StorageUnit.UnitId);

if (doesExist == false)

{

//TO:DO

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

\_sqlCon.Open(); // open connection

string insertQuery = "INSERT INTO StorageUnits([suID], [suClassification], [suPrice], " +

"[suSize],[suArrears],[suOccupied],[suAdvance],[suUpToDate],[suOwnerID] ) VALUES(" +

"@suID, @suClassification, @suPrice, @suSize,@suArrears,@suOccupied," +

"@suAdvance,@suUpToDate,@suOwnerID)";

SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@suID",DbType.String),

new SQLiteParameter("@suClassification",DbType.String),

new SQLiteParameter("@suPrice", DbType.VarNumeric),

new SQLiteParameter("@suSize", DbType.String),

new SQLiteParameter("@suArrears",DbType.Int16),

new SQLiteParameter("@suOccupied",DbType.Int16),

new SQLiteParameter("@suAdvance",DbType.Int16),

new SQLiteParameter("@suUpToDate",DbType.Int16),

new SQLiteParameter("@suOwnerID",DbType.String),

};

sqlParams[0].Value = StorageUnit.UnitId;

sqlParams[1].Value = StorageUnit.UnitClassification;

sqlParams[2].Value = StorageUnit.UnitPrice;

sqlParams[3].Value = StorageUnit.UnitSize;

sqlParams[4].Value = StorageUnit.UnitArrears ? 1 : 0; // ? 1 : 0 Converts Boolean to Int16 for Database Storage

sqlParams[5].Value = StorageUnit.UnitOccupied ? 1 : 0;

sqlParams[6].Value = StorageUnit.UnitInAdvance ? 1 : 0;

sqlParams[7].Value = StorageUnit.UnitUpToDate ? 1 : 0;

sqlParams[8].Value = StorageUnit.UnitOwnerId;

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 1) // Test rowsAffected

{

// 1 row affected, thus 1 row added to datastore, thus success

rc = 0;

} // end if

} // end if

else

{

rc = -1;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override int Update(StorageUnit StorageUnit)

{

//

//Method Name : int Update(StorageUnit StorageUnit)

//Purpose : Try to update a row in the StorageUnit datastore

//Re-use : None

//Input : StorageUnit StorageUnit

// - The new StorageUnit data for the row in the StorageUnit datastore

//Output : - int

// 0 : StorageUnit found and updated successfully

// -1 : StorageUnit not updated because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // open connection

//

// REMEMBER: DO NOT update primary key (ID)!!!

//

//

// A better option would be to only update the fields that actually changed

//

string updateQuery = string.Format("UPDATE StorageUnits SET [suClassification]=@suClassification, [suPrice]=@suPrice, " +

"[suSize]=@suSize,[suArrears]=@suArrears," +

"[suOccupied]=@suOccupied,[suAdvance]=@suAdvance," +

"[suUpToDate]=@suUpToDate WHERE " +

"[suID] = '{0}'", StorageUnit.UnitId);

SQLiteCommand sqlCommand = new SQLiteCommand(updateQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@suID",DbType.String),

new SQLiteParameter("@suClassification",DbType.String),

new SQLiteParameter("@suPrice", DbType.VarNumeric),

new SQLiteParameter("@suSize", DbType.String),

new SQLiteParameter("@suArrears",DbType.Int16),

new SQLiteParameter("@suOccupied",DbType.Int16),

new SQLiteParameter("@suAdvance",DbType.Int16),

new SQLiteParameter("@suUpToDate",DbType.Int16)

};

sqlParams[0].Value = StorageUnit.UnitId; // Populate SQLiteParameters from StorageUnit

sqlParams[1].Value = StorageUnit.UnitClassification;

sqlParams[2].Value = StorageUnit.UnitPrice;

sqlParams[3].Value = StorageUnit.UnitSize;

sqlParams[4].Value = StorageUnit.UnitArrears ? 1 : 0; // ? 1 : 0 Converts Boolean to Int16 for Database Storage

sqlParams[5].Value = StorageUnit.UnitOccupied ? 1 : 0;

sqlParams[6].Value = StorageUnit.UnitInAdvance ? 1 : 0;

sqlParams[7].Value = StorageUnit.UnitUpToDate ? 1 : 0;

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int Delete(string ID)

{

//

//Method Name : int Delete(string ID)

//Purpose : Try to delete a row from the StorageUnit datastore

//Re-use : None

//Input : string ID

// - the ID of the StorageUnit to delete in the StorageUnit datastore

//Output : - int

// 0 : StorageUnit found and deleted successfully

// -1 : StorageUnit not deleted because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // Open connection

string deleteQuery = string.Format("DELETE FROM StorageUnits WHERE [suID] = '{0}'", ID);

SQLiteCommand sqlCommand = new SQLiteCommand(deleteQuery, \_sqlCon); // setup command

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override bool DoesExist(string ID)

{

//

//Method Name : bool DoesExist(string ID)

//Purpose : Determines if a given StorageUnit exists in the StorageUnit datastore.

//Re-use : None

//Input : string ID

// - the ID of the StorageUnit to search in the StorageUnit datastore

//Output : - bool

// true : StorageUnit found

// false : StorageUnit not found

//

bool rc = false; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM StorageUnits WHERE [suID] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

rc = true;

} // end if

else

{

rc = false;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

} // end class

}//end namespace

UserProviderBase

namespace StorageUnitManagementSystem.DAL

{

public abstract class UserProviderBase

{

/// <summary>

/// This method gets the list of all the business objects from the Client datastore.

/// It returns the list of business objects

/// </summary>

public abstract List<User> SelectAll();

/// <summary>

/// This method gets a single Client object from the Client datastore.

/// It returns 0 to indicate the Client was loaded from datastore, or

/// -1 to indicate that no Client was loaded from the datastore (not found).

/// </summary>

/// <param name="ID">The ID of the Client to load from the datastore.</param>

/// <param name="User">The Client object loaded from the datastore.</param>

public abstract int SelectUser(string ID, ref User User);

/// <summary>

/// This method inserts a row in the Client datastore.

/// It returns 0 to indicate the Client was inserted into datastore, or

/// -1 to indicate the Client was not inserted because a duplicate was found

/// </summary>

/// <param name="User">The Client object to add to the Client datastore.</param>

public abstract int Insert(User User);

/// <summary>

/// This method updates a row in the Client datastore.

/// It returns 0 to indicate the Client was found and updated successfully, or

/// -1 to indicate the Client was not updated because the record was not found

/// </summary>

/// <param name="User">The new Client data for the row in the Client datastore.</param>

public abstract int Update(User User);

/// <summary>

/// This method deletes a row in the Client datastore.

/// It returns 0 to indicate the Client was found and deleted successfully, or

/// -1 to indicate the Client was not deleted because the record was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to delete in the Client datastore.</param>

public abstract int Delete(string ID);

/// <summary>

/// This method determines if a given Client exists in the Client datastore.

/// It returns true to indicate the Client was foundy, or

/// false to indicate the Client was not found

/// </summary>

/// <param name="ID">The Client ID of the Client to search in the Client datastore.</param>

public abstract bool DoesExist(string ID);

}//end class

}//end namespace

UserSQLprovider

namespace StorageUnitManagementSystem.DAL

{

public class UserSQLiteProvider : UserProviderBase

{

private static string Path = System.IO.Path.Combine(Environment.GetFolderPath(

Environment.SpecialFolder.ApplicationData), "StorageUnitManagementDB.db");

//private string \_conStr = "Data Source=c:\\DataStores\\StorageUnitManagementDB.db;Version=3;";

private SQLiteConnection \_sqlCon;

private string \_conStr = "Data Source=" + Path + ";Version=3;";

public override List<User> SelectAll()

{

//

//Method Name : List<StorageUnit> SelectAll()

//Purpose : Try to get all the StorageUnit objects from the datastore

//Re-use : None

//Input : None

//Output : - ref List<StorageUnit>

// - the list that will contain the StorageUnit objects loaded from datastore

//

List<User> User; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

User = new List<User>(); // this ensures that if there are no records,

// the returned list will not be null, but

// it will be empty (Count = 0)

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM Users";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read(); // Priming read (must have 2nd read in loop)

while (bRead == true) // false indicates no more rows/records4\4

{

User user = new User();

user.Id = Convert.ToString(sdr["UId"]);

user.UserName = Convert.ToString(sdr["UName"]);

user.Password = Convert.ToString(sdr["UPassword"]);

user.Role = Convert.ToString(sdr["UPosition"]);

User.Add(user);

bRead = sdr.Read(); // Priming read (must have 1st read before loop)

} // end while

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return User; // Single return

} // end method

public override int SelectUser(string ID, ref User User)

{

//

//Method Name : int SelectStorageUnit(string ID, ref StorageUnit StorageUnit)

//Purpose : Try to get a single StorageUnit object from the StorageUnit datastore

//Re-use :

//Input : string ID

// - The ID of the StorageUnit to load from the datastore

// ref StorageUnit StorageUnit

// - The StorageUnit object loaded from the datastore

//Output : - int

// 0 : StorageUnit loaded from datastore

// -1 : no StorageUnit was loaded from the datastore (not found)

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

User = new User();

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM Users WHERE [UId] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

User.Id = Convert.ToString(sdr["UId"]);

User.UserName = Convert.ToString(sdr["UName"]);

User.Password = Convert.ToString(sdr["UPassword"]);

User.Role = Convert.ToString(sdr["UPosition"]);

rc = 0;

} // end if

else

{

rc = -1;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int Insert(User User)

{

//

//Method Name : int Insert(StorageUnit StorageUnit)

//Purpose : Try to insert a row in the StorageUnit datastore

//Re-use : DoesExist()

//Input : StorageUnit StorageUnit

// - The StorageUnit object to add to the StorageUnit datastore

//Output : - int

// 0 : StorageUnit inserted into datastore

// -1 : StorageUnit not inserted because a duplicate was found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

bool doesExist = false;

int rowsAffected = 0;

doesExist = DoesExist(User.Id);

if (doesExist == false)

{

//TO:DO

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

\_sqlCon.Open(); // open connection

string insertQuery = "INSERT INTO Users([UId], [UserName], " +

"[UPassword], [UPosition] ) VALUES(" +

"@UId, @UName, @UPassword, @UPosition" +

")";

SQLiteCommand sqlCommand = new SQLiteCommand(insertQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@UId",DbType.String),

new SQLiteParameter("@UName",DbType.String),

new SQLiteParameter("@UPassword", DbType.String),

new SQLiteParameter("@UPosition",DbType.String),

};

sqlParams[0].Value = User.Id; // Populate SQLiteParameters from StorageUnit

sqlParams[1].Value = User.UserName;

sqlParams[3].Value = User.Password;

sqlParams[4].Value = User.Role;

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 1) // Test rowsAffected

{

// 1 row affected, thus 1 row added to datastore, thus success

rc = 0;

} // end if

} // end if

else

{

rc = -1;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override int Update(User User)

{

//

//Method Name : int Update(StorageUnit StorageUnit)

//Purpose : Try to update a row in the StorageUnit datastore

//Re-use : None

//Input : StorageUnit StorageUnit

// - The new StorageUnit data for the row in the StorageUnit datastore

//Output : - int

// 0 : StorageUnit found and updated successfully

// -1 : StorageUnit not updated because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // open connection

//

// REMEMBER: DO NOT update primary key (ID)!!!

//

//

// A better option would be to only update the fields that actually changed

//

string updateQuery = string.Format("UPDATE Users SET [UName]=@UName, " +

"[UPassword]=@UPassword,[UPosition]=@UPosition" +

" WHERE " +

"[UId] = '{0}'", User.Id);

SQLiteCommand sqlCommand = new SQLiteCommand(updateQuery, \_sqlCon); // setup command

SQLiteParameter[] sqlParams = new SQLiteParameter[] // setup parameters

{

new SQLiteParameter("@UId",DbType.String),

new SQLiteParameter("@UName",DbType.String),

new SQLiteParameter("@UPassword", DbType.String),

new SQLiteParameter("@UPosition",DbType.String),

};

sqlParams[0].Value = User.Id; // Populate SQLiteParameters from StorageUnit

sqlParams[1].Value = User.UserName;

sqlParams[3].Value = User.Password;

sqlParams[4].Value = User.Role;

sqlCommand.Parameters.AddRange(sqlParams);

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

public override int Delete(string ID)

{

//

//Method Name : int Delete(string ID)

//Purpose : Try to delete a row from the StorageUnit datastore

//Re-use : None

//Input : string ID

// - the ID of the StorageUnit to delete in the StorageUnit datastore

//Output : - int

// 0 : StorageUnit found and deleted successfully

// -1 : StorageUnit not deleted because the record was not found

//

int rc = 0; // will be returned, thus can not be declared in try block

try

{

int rowsAffected = 0;

\_sqlCon = new SQLiteConnection(\_conStr); // New connection

\_sqlCon.Open(); // Open connection

string deleteQuery = string.Format("DELETE FROM Users WHERE [UId] = '{0}'", ID);

SQLiteCommand sqlCommand = new SQLiteCommand(deleteQuery, \_sqlCon); // setup command

rowsAffected = sqlCommand.ExecuteNonQuery();

if (rowsAffected == 0) // Test rowsAffected

{

// 0 rows affected, thus NO row updated in datastore, thus not found, thus failure

rc = -1;

} // end if

else

{

// 1 row affected, thus 1 row updated in datastore, thus success

rc = 0;

} // end else

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // Single return

} // end method

public override bool DoesExist(string ID)

{

//

//Method Name : bool DoesExist(string ID)

//Purpose : Determines if a given StorageUnit exists in the StorageUnit datastore.

//Re-use : None

//Input : string ID

// - the ID of the StorageUnit to search in the StorageUnit datastore

//Output : - bool

// true : StorageUnit found

// false : StorageUnit not found

//

bool rc = false; // will be returned, thus can not be declared in try block

try

{

\_sqlCon = new SQLiteConnection(\_conStr); // new connection

bool bRead = false;

\_sqlCon.Open(); // open connection

string selectQuery = "SELECT \* FROM Users WHERE [UId] = '" + ID + "'";

SQLiteCommand sqlCommand = new SQLiteCommand(selectQuery, \_sqlCon); // setup command

SQLiteDataReader sdr = sqlCommand.ExecuteReader();

bRead = sdr.Read();

if (bRead == true) // false indicates no row/record read

{

rc = true;

} // end if

else

{

rc = false;

} // end else

sdr.Close(); // close reader

} // end try

catch (Exception ex)

{

throw ex;

} // end catch

finally

{

\_sqlCon.Close(); // Close connection

} // end finally

return rc; // single return

} // end method

}//end class

}//end namespace