# **Helios project analysis and implementation – Method 2**

1. **General**

The Graphic User Interface (GUI) layout is setup into different tabs (pages). However, on running the programme only 1 tab will be visible to the user, which can be administrator or the customer depending on the tab.

The tabs will appear sequentially in this order:

1. Login
2. Customer database
3. Insert card
4. Enter PIN
5. Menu
6. Check balance
7. Change PIN
8. Cash withdraw
9. Cash receipt

The login and customer database tabs will be visible to the administrator, but the client will be able to see only from the insert card tab window.

Implementation was done in Visual Studio 2015 .NET framework with C# programming language.

1. **Login tab**

Upon running the programme “HeliosATM.exe”, the administrator will first see this tab window. It has the button and corresponding text boxes – “User name” and “Password”. They are “helios” and “\*\*\*\*\*\*”, respectively, where the asterisks mask the password “helios”. This information is kept in these two text boxes for testing purposes, but for real-life scenario, they are kept secret by the administrator.

Two buttons are found in this tab – “Login” and “Skip >”. The former button upon successful login with user name and password will proceed to the customer database page. The latter button will help the administrator to come directly to the third tab window by skipping the customer database window.

The code snippet for login and skip > / next >:

// click skip > button to go directly to insert card tab, bypassing customer DB tab

private void btnSkip\_Click(object sender, EventArgs e)

{

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCard);

}

// if admin enters correct login name & pwd, s/he can go to customer DB tab

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

{

if (txtLogin.Text == "helios" && txtPassword.Text == "helios")

// ... password is masked in the properties

{

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCustomer);

}

else

MessageBox.Show("Invalid user name / password", "Wrong credentials entry");

}

// Next > button

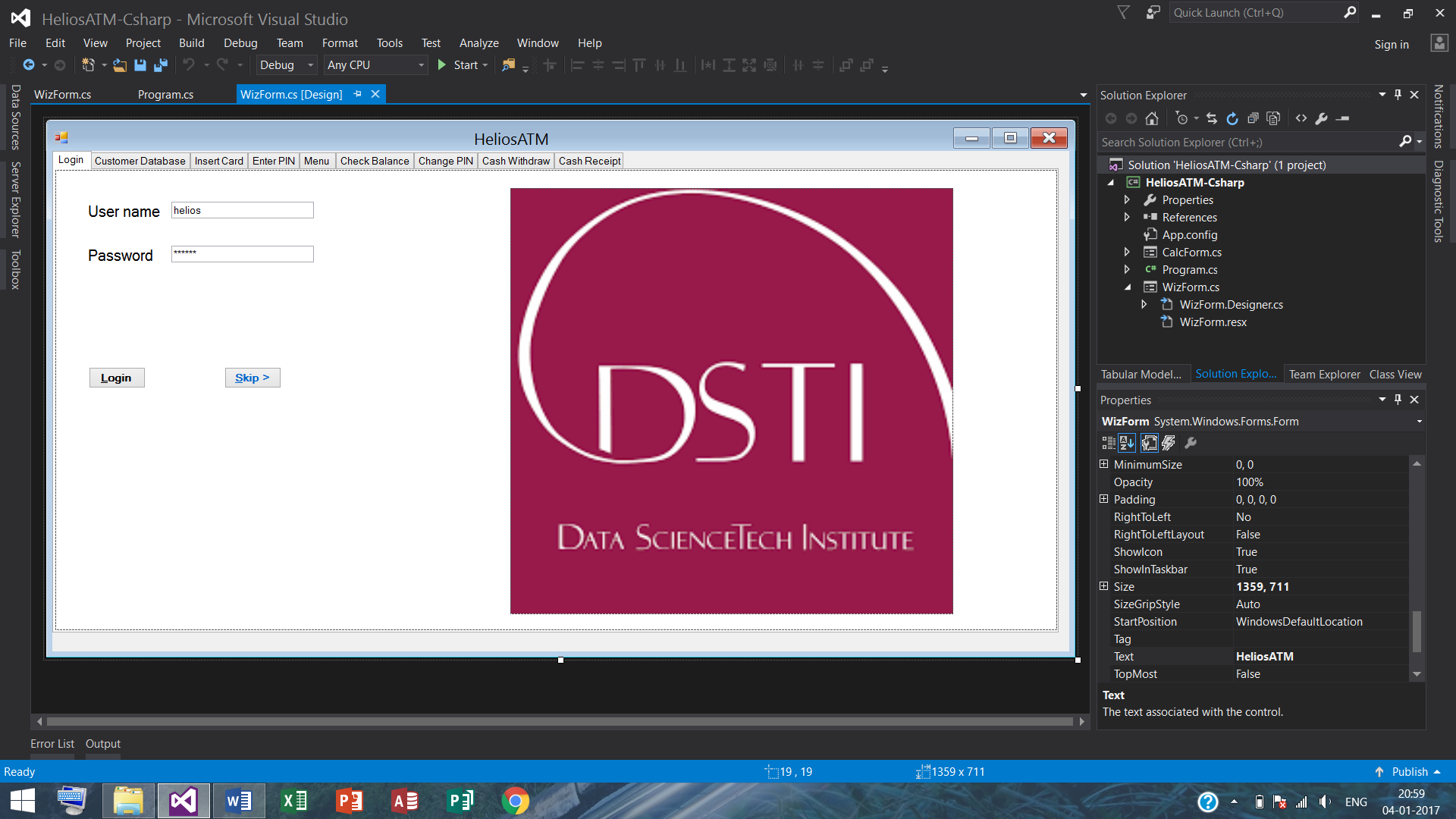
private void btnGoToInsertCard\_Click(object sender, EventArgs e)

{

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCard);

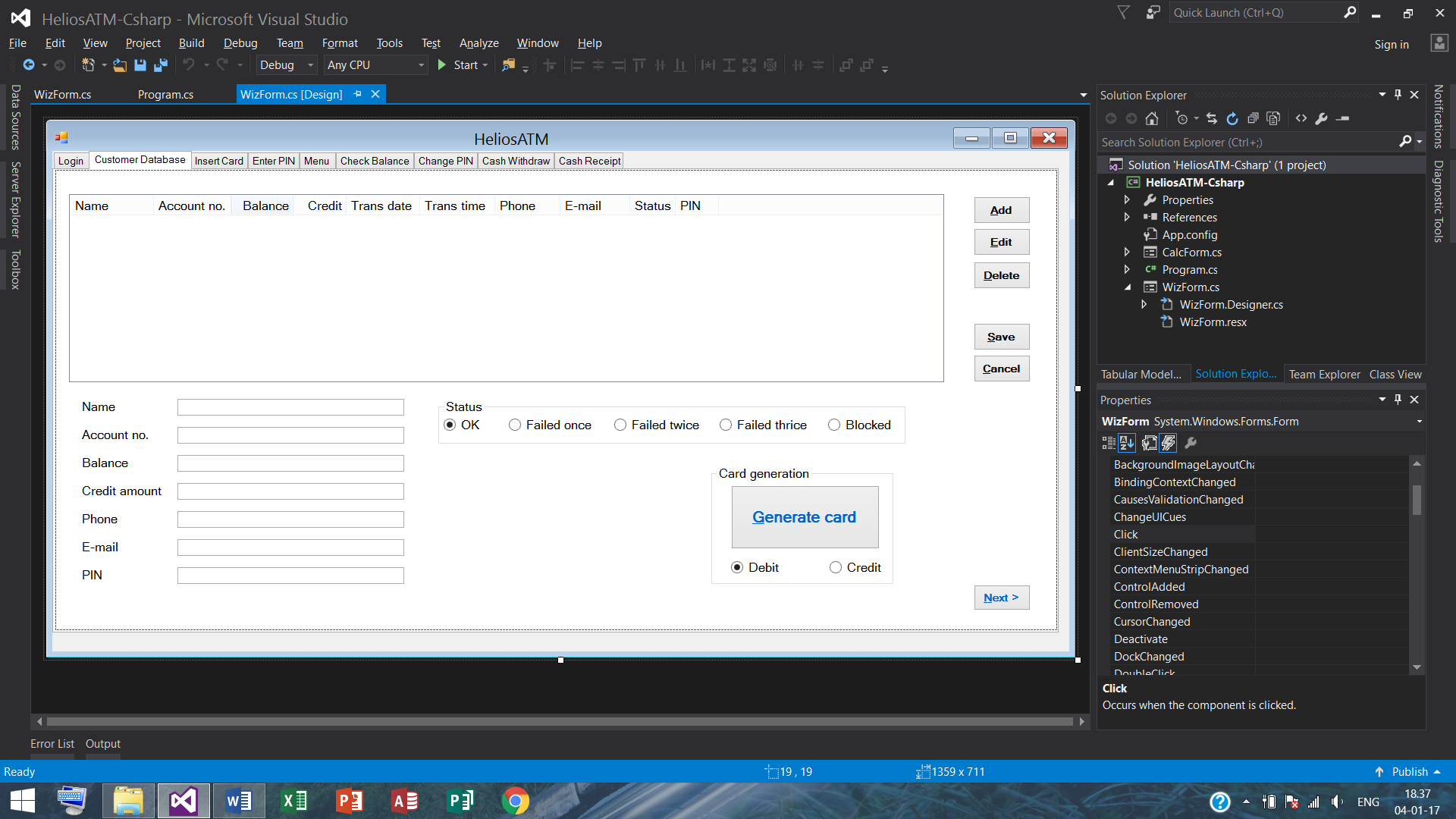
}



1. **Customer database tab**

The main feature is a ListViewItem called lstCustomer. Each row is called an item. Within each item, there are Subitems, which contain the different elements of information about the customer like name, account no., balance, credit limit, transaction date and time, phone no., email, status, PIN.

The administrator will be able to her/himself create a test profile and generate card (credit or debit) and carry out all operations that are listed later in the Menu tab. In addition, there can be a customer.dat file that will store all the previously created and newly added customer data. The customer will not be able to see this page, but the enter card page, where s/he will enter her/his ATM card to carry out the operation(s).



This consists of the following customer details as columns:

1. Name – not null
2. Account no. – not null and unique
3. Balance – not null
4. Credit – not null
5. Transaction date
6. Transaction time
7. Phone no. – not null and unique
8. E-mail
9. Status – not null
10. PIN – not null

Individual columns are described as:

1. **Name** – customer’s full name is entered.
2. **Account no.** – unique customer’s account no. is entered.
3. **Balance** – it shows the customer account balance of the customer in her/his current account from where money will be withdrawn (deducted) or deposited (added) [the latter facility is not implemented here]. In this method, we have not used a maximum withdrawal limit in the last 7 days and last 30 days. The main reason for not implementing this feature is in order to carry out such a facility, we need to have one or more withdrawals happening for each client in the last 7 days and last 30 days in a sliding manner. If the card is a debit card, the customer will be able to withdraw only a sum inferior to (or equal to) the balance shown here for simplicity purposes. The balance will be updated following a cash withdrawal operation.
4. **Credit** – it shows the customer’s allowed credit limit that the customer is able to withdraw (deduct) money from. If the card used is a credit card, the customer will be able to withdraw money a sum inferior to (or equal to) the amount plus the balance amount shown here. The credit will be updated following operations.
5. **Last transaction date** – this feature cannot be entered from the controls text boxes at the bottom left area of this tab window, but will be populated when a transaction is made and updated. The for date used is “dd/MM/yyyy”, which signifies “day/month/year” in digits.
6. **Last transaction time** – again this feature cannot be entered from the controls text boxes, but will be filled when a transaction is made and hence updated. The format for time used is “hh:mm:ss tt”, which signifies hour:minute:second in the AM/PM mode.
7. **Phone no.** – unique customer’s phone no. is entered.
8. **E-mail** – customer’s e-mail address is entered. This field may be empty.
9. **Status** – customer’s status on cash withdrawal is entered in this group of radio buttons. The status; OK – 0 (default), upon failing to enter the right PIN once, the status will be updated to 1, and in the same way upon failing twice, the status will be updated to 2 and finally 3, which signifies 3 failures. Upon 3 failures, the card will be disqualified and no further operations of any type will be possible. There is an additional special status; blocked – -1, which may arise due to the fact that the customer or the bank has blocked the card, when no operations will be possible. We have not used a resetting method based on the time of last one or two PIN entry failures in a given time, *e.g.*, last 24 or 48 hours based, which requires money withdrawal attempts and recording the corresponding transaction times at different times in a given time period. For this reason, for simplicity, we have kept the counter of status failure to be equal to 3 in order to stop any further operation and disqualify the card.
10. **PIN** – customer’s PIN is entered by the administrator. But they are not kept masked in our testing case scenario. In the real-world setting, this information is kept masked from everyone except the system administrator. Again, the user will change her/his PIN upon first use.

The buttons are:

1. **Add** – this button is active at the beginning when there is no item. Upon clicking this tab, the lstCustomer becomes active and so do the controls textboxes, the save and cancel buttons, and the card generate buttons. The administrator can then add different data in the controls text boxes and click save or cancel, as appropriate.

The code snippet for add button:

// add button properties

private void btnAdd\_Click(object sender, EventArgs e)

{

aec = 'A'; // add mode

blankControls(); // user-defined function call

enableControls(true); // user-defined function call

txtName.Focus(); // bring the cursor to the 1st text box

}

1. **Edit** – this button is active when at least one item exists and is selected. After selecting an item, the administrator can edit the controls textboxes and click save or cancel, as appropriate.

The code snippet for edit button:

// edit button properties

private void btnEdit\_Click(object sender, EventArgs e)

{

if (lstCustomer.SelectedItems.Count > 0) // if at least (only) 1 row selected

{

aec = 'E'; // edit mode

enableControls(true); // user-defined function call

txtName.Focus(); // bring the cursor to the 1st text box

}

}

1. **Delete** – this button is also active when at least one item is selected. After selecting an item, the administrator can delete the item or cancel, as appropriate.

The code snippet for delete button:

// delete button properties

private void btnDelete\_Click(object sender, EventArgs e)

{

if (lstCustomer.SelectedItems.Count > 0) // if at least (only) 1 row selected

{

if (MessageBox.Show("Are you sure to delete this row?", "Confirm delete", MessageBoxButtons.YesNo) == DialogResult.Yes)

// ... confirm before deleting customer data

{

int p = lstCustomer.SelectedIndices[0]; // of the 1st selected row (only 1)

lstCustomer.Items.RemoveAt(p); // remove selected row

blankControls(); // user-defined function call

if (lstCustomer.Items.Count > 0) // if no. of rows of customer data > 0

{

if (p == lstCustomer.Items.Count) // no. of rows of customer data

p--; // updated item count after deleting one item

lstCustomer.Items[p].Selected = true; // select the updated p-th item, send it to controls

}

enableControls(false); // user-defined function call

}

}

}

1. **Cancel** – this button is active when controls text boxes are active and data may or may not have been changed in these controls.

The code snippet for cancel button:

// cancel button properties

private void btnCancel\_Click(object sender, EventArgs e)

{

blankControls(); // user-defined function call

if (lstCustomer.SelectedItems.Count > 0) // if at least (only) 1 row selected

rowToControls(lstCustomer.SelectedIndices[0]); // user-defined function call

aec = 'C'; // default cancel mode

enableControls(false); // user-defined function call

}

1. **Save** – this button is active when controls text boxes are active and data may or may not have been changed in these controls.

The code snippet for save button:

// save button properties

private void btnSave\_Click(object sender, EventArgs e)

{

if (!isDataValid()) // user-defined function call

return;

ListViewItem item = controlsToRow(); // user-defined function call

if (aec == 'A') // if add mode

lstCustomer.Items.Add(item); // add customer data as a row (= item)

else if (lstCustomer.SelectedItems.Count > 0) // else if at least (only) 1 row selected

// ... else {edit or cancel} mode

{

int p = lstCustomer.SelectedIndices[0]; // of the 1st selected row (only 1)

string sDate, sTime;

sDate = lstCustomer.Items[p].SubItems[(int)CustColumn.TranDate].Text;

sTime = lstCustomer.Items[p].SubItems[(int)CustColumn.TranTime].Text;

item.SubItems[(int)CustColumn.TranDate].Text = sDate; // rewrite TranDate

item.SubItems[(int)CustColumn.TranTime].Text = sTime; // rewrite TranTime

lstCustomer.Items.RemoveAt(p); // remove selected row

lstCustomer.Items.Insert(p, item); // reinsert the same row, cancelled or edited

}

saveRowsToFile(); // user-defined function call

aec = 'C'; // restore default cancel mode after if add mode

enableControls(false); // user-defined function call

}

1. **Status** – this group of radio buttons’ functionalities are described before.
2. **Generate card** – as a physical true card with user information incorporated in a SIM is not possible in the current context, we have created a binary card file generating system by use of button(s). Upon clicking this generate card button and along with that selecting the ‘credit’ or ‘debit’ radio button, one can create a binary card file that will have the name format [{account no.}-D.crd] or [{account no.}-C.crd], for ‘debit’ or ‘credit’ card, respectively. This button is active when controls text boxes are active and data may or may not have been changed in these controls; the card generated after saving the customer information first will create the appropriate card, inside the cards folder under heliosData folder; where the latter contains customer database. Although we have entered some test customer data and created sample cards, the advantage of this feature is that the administrator her/himself can straight away test generate a binary card file after entering and saving a customer data; one person can have one ‘credit’ and another ‘debit’ card, but not have two or more credit cards, nor have two or more debit cards. The administrator will need to first select a row and then generate the card as described before.

The security features like encryption of card data were not done. However, we have created 3 kinds of information to add some security-like features to the binary card file. A hexadecimal signature, a ‘credit’ or ‘debit’ information – ‘0’ or ‘1’, respectively, and a placeholder that determines the sum of any digit characters in the customer’s bank account no.

The code snippet for generate card button:

// customer DB tab: generate card

private void btnGenerate\_Click(object sender, EventArgs e)

{

if (lstCustomer.SelectedItems.Count > 0) // if at least (only) 1 row selected

{

ListViewItem item = lstCustomer.SelectedItems[0]; // pick the selected item

string sFolder = Path.Combine(Customer.dataFolderName, Customer.cardFolderName);

FileStream fs;

string sAcNo = item.SubItems[1].Text; // account no. sub-item[1]

// signature components

ushort sig1 = 0xface; // hexadecimal signature

byte sig2 = (byte)(radDebit.Checked? 1 : 0); // 1:Debit / 0:Credit

int sig3 = sumDigits(item.SubItems[1].Text); // user-defined function call

// create a file stream > binary writer > close binary writer > close file stream

fs = new FileStream(Path.Combine(sFolder, sAcNo + '-' + (radDebit.Checked? 'D' : 'C') + Customer.cardExt), FileMode.Create);

BinaryWriter bw = new BinaryWriter(fs);

bw.Write(sig1); // hexadecimal signature write

bw.Write(sig2); // 1:Debit / 0:Credit write

bw.Write(sig3); // user-defined function call

bw.Write(sAcNo);

bw.Close();

fs.Close();

}

}

// function to add only the digits of a string -> sig3 component of card signature

private int sumDigits(string s)

{

int sum = 0, i;

for (i = 0; i < s.Length; i++)

if (char.IsDigit(s[i]))

sum += (s[i] - '0'); // for parsing the ASCII code = 48 for '0'

return sum;

}

1. **Insert card tab**

This is the first tab or page that the customer will see and from where will start her/his operations.

The card file replaces a physical SIM card. It is identified by the account no. and ‘D’ or ‘C’ for ‘debit’ or ‘credit’, respectively, ending in a .crd extension. This is opened with the Browse button. It is located or created as defined previously. Failure to read a valid card file or an unknown customer will result in errors – this will be printed on a message box or on the label ‘card entry message’. On finding a valid card that has the 3 kinds of information, the customer will be prompted to the next enter PIN tab.

The code snippet for insert card browse button:

// insert card tab: browse card from an external card file and open dialogue box

private void btnBrowseCard\_Click(object sender, EventArgs e)

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.InitialDirectory = Path.Combine(Customer.dataFolderName, Customer.cardFolderName);

dlg.CheckPathExists = true;

dlg.CheckFileExists = true;

dlg.Filter = "Card files (\*.crd)|\*.crd|All files (\*.\*)|\*.\*";

dlg.DefaultExt = "crd";

dlg.FilterIndex = 1;

// dlg.RestoreDirectory = true;

dlg.ReadOnlyChecked = true;

dlg.ShowReadOnly = true;

if (dlg.ShowDialog() != DialogResult.OK)

return; // cannot read/open the card

txtCardFileName.Text = dlg.FileName; // for the text box - path of the card file

if (!readCardFile(dlg.FileName, out selectedCard)) // user-defined function call

MessageBox.Show("Not a valid card.\nPlease enter a valid card.", "Invalid card");

else

{

lblCardEntryMessage.Text = (selectedCard.sig2 == 1? "Debit" : "Credit") + " card [" + selectedCard.acno + "]";

selectedCustItem = searchItemInListViewByAcNo(selectedCard.acno); // user-defined function call

if (selectedCustItem == null)

lblCardEntryMessage.Text = "Unknown customer";

else

{

int status = int.Parse(selectedCustItem.SubItems[8].Text); // current status

lblWelcome.Text = "Welcome " + selectedCustItem.SubItems[0].Text + "!";

if (status > 2 || status < 0) // card is in status 3 times failed or blocked

lblCardEntryMessage.Text += "\nYour card is blocked!";

else

{

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagPin);

txtPinEntry.Focus();

}

}

}

}

// function to read card file

private bool readCardFile(string cardFileName, out Card c) // store the parsed contents in 'c'

{

// create a file stream > binary reader > close file stream

FileStream fs = null;

c = new Card(); // intialise public user-defined public struct

try

{

fs = new FileStream(cardFileName, FileMode.Open);

// read the 3 signature items

BinaryReader br = new BinaryReader(fs);

c.sig1 = br.ReadUInt16();

if (c.sig1 != 0xface)

return false;

c.sig2 = br.ReadByte();

if (c.sig2 != 1 && c.sig2 != 0) // 1:Debit / 0:Credit

return false;

c.sig3 = br.ReadInt32();

c.acno = br.ReadString();

if (c.sig3 != sumDigits(c.acno))

return false;

return true;

}

catch (FileNotFoundException ex) // precautionary file not found error capture

{

return false;

}

finally

{

if (fs != null)

fs.Close();

}

}

// function to search item in ListView by AcNo

private ListViewItem searchItemInListViewByAcNo(string sAcNo)

{

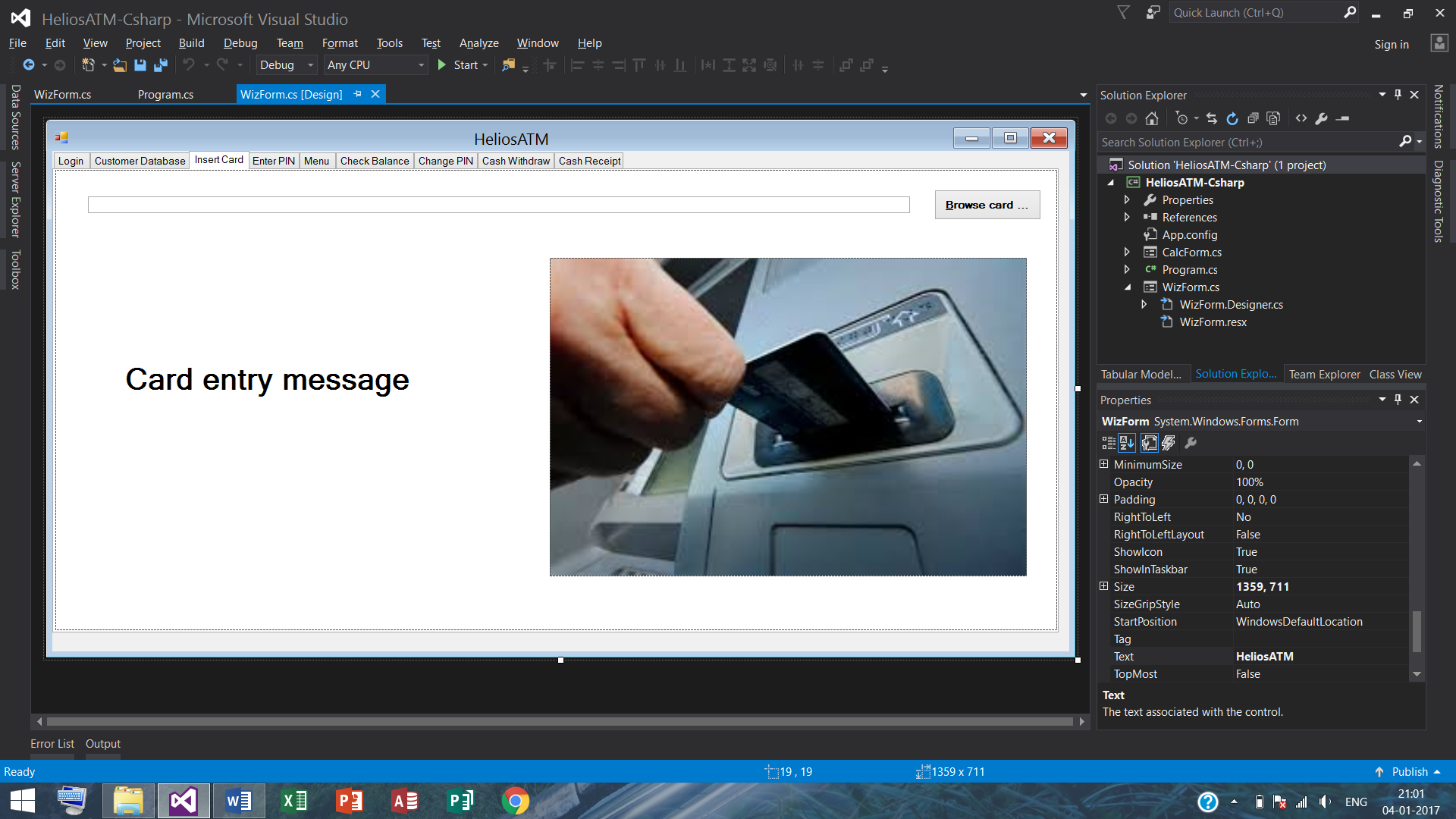
foreach (ListViewItem item in lstCustomer.Items)

if (item.SubItems[1].Text == sAcNo) // AcNo sub-item found

return item; // entire customer data

return null; // sub-item not found

}



1. **Enter PIN tab**

On the top of the screen, a successful card reading in the previous step will yield a ‘welcome message’ with the card holder’s account no. If the customer’s card is blocked she/he will get a message on the label ‘PIN result’ in addition. In the same way entering a wrong PIN will result in an error message. Successive wrong PIN entry will update the status in the database. On reaching the status = 3, meaning 3 failed PIN entries, no more card operation will be possible. If the customer before reaching 3 failed PIN entries, that is when there are 1 or 2 failed PIN entries, enters a correct PIN code, her/his status will be restored to ok (or status ‘0’).

The code snippet for Textchanged and enter PIN OK button:

// enter PIN tab: checking correct PIN and messages

private void txtPinEntry\_TextChanged(object sender, EventArgs e)

{

// clear PIN entry message upon typing a single digit

if (txtPinEntry.Text.Length > 0)

lblPinResult.Text = "";

}

// enter PIN tab: OK

// remark: maximum PIN digits are set to 4 in the properties

private void btnPinOK\_Click(object sender, EventArgs e)

{

if (txtPinEntry.Text.Length < 4 || txtPinEntry.Text.Length > 4 || (selectedCustItem.SubItems[9].Text != txtPinEntry.Text))

// ... check if wrong PIN

{

int status = int.Parse(selectedCustItem.SubItems[8].Text) + 1; // updated status

selectedCustItem.SubItems[8].Text = status.ToString(); // assign status update to selected item.sub-item

saveRowsToFile(); // user-defined function call

lblPinResult.Text = "Incorrect PIN.";

if (status >= 0 && status <= 2) // updated card status OK

{

lblPinResult.Text += "\nPlease try again.";

txtPinEntry.Text = ""; // reset PIN entry text box to empty

txtPinEntry.Focus();

}

else // updated card status not OK

{

MessageBox.Show("Your card is blocked!", "Card blocked");

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCard);

}

}

else // if PIN matches and consists of 4 digits

{

int status = int.Parse(selectedCustItem.SubItems[8].Text); // so status not updated

if (status < 0 || status > 2) // blocked or failed thrice card

{

lblPinResult.Text += "\nYour card is blocked!";

MessageBox.Show("Your card is blocked!", "Card blocked");

lblPinResult.Text = ""; // reset PIN result message box to empty

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCard);

}

else // card status ok

{

selectedCustItem.SubItems[8].Text = "0"; // assign status = OK on right PIN entry

saveRowsToFile(); // user-defined function call

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagMenu);

}

}

}

The code snippet for enter PIN cancel button:

// enter PIN tab: cancel

private void btnPinCancel\_Click(object sender, EventArgs e)

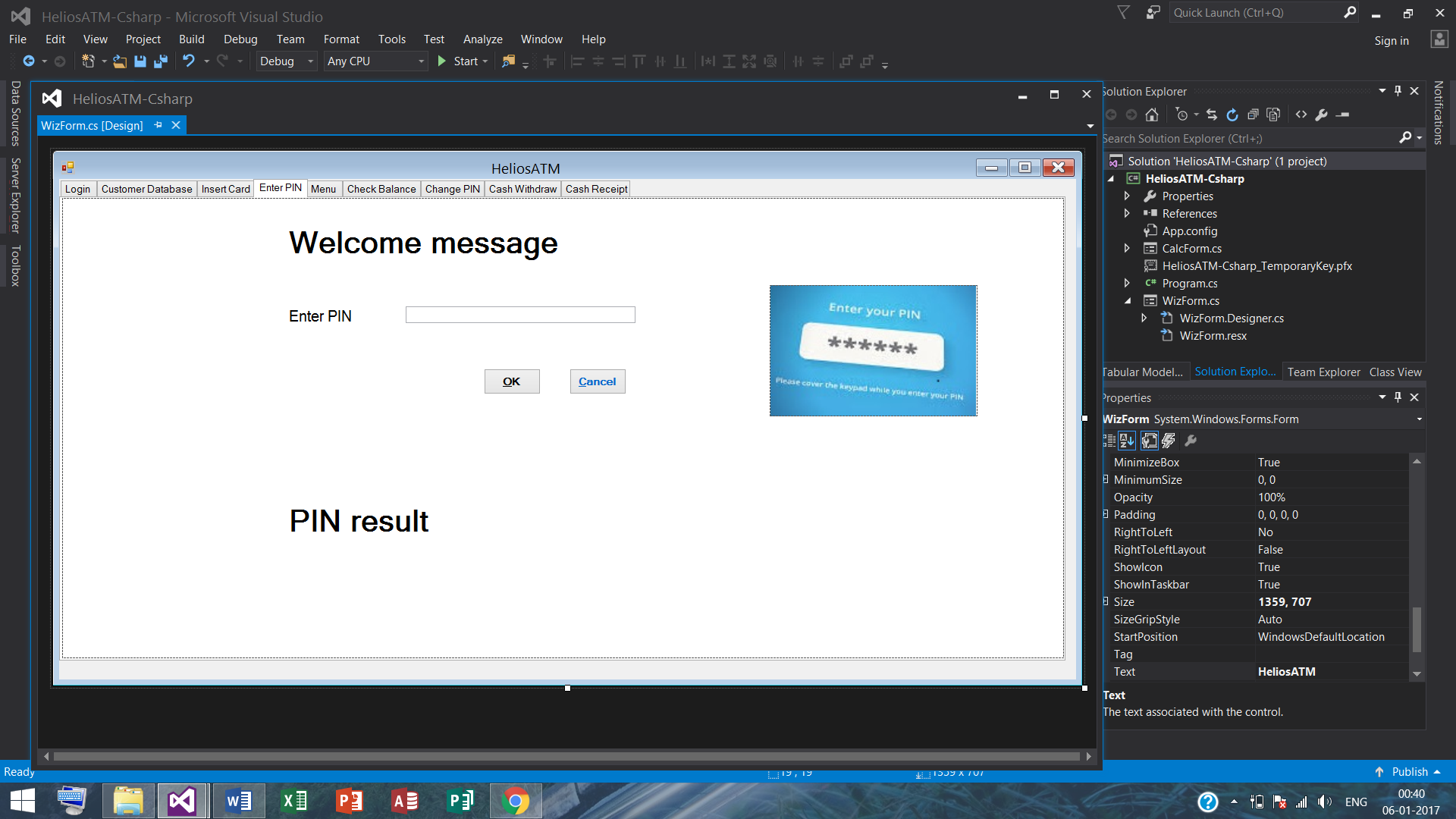
{

resetAllLabelsAndTextBoxes(); // user-defined function call

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCard);

}



1. **Menu tab**

Menu tab contains links to 3 buttons:

1. Check balance button
2. Change PIN button
3. Cash withdraw button 🡪 Cash receipt button
4. A ‘cancel’ button.

The code snippet for the Menu tabs:

// menu tab: check balance

private void btnCheckBalance\_Click(object sender, EventArgs e)

{

lblBalance.Text = selectedCustItem.SubItems[(int)CustColumn.Balance].Text;

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCheckBalance);

}

// menu tab: change PIN

private void btnChangePin\_Click(object sender, EventArgs e)

{

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagChangePin);

}

// menu tab: withdraw cash

private void btnWithdrawCash\_Click(object sender, EventArgs e)

{

mskCashWithdrawAmt.Focus();

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCashWithdraw);

}

// menu tab: close (or cancel)

private void btnMenuCancel\_Click(object sender, EventArgs e)

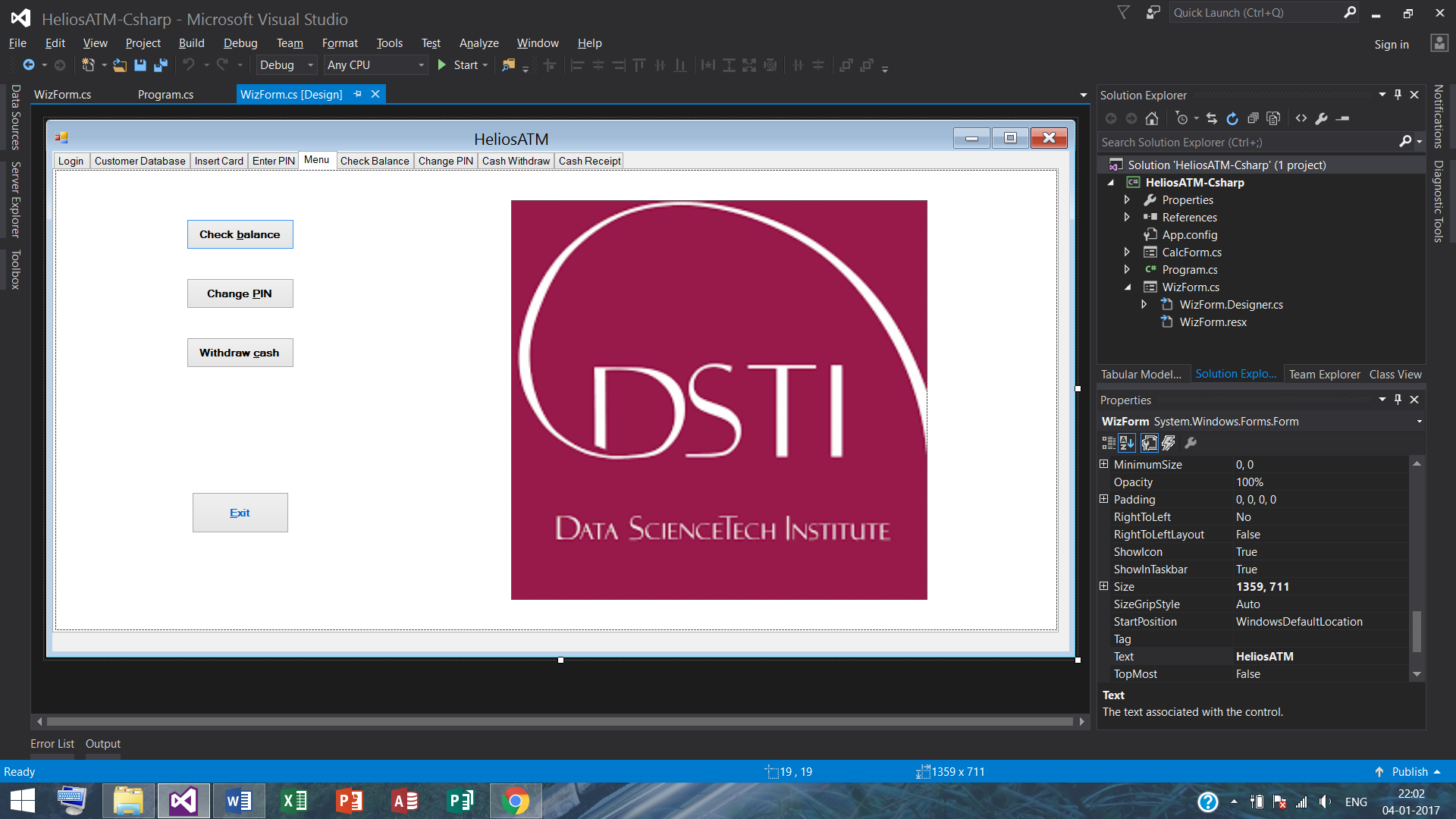
{

resetAllLabelsAndTextBoxes(); // user-defined function call

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagCard);

}



1. **Check balance**

The label ‘balance amount’ will show the current or updated balance (if the customer already took cash). On clicking OK, the customer will return to the Menu tab.

The code snippet for check balance OK button:

// check balance tab: OK

private void btnCheckBalanceOK\_Click(object sender, EventArgs e)

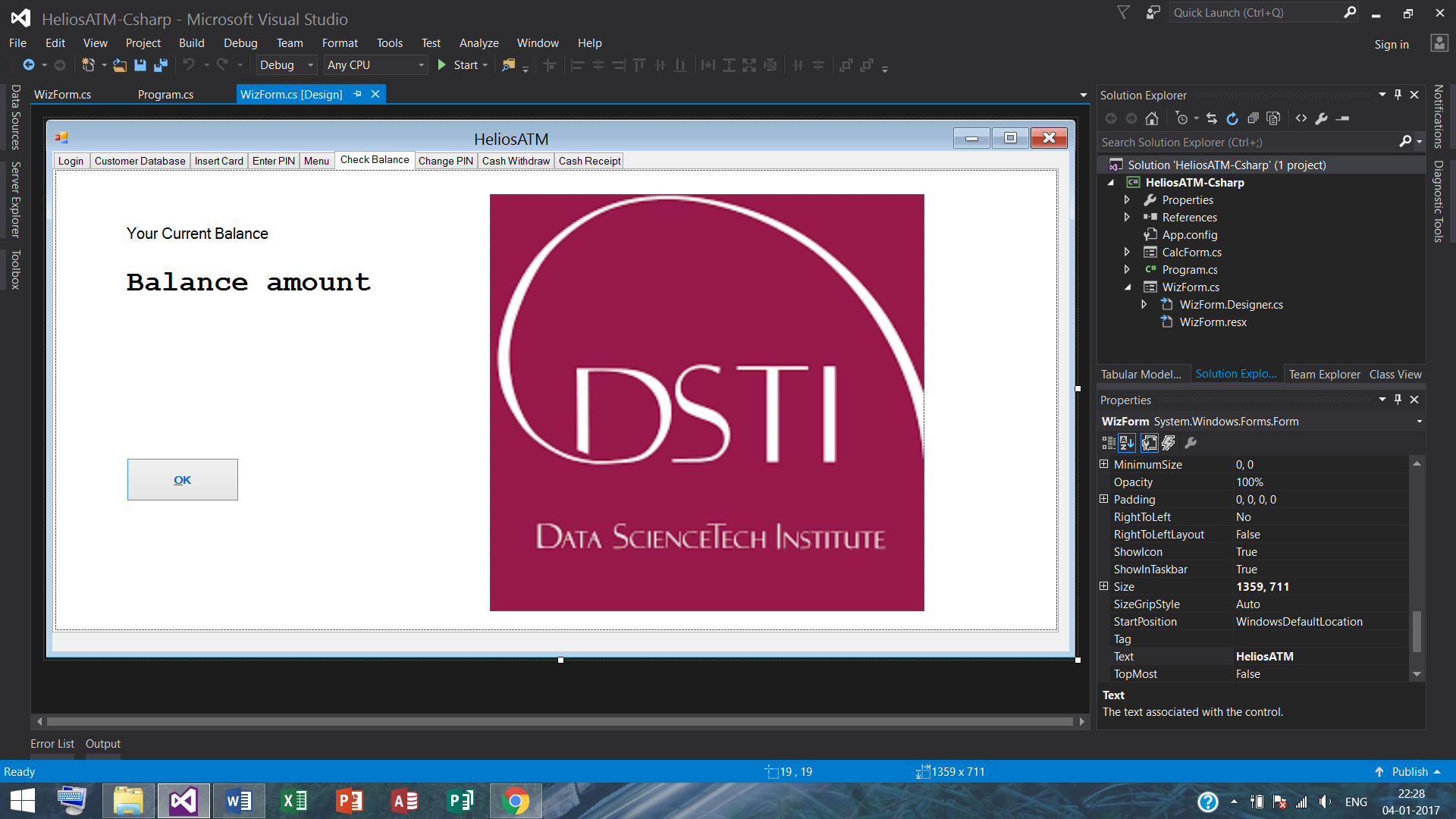
{

resetAllLabelsAndTextBoxes(); // user-defined function call

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagMenu);

}



1. **Change PIN tab**

The PIN change operation has 3 text boxes (and corresponding labels) – old PIN, new PIN, confirm new PIN. It will check for invalid PIN – alphabets, exactly 4 digits are allowed, check old PIN, check if new PIN is same as the old PIN, and finally if the new PIN and the confirmed new PIN are same. On successful PIN change, a message box will appear confirming that operation, and the customer will be prompted to the Menu tab.

The code snippet for change PIN OK and change PIN go to Menu buttons:

// change PIN tab: OK

// remark: maximum PIN digits are set to 4 in the properties

private void btnChangePinOK\_Click(object sender, EventArgs e)

{

bool isValid = false;

string sMessage = ""; // initialise message

if (txtOldPin.Text != selectedCustItem.SubItems[(int)CustColumn.Pin].Text) // use of enum

sMessage = "Incorrect PIN"; // old PIN not correct

else if (txtOldPin.Text == txtNewPin.Text)

sMessage = "New PIN must be different from Old PIN"; // old PIN != new PIN

else if (!areAllDigits(txtNewPin.Text)) // user-defined function call

sMessage = "New PIN must contain digits only"; // new PIN must contain digits only

else if (txtNewPin.Text.Length < 4)

sMessage = "New PIN must contain for digits"; // new PIN cannot have < 4 digits

else if (txtNewPin.Text != txtConfirmPin.Text)

sMessage = "New PIN must match with confirmed PIN"; // new PIN and confirmed PIN must match

else

isValid = true;

if (isValid == true)

{

selectedCustItem.SubItems[(int)CustColumn.Pin].Text = txtNewPin.Text; // use of enum

saveRowsToFile(); // user-defined function call

MessageBox.Show("PIN changed successfully", "PIN changed");

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagMenu);

}

else // show the corresponding error message according to the error

MessageBox.Show(sMessage, "PIN change error");

}

// function to check if all entered characters are digits

private bool areAllDigits(string s)

{

for (int i = 0; i < s.Length; i++)

if (!char.IsDigit(s[i]))

return false;

return (s.Length > 0); // meaning return true

}

// change PIN tab: go to menu

private void btnChangePinMenu\_Click(object sender, EventArgs e)

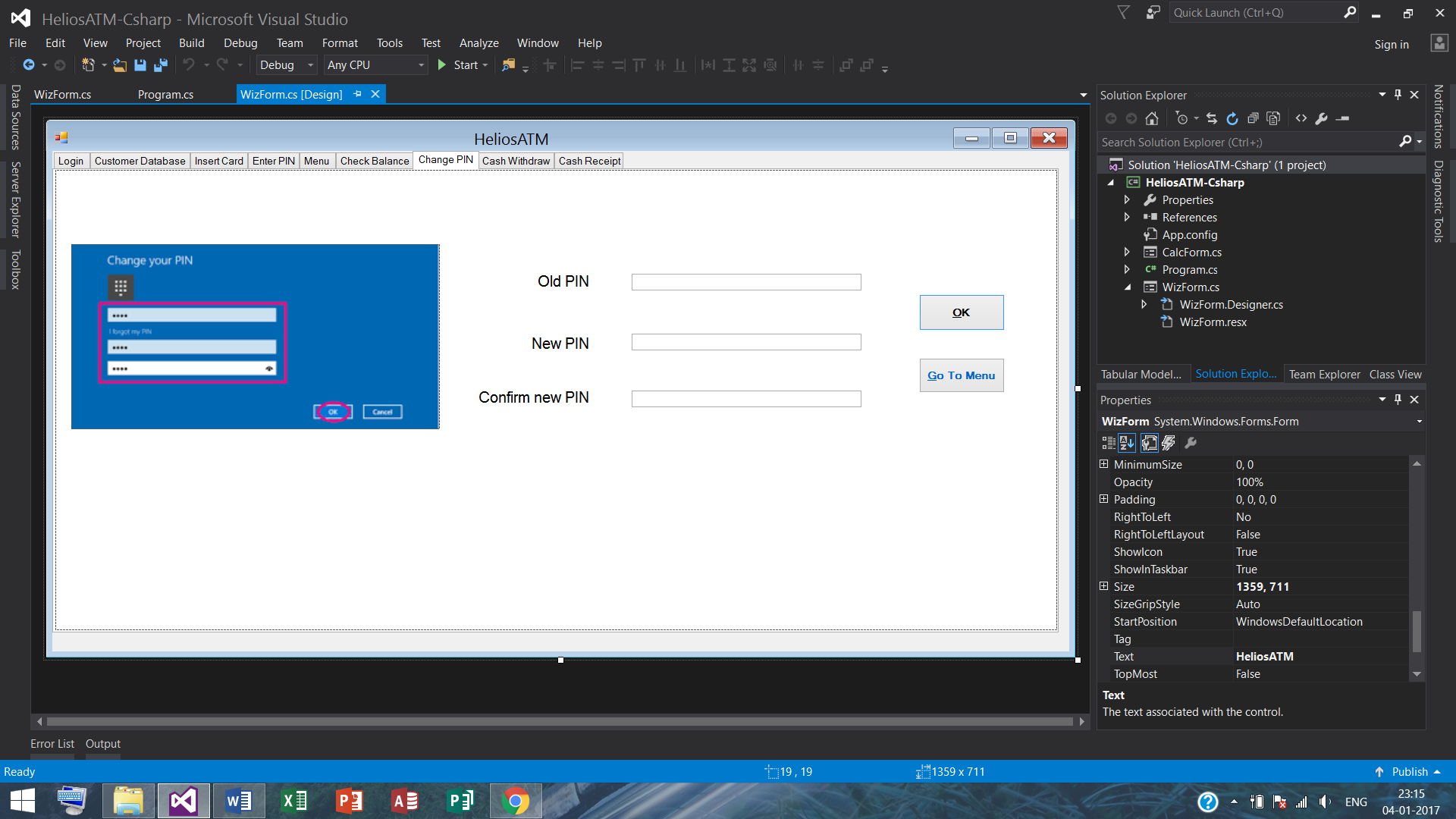
{

resetAllLabelsAndTextBoxes(); // user-defined function call

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagMenu);

}



1. **Cash withdraw tab**

The text box to enter the sum has maximum 5 digits. If wrong entry is done, it gives a beep sound. The OK button will take the customer to the cash receipt tab, and the cancel button will take to the Menu tab.

The code snippet for the cash withdraw OK and cash withdraw cancel button:

// cash withdraw tab: OK

private void btnCashWithdrawOK\_Click(object sender, EventArgs e)

{

int amt = 0, balance = 0; // initialise amt, balance

int.TryParse(mskCashWithdrawAmt.Text, out amt); // using masked text box

bool isValid = false;

string sMessage = ""; // initialise message

// balance query

if (amt <= 0)

sMessage = "Please enter a valid amount.";

else if (amt % 10 != 0)

sMessage = "Enter a sum ending in zero.";

else // valid amount query

{

balance = int.Parse(selectedCustItem.SubItems[(int)CustColumn.Balance].Text);

int creditamt = int.Parse(selectedCustItem.SubItems[(int)CustColumn.Credit].Text);

if ((selectedCard.sig2 == 1 /\* Debit \*/ && amt > balance)

|| (selectedCard.sig2 == 0 /\* Credit \*/ && amt > balance + creditamt))

sMessage = "Insufficient balance / credit limit.";

else

isValid = true;

}

// cash operation, and balance and transaction datetime update

if (isValid == true)

{

DateTime dt = DateTime.Now;

balance = balance - amt;

selectedCustItem.SubItems[(int)CustColumn.Balance].Text = balance.ToString();

selectedCustItem.SubItems[(int)CustColumn.TranDate].Text = dt.ToString("dd/MM/yyyy");

selectedCustItem.SubItems[(int)CustColumn.TranTime].Text = dt.ToString("hh:mm:ss tt");

saveRowsToFile(); // user-defined function call

MessageBox.Show("Amount debited.\nPlease collect your cash.", "Transaction successful");

lblName.Text = selectedCustItem.SubItems[(int)CustColumn.Name].Text;

lblAcNo.Text = selectedCustItem.SubItems[(int)CustColumn.AcNo].Text;

lblTranDate.Text = selectedCustItem.SubItems[(int)CustColumn.TranDate].Text;

lblTranTime.Text = selectedCustItem.SubItems[(int)CustColumn.TranTime].Text;

lblCashWithdrawn.Text = amt.ToString();

lblBalanceRemaining.Text = balance.ToString();

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagReceipt);

}

else

{

MessageBox.Show(sMessage, "Value error");

mskCashWithdrawAmt.SelectAll();

mskCashWithdrawAmt.Focus();

}

}

// cash withdraw tab: cancel

private void btnCashWithdrawCancel\_Click(object sender, EventArgs e)

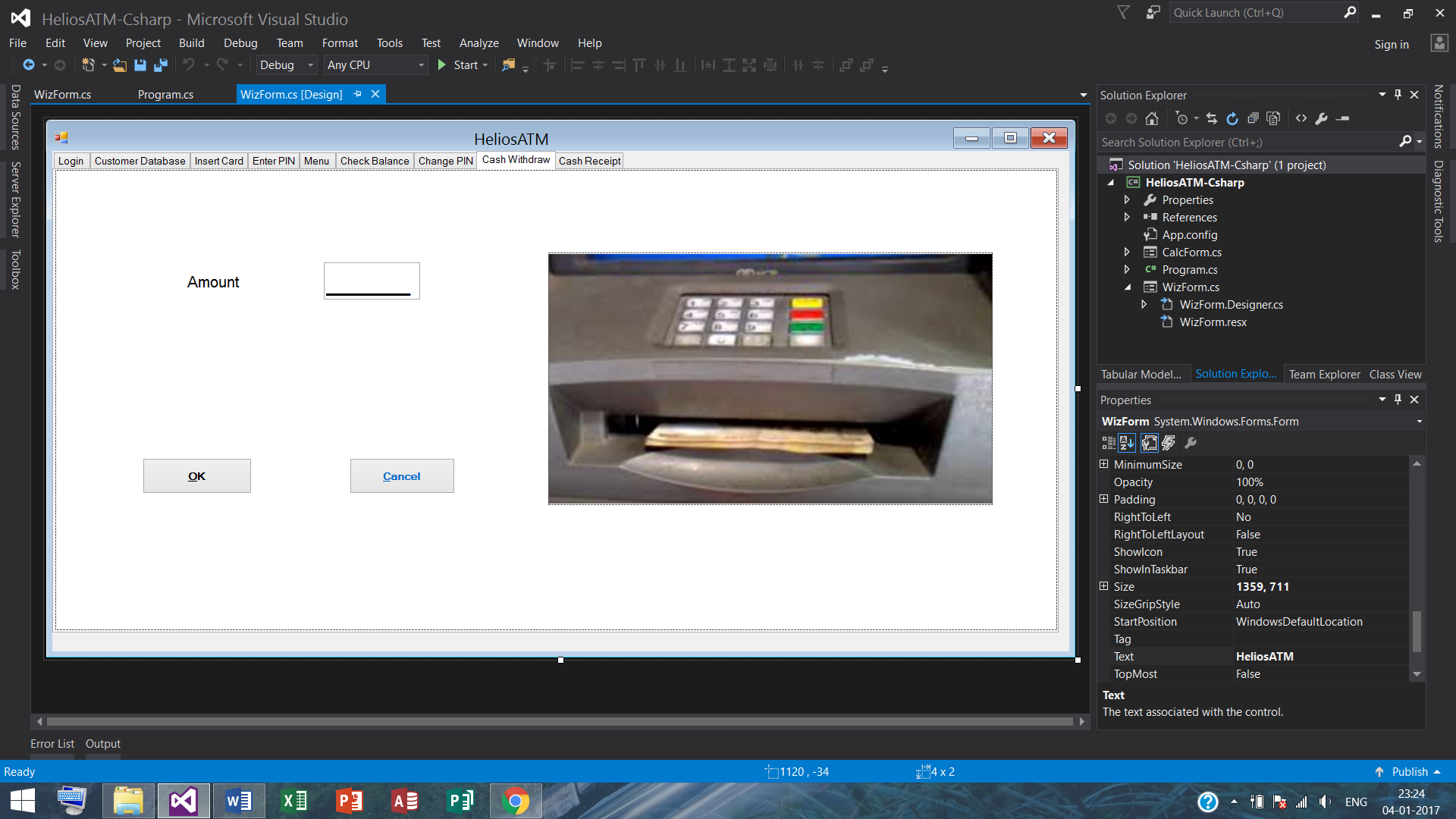
{

resetAllLabelsAndTextBoxes(); // user-defined function call

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagMenu);

}



1. **Cash receipt tab**

We simply used 6 display labels to show after a successful cash delivery a screen akin to a receipt the name, account no., transaction date, transaction time, cash withdrawn and remaining balance. The OK button will prompt the customer to the Menu tab.

The code snippet for the cash receipt OK button:

// cash receipt tab: OK

private void btnReceiptOK\_Click(object sender, EventArgs e)

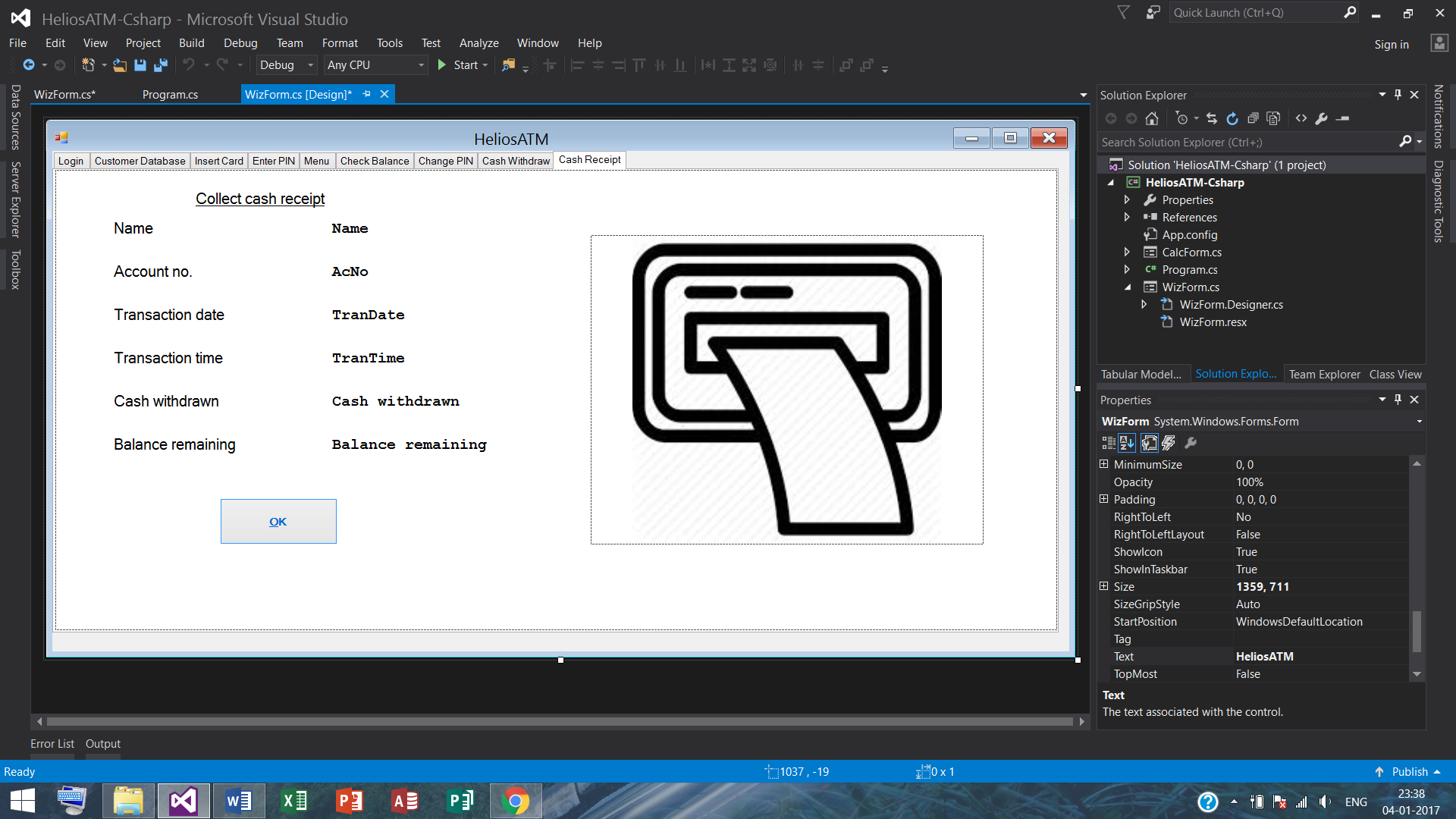
{

resetAllLabelsAndTextBoxes(); // user-defined function call

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagMenu);

}



1. **Additional codes – functions, classes, global constants, libraries, tab controls that were employed**

**namespace HeliosATM\_Csharp // project name**

{

public partial class WizForm : Form

{

// all the projects’ codes are written inside these curly braces

}

}

1. **Libraries**

// additionally required libraries

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization;

using System.Globalization;

1. **Global constants and initialisation of variables**

// initialise epoch variable as date/time is non-nullable

private readonly DateTime epoch = new DateTime();

// global variable

private Card selectedCard;

// initialise ListViewItem that will enable to select row (= item) containing customer data

private ListViewItem selectedCustItem = null;

// create a flag variable to Add or Edit or Cancel item containing customer data

private char aec = 'C'; // default value is cancel mode

// **name of the main wizard-like form, which contains different windows as tabs (= pages)**

public WizForm()

{

InitializeComponent();

}

// THE WizForm

private void WizForm\_Load(object sender, EventArgs e)

{

// initially clearing labels

lblCardEntryMessage.Text = ""; // empty message - prints card read (after card generate)

lblWelcome.Text = ""; // empty message - prints card read

lblPinResult.Text = ""; // empty message - prints PIN result

lblBalance.Text = ""; // empty message - prints balance amount

// store customer details by creating a folder that will contain external data file

DirectoryInfo dataFolder = new DirectoryInfo(Customer.dataFolderName);

// create folder to store customer details if does not exist

if (!dataFolder.Exists)

dataFolder.Create();

// store cards by creating a folder that will contain external card files

DirectoryInfo cardFolder = new DirectoryInfo(Path.Combine(Customer.dataFolderName, Customer.cardFolderName));

// create folder to store card details if does not exist (under customer details)

if (!cardFolder.Exists)

cardFolder.Create();

// show only the admin login tab & hide the subsequent ones at start

tabWiz.TabPages.Clear();

tabWiz.TabPages.Add(pagLogin);

loadRowsFromFile(); // user-defined function call

enableControls(false); // user-defined function call

// ... initialise this function at start - only Add (Edit, Delete) button(s) will remain active (upon selection)

}

1. **Functions**

// function that resets certain text & message boxes' display to empty state

private void resetAllLabelsAndTextBoxes()

{

lblCardEntryMessage.Text = "";

lblWelcome.Text = "";

lblPinResult.Text = "";

lblBalance.Text = ""; // shows current balance

txtPinEntry.Text = "";

txtOldPin.Text = "";

txtNewPin.Text = "";

txtConfirmPin.Text = "";

mskCashWithdrawAmt.Text = "";

txtCardFileName.Text = ""; // card file address bar

}

// **function to save entire customer data to an external file**

private bool saveRowsToFile()

{

if (lstCustomer.Items.Count == 0) // if no customer data

return true;

// create a file stream > binary formatter > serialise > close file stream

FileStream fs = new FileStream(Path.Combine(Customer.dataFolderName, Customer.custFileName + Customer.dataExt), FileMode.Create);

BinaryFormatter bf = new BinaryFormatter();

Customer c = null; // intialise public user-defined public class

foreach (ListViewItem item in lstCustomer.Items)

{

DateTime dt = epoch; // datetime initialise, display null if transaction datetime not found, else take user-defined style

try

{

dt = DateTime.ParseExact(item.SubItems[(int)CustColumn.TranDate].Text + " " + item.SubItems[(int)CustColumn.TranTime].Text, "dd/MM/yyyy hh:mm:ss tt", null, DateTimeStyles.None);

}

catch { } // precautionary parsing of datetime error capture

c = new Customer(item.SubItems[0].Text, item.SubItems[1].Text, int.Parse(item.SubItems[2].Text), int.Parse(item.SubItems[3].Text), dt, item.SubItems[6].Text, item.SubItems[7].Text, int.Parse(item.SubItems[8].Text), int.Parse(item.SubItems[9].Text));

bf.Serialize(fs, c); // save customer data in 'c' to the file ref. 'fs'

}

fs.Close();

return true;

}

// **function to load entire customer data from an external file**

private void loadRowsFromFile()

{

FileStream fs;

try

{

fs = new FileStream(Path.Combine(Customer.dataFolderName, Customer.custFileName + Customer.dataExt), FileMode.Open);

}

catch (FileNotFoundException ex) // precautionary file not found error capture

{

return;

}

BinaryFormatter bf = new BinaryFormatter();

Customer c = null; // intialise public user-defined public class

lstCustomer.Items.Clear(); // clear previous data before loading entire customer data

while (true)

{

try

{

c = (Customer)bf.Deserialize(fs); // get customer data in 'c' from the file ref. 'fs'

}

catch (SerializationException ex) // precautionary file opening error capture

{

c = null;

}

if (c == null)

break; // if file could not be opened

ListViewItem item = new ListViewItem(c.name);

item.SubItems.Add(c.acno);

item.SubItems.Add(c.balance.ToString());

item.SubItems.Add(c.credit.ToString());

item.SubItems.Add(c.trandate == epoch? "" : c.trandate.ToString("dd/MM/yyyy"));

item.SubItems.Add(c.trandate == epoch? "" : c.trandate.ToString("hh:mm:ss tt"));

item.SubItems.Add(c.phone);

item.SubItems.Add(c.email);

item.SubItems.Add(c.status.ToString());

item.SubItems.Add(c.pin.ToString());

lstCustomer.Items.Add(item);

}

}

// **function to send text-entry data from controls to a row in a ListViewItem:**

// - it adds the 1st field (= subitem), and then adds subsequent fields (= subitems)

// - two transaction subitems, date & time, will not be entered via controls

private ListViewItem controlsToRow()

{

ListViewItem item = new ListViewItem(txtName.Text);

item.SubItems.Add(txtAcNo.Text);

item.SubItems.Add(txtBalance.Text);

item.SubItems.Add(txtCredit.Text);

item.SubItems.Add(""); // keep empty for transaction date as it will be filled from elsewhere

item.SubItems.Add(""); // keep empty for transaction time as it will be filled from elsewhere

item.SubItems.Add(txtPhone.Text);

item.SubItems.Add(txtEmail.Text);

item.SubItems.Add(radioToStatus().ToString()); // user-defined function call

item.SubItems.Add(txtPin.Text);

return item;

}

// function for card status (as integer) generation from a radio button (as checked)

private int radioToStatus()

{

int status;

if (radOK.Checked)

status = 0;

else if (radOnce.Checked)

status = 1;

else if (radTwice.Checked)

status = 2;

else if (radThrice.Checked)

status = 3;

else // if radio button is radBlocked.Checked

status = -1;

return status;

}

// customer DB tab: if 1 row is selected, send the customer data to controls

private void lstCustomer\_SelectedIndexChanged(object sender, EventArgs e)

{

if (lstCustomer.SelectedItems.Count > 0) // if at least (only) 1 row selected

rowToControls(lstCustomer.SelectedIndices[0]); // user-defined function call

}

// **function to send customer data from a selected row to controls**

// remark: each item (row) contains several sub-items (fields)

private void rowToControls(int row)

{

ListViewItem item = lstCustomer.Items[row];

txtName.Text = item.SubItems[0].Text;

txtAcNo.Text = item.SubItems[1].Text;

txtBalance.Text = item.SubItems[2].Text;

txtCredit.Text = item.SubItems[3].Text;

txtPhone.Text = item.SubItems[6].Text;

txtEmail.Text = item.SubItems[7].Text;

statusToRadio(int.Parse(item.SubItems[8].Text)); // user-defined function call

txtPin.Text = item.SubItems[9].Text;

}

// function to populate appropriate ratio button (as checked) from a card status (as integer)

private void statusToRadio(int status)

{

switch (status)

{

case 0:

radOK.Checked = true;

break;

case 1:

radOnce.Checked = true;

break;

case 2:

radTwice.Checked = true;

break;

case 3:

radThrice.Checked = true;

break;

default: // if status is -1

radBlocked.Checked = true;

break;

}

}

// function for resetting the controls into empty text boxes

private void blankControls()

{

txtName.Text = "";

txtAcNo.Text = "";

txtBalance.Text = "";

txtCredit.Text = "";

txtPhone.Text = "";

txtEmail.Text = "";

radOK.Checked = true;

txtPin.Text = "";

}

// function for greyed out / active buttons' status as appropriate

private void enableControls(bool b)

{

lstCustomer.Enabled = !b;

txtName.Enabled = b;

txtAcNo.Enabled = b;

txtBalance.Enabled = b;

txtCredit.Enabled = b;

txtPhone.Enabled = b;

txtEmail.Enabled = b;

grpStatus.Enabled = b; // remark: {grpStatus} for the set of radio buttons

txtPin.Enabled = b;

btnSave.Enabled = b;

btnCancel.Enabled = b;

grpGenCard.Enabled = !b; // remark: {grpGenCard} for the set of button & radio buttons

btnAdd.Enabled = !b;

btnEdit.Enabled = !b && lstCustomer.Items.Count > 0; // and if at least 1 row exists

btnDelete.Enabled = !b && lstCustomer.Items.Count > 0; // and if at least 1 row exists

}

// data validation function (for save button)

// ... check non empty subitems (except e-mail and PIN sub-items), and

// ... check unique account and phone no.s

private bool isDataValid()

{

string s, g; // substitute for txtAcNo, txtPhone

// check if name is empty

txtName.Text = txtName.Text.Trim();

if (txtName.Text.Length == 0)

{

MessageBox.Show("Customer name cannot be empty.\nPlease enter a valid name", "Empty name");

txtName.Focus();

return false;

}

// check if account no. is empty

txtAcNo.Text = txtAcNo.Text.Trim();

if (txtAcNo.Text.Length == 0)

{

MessageBox.Show("Account no. cannot be empty.\nPlease enter a valid account no.", "Empty account no.");

txtAcNo.Focus();

return false;

}

// check if account no. is unique

int r;

s = txtAcNo.Text; // account no. that is going to be entered at controls

for (r = 0; r < lstCustomer.Items.Count; r++)

if (lstCustomer.Items[r].SubItems[1].Text == s) // if previous account no. == new account no.

break;

if (r < lstCustomer.Items.Count) // if value of r found

{

if (aec == 'A' || r != lstCustomer.SelectedIndices[0])

// ... or during edit mode, value of r found is different from that is being edited

{

MessageBox.Show("Duplicate account no.\nPlease enter a unique account no.", "Duplicate account no.");

txtAcNo.Focus();

return false;

}

}

// check if balance is empty

txtBalance.Text = txtBalance.Text.Trim();

if (txtBalance.Text.Length == 0)

{

MessageBox.Show("Customer balance cannot be empty.\nPlease enter a valid balance", "Empty balance");

txtBalance.Focus();

return false;

}

// check if credit is empty

txtCredit.Text = txtCredit.Text.Trim();

if (txtCredit.Text.Length == 0)

{

MessageBox.Show("Customer credit cannot be empty.\nPlease enter a valid credit", "Empty credit");

txtCredit.Focus();

return false;

}

// check if phone no. is empty

txtPhone.Text = txtPhone.Text.Trim();

if (txtPhone.Text.Length == 0)

{

MessageBox.Show("Customer phone no. cannot be empty.\nPlease enter a valid phone no.", "Empty phone no.");

txtPhone.Focus();

return false;

}

// check if phone no. is unique

int k;

g = txtPhone.Text; // phone no. that is going to be entered at controls

for (k = 0; k < lstCustomer.Items.Count; k++)

if (lstCustomer.Items[k].SubItems[7].Text == g) // if previous phone no. == new phone no.

break;

if (k < lstCustomer.Items.Count) // if value of k found

{

if (aec == 'A' || k != lstCustomer.SelectedIndices[0])

// ... or during edit mode, value of k found is different from that is being edited

{

MessageBox.Show("Duplicate phone no.\nPlease enter a unique phone no.", "Duplicate phone no.");

txtPhone.Focus();

return false;

}

}

// check if PIN contains exactly 4 (digits)

txtPin.Text = txtPin.Text.Trim();

if (txtPin.Text.Length != 4)

{

MessageBox.Show("Customer PIN must contain 4 digits.\nPlease enter a valid PIN", "Invalid PIN");

txtPin.Focus();

return false;

}

return true;

}

1. **Tab controls**

// Tab control added for enter for OK and escape for cancel / skip buttons

private void tabWiz\_ControlAdded(object sender, ControlEventArgs e)

{

Button okButton = null, cancelButton = null; //

if (e.Control == pagLogin)

{

okButton = btnLogin;

cancelButton = btnSkip;

}

else if (e.Control == pagCustomer)

cancelButton = btnGoToInsertCard;

else if (e.Control == pagCard)

okButton = btnBrowseCard;

else if (e.Control == pagPin)

{

okButton = btnPinOK;

cancelButton = btnPinCancel;

}

else if (e.Control == pagMenu)

{

okButton = btnWithdrawCash;

cancelButton = btnMenuCancel;

}

else if (e.Control == pagCheckBalance)

okButton = btnCheckBalanceOK;

else if (e.Control == pagChangePin)

{

okButton = btnChangePinOK;

cancelButton = btnChangePinMenu;

}

else if (e.Control == pagCashWithdraw)

{

okButton = btnCashWithdrawOK;

cancelButton = btnCashWithdrawCancel;

}

else if (e.Control == pagReceipt)

okButton = btnReceiptOK;

this.AcceptButton = okButton;

this.CancelButton = cancelButton;

}

1. **Classes**

// numeric equivalent for {lstCustomer} ListView columns

public enum CustColumn

{

Name, AcNo, Balance, Credit, TranDate, TranTime, Phone, Email, Status, Pin

}

// structure of a valid card file

public struct Card

{

public ushort sig1; // 0xface

public byte sig2; // 1:Debit / 0:Credit

public int sig3; // sum of digits in the account no.

public string acno;

}

// customer details file

[Serializable]

public class Customer

{

// file + extension, folder name handling

public static readonly string dataFolderName = Path.Combine(Application.StartupPath, "heliosData");

public static readonly string custFileName = "customer";

public static readonly string dataExt = ".dat";

public static readonly string cardFolderName = "cards";

public static readonly string cardExt = ".crd";

// columns

public string name;

public string acno;

public int balance;

public int credit;

public DateTime trandate;

public string phone;

public string email;

public int status;

public int pin;

public Customer(string name, string acno, int balance, int credit, DateTime trandate, string phone, string email, int status, int pin)

{

this.name = name;

this.acno = acno;

this.balance = balance;

this.credit = credit;

this.trandate = trandate;

this.phone = phone;

this.email = email;

this.status = status;

this.pin = pin;

}

}