```
using System;
using System.Windows.Forms;

namespace FBLA
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new HomeView()); // Starts the program by opening the
"HomeView" form
        }
    }
}
```

```
using FBLA.Utils;
using System;
using System.Data;
using System.Data.SQLite;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace FBLA
{
    // HomeView class is the first screen that opens when the application is
launched.
    // This form initially shows all of the data in the database. From this form,
all of the other forms can be accessed.
    public partial class HomeView : Form
    {
        public HomeView()
        {
            InitializeComponent();
            UpdateTable();
            CenterToScreen();
        }

        // this function updates the data shown in the data grid view after using on
the search function, or after records are added, edited, or deleted
        private void UpdateTable()
        {
            StringBuilder queryString = new StringBuilder("SELECT * FROM
Membership");

            string whereClause = getSearchClause();
            if (!String.IsNullOrWhiteSpace(whereClause))
                queryString.Append(" WHERE " + whereClause);

            SQLiteDataAdapter ad = DButils.getDBData(queryString.ToString(), null);
            DataTable dt = new DataTable();
            ad.Fill(dt);

            dt.Columns["id"].ColumnMapping = MappingType.Hidden;
            dt.Columns["membershipNumber"].ColumnName = "Membership Number";
            dt.Columns["firstName"].ColumnName = "First Name";
            dt.Columns["lastName"].ColumnName = "Last Name";
            dt.Columns["email"].ColumnName = "Email";
            dt.Columns["schoolGrade"].ColumnName = "School Grade";
            dt.Columns["school"].ColumnName = "School";
            dt.Columns["USstate"].ColumnName = "State";
            dt.Columns["yearJoined"].ColumnName = "Year Joined";
            dt.Columns["active"].ColumnName = "Active?";
```

```
            dt.Columns["amountOwed"].ColumnName = "Amount Owed";

            dataGridViewMembers.DataSource = dt;
            dataGridViewMembers.Columns["Amount Owed"].DefaultCellStyle.Format =
"c";

            labelRecordCount.Text = String.Format("{0}", dt.Rows.Count);
            labelRecordCount.ForeColor = (dt.Rows.Count > 0) ? Color.DarkGreen :
Color.Red;


        }

        // this function starts the search function based on the filters provided
        private void buttonSearchMember_Click(object sender, EventArgs e)
        {
            if (comboBoxField.SelectedIndex < 0)
            {
                labelSelectFieldError.Visible = true;
                return;
            }

            if (comboBoxOperator.SelectedIndex < 0)
            {
                labelSelectOperatorError.Visible = true;
                return;
            }

            if (String.IsNullOrWhiteSpace(textBoxKeyword.Text))
            {
                labelEnterKeywordError.Visible = true;
                return;
            }

            UpdateTable();

        }

        // this function retrieves the search filters
        private string getSearchClause()
        {
            if (String.IsNullOrWhiteSpace(textBoxKeyword.Text))
                return "";

            StringBuilder searchClause = new StringBuilder("");
            switch(comboBoxField.Text)
            {
                case "Membership Number":
```

```
                searchClause.Append("membershipNumber");
                break;
            case "First Name":
                searchClause.Append("firstName");
                break;
            case "Last Name":
                searchClause.Append("lastName");
                break;
            case "Email Address":
                searchClause.Append("email");
                break;
            case "School":
                searchClause.Append("school");
                break;
            case "State":
                searchClause.Append("USstate");
                break;
            default:
                return "";
        }

        switch(comboBoxOperator.Text)
        {
            case "Begins With":
                searchClause.Append(@" LIKE '" + textBoxKeyword.Text + @"%'");
                break;
            case "Ends With":
                searchClause.Append(@" LIKE '%" + textBoxKeyword.Text + @"'");
                break;
            case "Contains":
                searchClause.Append(@" LIKE '%" + textBoxKeyword.Text + @"%'");
                break;
            case "Is Exactly":
                searchClause.Append(@" = '" + textBoxKeyword.Text + @"'");
                break;
            default:
                return "";
        }

        return searchClause.ToString();

    }

    // this function is used to show all records in the database
    // this is mainly useful after the user has performed a search and want to
view the entire list of records again
    private void buttonShowAllMembers_Click(object sender, EventArgs e)
    {
```

```
            comboBoxField.SelectedIndex = -1;
            comboBoxOperator.SelectedIndex = -1;
            textBoxKeyword.Text = "";

            UpdateTable();
        }

        // this function opens the CreateReportForm window
        private void buttonCreateReport_Click(object sender, EventArgs e)
        {
            CreateReportForm createNewReport = new CreateReportForm();
            createNewReport.ShowDialog();
        }

        // this function opens the AddEditMember form in the add member mode
        private void buttonAddMember_Click(object sender, EventArgs e)
        {
            AddEditMember createNewMemberForm = new AddEditMember();
            if (createNewMemberForm.addMember())
                UpdateTable();
        }

        // this function starts the process of opening the AddEditMember form in
edit mode
        private void buttonEditMember_Click(object sender, EventArgs e)
        {
            editMember();
        }

        // this function opens the AddEditMember form in edit mode
        private void editMember()
        {
            if (dataGridViewMembers.SelectedRows.Count == 1)
            {
                var membershipNumber =
dataGridViewMembers.SelectedRows[0].Cells[0].Value.ToString();
                if (!String.IsNullOrWhiteSpace(membershipNumber))
                {
                    AddEditMember editMemberForm = new AddEditMember();
                    if (editMemberForm.editMember(membershipNumber))
                        UpdateTable();
                }
            }
            else
            {
                MessageBox.Show("Please select a member row to edit its
information", "Edit Member", MessageBoxButtons.OK);
            }
```

```
        }

        private void dataGridViewMembers_CellContentDoubleClick(object sender,
DataGridViewCellEventArgs e)
        {
            editMember();
        }

        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }

        // this function opens the About This Application page
        private void labelAbout_Click(object sender, EventArgs e)
        {
            var aboutForm = new AboutForm();
            aboutForm.ShowDialog();
        }

        // this function is used to delete a member from the database
        private void buttonDeleteMember_Click(object sender, EventArgs e)
        {
            if (dataGridViewMembers.SelectedRows.Count == 1)
            {
                var memberObject =
FBLAMember.getMemberByMembershipNumber(dataGridViewMembers.SelectedRows[0].Cells[0].
Value.ToString());

                if (memberObject != null)
                {
                    var memberId = memberObject.Id;
                    if (MessageBox.Show(String.Format("Are you sure you want to
delete member {0} {1} (#{2})? \nClick YES to delete",

dataGridViewMembers.SelectedRows[0].Cells[1].Value.ToString(),

dataGridViewMembers.SelectedRows[0].Cells[2].Value.ToString(),

dataGridViewMembers.SelectedRows[0].Cells[0].Value.ToString()), "Delete Member",
MessageBoxButtons.YesNo) == System.Windows.Forms.DialogResult.Yes)
                    {
                        if (FBLAMember.deleteMember(memberId))
                            MessageBox.Show("Member was deleted successfully",
"Delete Member", MessageBoxButtons.OK);
                        else
                            MessageBox.Show("Failed to delete member from database",
"Delete Member", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                        UpdateTable();
                    }

                }
            }
            else
            {
                MessageBox.Show("Please select a member row to edit its
information", "Edit Member", MessageBoxButtons.OK);
            }

        }
    }
}
```

```
using FBLA.Utils;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SQLite;
using System.Drawing;
using System.Windows.Forms;

namespace FBLA
{
    // AddEditMember form allows the user to add a new member or edit the data of an
existing member.
    public partial class AddEditMember : Form
    {
        private bool addMode = true;
        private int memberId = 0; //used to keep track of the member id being edited
in edit mode
        private string origMembershipNumber = ""; //used to keep track of original
number in edit mode
        public bool RecordSaved { get; private set; }

        public AddEditMember()
        {
            InitializeComponent();
            RecordSaved = false;
            CenterToParent();
        }

        // opens the form in "add member" mode, so that "save" will create a new
record in the database
        public bool addMember()
        {
            addMode = true;
            labelAddEditTitle.Text = "Add FBLA Member";
            Text = "Add FBLA Member";
            ShowDialog();
            return RecordSaved;
        }

        // opens the form in "edit member" mode, so that the form is pre-populated
with the existing data, and "save" will update the existing record
        public bool editMember(string membershipNumber)
        {
            addMode = false;
            labelAddEditTitle.Text = "Edit FBLA Member";
            Text = "Edit FBLA Member";
            var memberObject =
FBLAMember.getMemberByMembershipNumber(membershipNumber);
```

```
            if (memberObject != null)
            {
                memberId = memberObject.Id;
                origMembershipNumber = memberObject.MembershipNumber;
                textBoxMemberNum.Text = memberObject.MembershipNumber;
                textBoxFirstName.Text = memberObject.FirstName;
                textBoxLastName.Text = memberObject.LastName;
                textBoxEmail.Text = memberObject.Email;
                textBoxSchool.Text = memberObject.School;
                comboBoxSchoolGrade.Text = memberObject.SchoolGrade.ToString();
                comboBoxState.Text = memberObject.USstate;
                comboBoxYearJoined.Text = memberObject.YearJoined.ToString();
                checkBoxActiveMem.Checked = (memberObject.Active == "True");
                textBoxAmtOwed.Text = memberObject.AmountOwed.ToString();
                ShowDialog();
            }
            else
            {
                MessageBox.Show("Membership record for number" +
membershipNumber.ToString() + " not found", "Edit Member", MessageBoxButtons.OK);
            }

            return RecordSaved;
        }

        // this function will check that all of the entered data is valid, then
construct a new FBLAMember object, then save the data in the database
        private void buttonSave_Click(object sender, EventArgs e)
        {

            if (!validateAllData())
            {
                MessageBox.Show("There are some errors on the form. Please fix the
highlighted fields and try again", "Save Member", MessageBoxButtons.OK);
            }

            else
            {
                var memberToSave = new FBLAMember();

                memberToSave.MembershipNumber = textBoxMemberNum.Text;
                memberToSave.FirstName = textBoxFirstName.Text;
                memberToSave.LastName = textBoxLastName.Text;
                memberToSave.Email = textBoxEmail.Text;
                memberToSave.SchoolGrade =
(Int32.Parse(comboBoxSchoolGrade.Items[comboBoxSchoolGrade.SelectedIndex].ToString()
));
```

```
                memberToSave.School = textBoxSchool.Text;
                memberToSave.USstate =
comboBoxState.Items[comboBoxState.SelectedIndex].ToString();
                memberToSave.YearJoined = Int32.Parse(comboBoxYearJoined.Text);
                memberToSave.Active =
checkBoxActiveMem.Checked.ToString().ToTitleCase();
                memberToSave.AmountOwed =
Convert.ToDecimal(textBoxAmtOwed.Text).ToCurrency();

                if (addMode)
                {
                    RecordSaved = FBLAMember.addMember(memberToSave);
                    if (RecordSaved)
                        MessageBox.Show("Member added successfully!", "Add Member",
MessageBoxButtons.OK);
                }
                else //edit mode. Update record
                {
                    memberToSave.Id = memberId;
                    RecordSaved = FBLAMember.updateMember(memberToSave);
                    if (RecordSaved)
                        MessageBox.Show("Member updated successfully!", "Edit
Member", MessageBoxButtons.OK);
                }

                if (!RecordSaved)
                    MessageBox.Show("Member information could not be saved.  Please
check the information and try again", "Save Member", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                else //All good, just close the form
                    Close();
            }

        }

        // this function will warn the user if the text entered is not a number
        // if there is an error, the text box will change color to alert the user
        private void textBoxMemberNum_TextChanged(object sender, EventArgs e)
        {
            if (!validateMemberNumIsNum())
            {
                 textBoxMemberNum.BackColor = Color.PaleVioletRed;
            }
            else
                textBoxMemberNum.BackColor = Color.White;
        }

        // this function will warn the user if the text entered is not a unique
```

```
number if in add mode, or if the membership number is changed in the edit mode and
is not unique
        // if there is an error, the text box will change color to alert the user
        private void textBoxMemberNum_Leave(object sender, EventArgs e)
        {
            if (textBoxMemberNum.Text.Length == 0)
                return;

            bool checkUniqueNumber = false;

            if (addMode)  //user is adding a new record
                checkUniqueNumber = true;
            else  //user is editing an existing record
            {
                if (origMembershipNumber != textBoxMemberNum.Text) //user changed
the membership number
                    checkUniqueNumber = true;
            }

            if (checkUniqueNumber)
            {
                if (!validateMemberNumIsUnique())
                {
                    textBoxMemberNum.BackColor = Color.PaleVioletRed;
                }
                else
                    textBoxMemberNum.BackColor = Color.White;
            }
        }


        // this function will warn the user if there is no text entered
        // if there is an error, the text box will change color to alert the user
        private void textBoxFirstName_Leave(object sender, EventArgs e)
        {
            if (textBoxFirstName.Text.Length == 0)
                return;

            if (!validateFirstName())
            {
                textBoxFirstName.BackColor = Color.PaleVioletRed;
            }
            else
                textBoxFirstName.BackColor = Color.White;
        }

        // this function will warn the user if there is no text entered
        // if there is an error, the text box will change color to alert the user
```

```
        private void textBoxLastName_Leave(object sender, EventArgs e)
        {
            if (textBoxLastName.Text.Length == 0)
                return;

            if (!validateLastName())
            {
                textBoxLastName.BackColor = Color.PaleVioletRed;
            }
            else
                textBoxLastName.BackColor = Color.White;
        }

        // this function will warn the user if the text entered is not in proper
email format, but this field can be left blank
        // if there is an error, the text box will change color to alert the user
        private void textBoxEmail_Leave(object sender, EventArgs e)
        {
            if (!validateEmail())
            {
                textBoxEmail.BackColor = Color.PaleVioletRed;
            }
            else
                textBoxEmail.BackColor = Color.White;
        }

        // this function will warn the user if there is no text entered
        // if there is an error, the text box will change color to alert the user
        private void textBoxSchool_Leave(object sender, EventArgs e)
        {
            if (!validateSchool())
            {
                textBoxSchool.BackColor = Color.PaleVioletRed;
            }
            else
                textBoxSchool.BackColor = Color.White;
        }

        // this function will warn the user if the text entered is not a number
        // if there is an error, the text box will change color to alert the user
        private void textBoxAmtOwed_TextChanged(object sender, EventArgs e)
        {
            if (!validateAmtOwedIsNum())
            {
                textBoxAmtOwed.BackColor = Color.PaleVioletRed;
            }
            else
                textBoxAmtOwed.BackColor = Color.White;
```

```
        }

        // this function will warn the user if the text entered is not a number and
will round the number to two decimal places
        // if there is an error, the text box will change color to alert the user
        private void textBoxAmtOwed_Leave(object sender, EventArgs e)
        {
            if (!validateAmtOwedAndRound())
            {
                textBoxAmtOwed.BackColor = Color.PaleVioletRed;
            }
            else
                textBoxAmtOwed.BackColor = Color.White;
        }

        // this method is used to validate all of the data when the user clicks the
Save button
        private bool validateAllData()
        {
            bool validData = true;

            if (addMode)  //user is adding a new record
            {
                if (!validateMemberNumIsUnique())
                    validData = false;
            }
            else  //user is editing an existing record
            {
                if (origMembershipNumber != textBoxMemberNum.Text) //user changed
membership number
                {
                    if (!validateMemberNumIsUnique())
                        validData = false;
                }
            }

            if (!validateFirstName())
            {
                textBoxFirstName.BackColor = Color.PaleVioletRed;
                validData = false;
            }

            if (!validateLastName())
            {
                textBoxLastName.BackColor = Color.PaleVioletRed;
                validData = false;
            }
```

```
            if (!validateEmail())
            {
                textBoxEmail.BackColor = Color.PaleVioletRed;
                validData = false;
            }

            if (!validateSchoolGrade())
            {
                validData = false;
            }

            if (!validateSchool())
            {
                textBoxSchool.BackColor = Color.PaleVioletRed;
                validData = false;
            }

            if (!validateState())
            {
                validData = false;
            }

            if (!validateAmtOwedIsNum())
            {
                textBoxAmtOwed.BackColor = Color.PaleVioletRed;
                validData = false;
            }
            // Do not need to round AmountOwed, it already happened on "Leave"

            return validData;
        }


//*************************************************************************************
****************************

        // this function makes sure that the MembershipNumber is a number, and will
show a message if it is invalid
        private bool validateMemberNumIsNum()
        {
            labelNotUniqueMemberNum.Enabled = true;
            labelInvalidMemberNum.Enabled = true;
            // Cannot be blank, must be a UNIQUE number
            long num;
            if (String.IsNullOrWhiteSpace(textBoxMemberNum.Text))
            {
                labelInvalidMemberNum.Visible = true;
                return false;
```

```
            }
            else if (!Int64.TryParse(textBoxMemberNum.Text, out num))
            {
                labelInvalidMemberNum.Visible = true;
                return false;
            }
            else
            {
                labelInvalidMemberNum.Visible = false;
                return true;
            }
        }

        // this function makes sure that the MembershipNumber is a unique number,
and will show a message if it is invalid
        private bool validateMemberNumIsUnique()
        {
            if (!validateMemberNumIsNum())
            {
                return false;
            }

            long num = Int64.Parse(textBoxMemberNum.Text);

            List<SQLiteParameter> paramList = new List<SQLiteParameter>();
            var memberNum = new SQLiteParameter();
            memberNum.Value = num;
            paramList.Add(memberNum);

            // check database to see if number has already been used
            SQLiteDataAdapter ad = DButils.getDBData("SELECT * FROM Membership WHERE
membershipNumber=?;", paramList);

            DataTable dt = new DataTable();
            ad.Fill(dt);

            if (dt.Rows.Count != 0) // Number is already used by another member
            {
                labelNotUniqueMemberNum.Visible = true;
                return false;
            }

            labelNotUniqueMemberNum.Visible = false;
            return true;
        }

        private bool validateFirstName()
        {
```

```
            // Cannot be blank
            return (!String.IsNullOrWhiteSpace(textBoxFirstName.Text));
        }

        private bool validateLastName()
        {
            // Cannot be blank
            return (!String.IsNullOrWhiteSpace(textBoxLastName.Text));
        }

        // this function makes sure that the email is either blank or in the proper
format, and will show a message if it is invalid
        private bool validateEmail()
        {

            labelInvalidEmail.Enabled = true;

            // Can be blank or must be proper email format
            string emailAddress = textBoxEmail.Text;
            if (String.IsNullOrEmpty(emailAddress))
            {
                labelInvalidEmail.Visible = false;
                return true;
            }
            else
            {
                // Must start with letter of digit
                if (!char.IsLetterOrDigit(emailAddress,0))
                {
                    labelInvalidEmail.Visible = true;
                    return false;
                }

                int indexAtSign = emailAddress.IndexOf('@');

                // Must have only one '@' symbol
                if (indexAtSign <= 0 || indexAtSign !=
emailAddress.LastIndexOf('@'))
                {
                    labelInvalidEmail.Visible = true;
                    return false;
                }

                // Must have a dot after the '@' symbol
                int indexPeriod = emailAddress.LastIndexOf('.');

                // Must have a dot with at least one char between the '@' and the
dot
```

```
                // and with at least two char after the dot
                if (indexPeriod <= (indexAtSign + 1) || indexPeriod >=
emailAddress.Length - 2)
                {
                    labelInvalidEmail.Visible = true;
                    return false;
                }

                labelInvalidEmail.Visible = false;
                textBoxEmail.Text = emailAddress.ToLower();
                return true;
            }
        }

        private bool validateSchoolGrade()
        {
            // Need to select an option
            return (comboBoxSchoolGrade.SelectedIndex != -1);
        }

        private bool validateSchool()
        {
            // Cannot be blank
            return (!String.IsNullOrWhiteSpace(textBoxSchool.Text));
        }

        private bool validateState()
        {
            // Need to select an option
            return (comboBoxState.SelectedIndex != -1);
        }

        private bool validateAmtOwedIsNum()
        {
            // Must be a number
            decimal num;
            bool isNum = decimal.TryParse(textBoxAmtOwed.Text, out num);

            return (!String.IsNullOrWhiteSpace(textBoxAmtOwed.Text) && isNum && num
>= 0);
        }

        private bool validateAmtOwedAndRound()
        {
            // Must be a number, round to two digits
            if (validateAmtOwedIsNum())
            {
                decimal num = decimal.Parse(textBoxAmtOwed.Text);
```

```
            textBoxAmtOwed.Text = num.ToCurrency().ToString();
            return true;
        }
        else
            return false;
    }

    private void buttonCancel_Click(object sender, EventArgs e)
    {
        Close();
    }

    private void AddEditMember_Load(object sender, EventArgs e)
    {
        for (int yearCount = 0; yearCount < 10; yearCount++)
        {
            comboBoxYearJoined.Items.Add(String.Format("{0}", DateTime.Now.Year
- yearCount));
        }

        comboBoxYearJoined.SelectedIndex = 0;
    }

    private void labelAbout_Click(object sender, EventArgs e)
    {
        var aboutForm = new AboutAddEditMember();
        aboutForm.ShowDialog();
    }
  }
}
```

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Text;
using System.Windows.Forms;

namespace FBLA
{
    // CreateReportForm class shows a form where the user can select the criteria to
generate a report
    public partial class CreateReportForm : Form
    {
        // this variable is used with the "Select All Columns" checkbox, so that it
does not cause errors when updating multiple check boxes
        private bool checkingBoxes = false;

        // Constructor - sets up the form with initial criteria pre-selected
        public CreateReportForm()
        {
            InitializeComponent();
            CenterToParent();
            resetForm();
        }

        private void buttonResetForm_Click(object sender, EventArgs e)
        {
            resetForm();
        }

        // this function selects the standard filters / criteria for a report
        private void resetForm()
        {
            comboBoxFilterState.SelectedIndex = 0;
            comboBoxActiveInactive.SelectedIndex = 0;
            comboBoxAmountOwed.SelectedIndex = 0;
            checkBoxFilterFreshmen.Checked = true;
            checkBoxFilterSophomores.Checked = true;
            checkBoxFilterJuniors.Checked = true;
            checkBoxFilterSeniors.Checked = true;
            comboBoxSortBy.SelectedIndex = 0;
        }

        // this function builds the SQL query string and passes it to the ReportForm
class
        private void buttonGenerateReport_Click(object sender, EventArgs e)
        {

            StringBuilder queryString = new StringBuilder("");
```

```
            var columnList = getColumnList();
            if (String.IsNullOrWhiteSpace(columnList))
                columnList = "*";

            queryString.Append("SELECT " + columnList + " FROM Membership");

            List<SQLiteParameter> paramList = new List<SQLiteParameter>();
            var whereClause = new StringBuilder("");

            // create the "where" clause for the SQL query string based on the
filters that the user has selected
            var stateParam = getStateParam();
            if (stateParam != null)
            {
                whereClause.Append("USstate = @state");
                paramList.Add(stateParam);
            }

            var activeParam = getActiveParam();
            if (activeParam != null)
            {
                if (whereClause.Length > 0)
                    whereClause.Append(" AND ");

                whereClause.Append("active = @active");
                paramList.Add(activeParam);
            }

            var gradeList = getSchoolGradeList();
            if (!String.IsNullOrWhiteSpace(gradeList))
            {
                if (whereClause.Length > 0)
                    whereClause.Append(" AND ");

                whereClause.Append("schoolGrade IN (" + gradeList + ")");
            }

            int amtOwedParam = getAmtOwedParam();
            if (amtOwedParam != -1)
            {
                if (whereClause.Length > 0)
                    whereClause.Append(" AND ");

                if (amtOwedParam == 1)
                    whereClause.Append("amountOwed > 0");
                else
                    whereClause.Append("amountOwed = 0");
```

```
            }

            // send the pieces of the query string to the ReportForm and open / show
the report
            ReportForm report = new ReportForm(queryString.ToString(),
whereClause.ToString(), getOrderByField(), paramList);
            report.ShowDialog();
        }

        // this function gets the list of all the columns that the user has selected
to include in the report
        private string getColumnList()
        {
            StringBuilder columnList = new StringBuilder("");

            if (checkBoxSelectAllColumns.Checked)
                return "*";

            if (checkBoxMembershipNumber.Checked)
                columnList.Append("membershipNumber,");
            if (checkBoxFirstName.Checked)
                columnList.Append("firstName,");
            if (checkBoxLastName.Checked)
                columnList.Append("lastName,");
            if (checkBoxEmail.Checked)
                columnList.Append("email,");
            if (checkBoxSchoolGrade.Checked)
                columnList.Append("schoolGrade,");
            if (checkBoxSchool.Checked)
                columnList.Append("school,");
            if (checkBoxState.Checked)
                columnList.Append("USstate,");
            if (checkBoxYearJoined.Checked)
                columnList.Append("yearJoined,");
            if (checkBoxActiveInactive.Checked)
                columnList.Append("active,");
            if (checkBoxAmountOwed.Checked)
                columnList.Append("amountOwed");

            if (columnList.Length > 0)
            {
                if (columnList[columnList.Length - 1] == ',')
                    columnList.Remove(columnList.Length - 1, 1);
                columnList.Insert(0, "id,"); // insert the ID column, which is
hidden from the user but is used in within the database
            }

            return columnList.ToString();
```

```
        }

        // this functions returns the state filter that the user has selected, or
returns "null" if "All States" (index 0) was selected
        private SQLiteParameter getStateParam()
        {
            if (comboBoxFilterState.SelectedIndex > 0)
            {
                string selectedState =
comboBoxFilterState.Items[comboBoxFilterState.SelectedIndex].ToString();
                return new SQLiteParameter("@state", selectedState);
            }

            return null;
        }

        // this functions returns the filter of whether to include active members,
inactive members, or both in the report
        private SQLiteParameter getActiveParam()
        {
            switch (comboBoxActiveInactive.Text)
            {
                case "Active Only":
                    return new SQLiteParameter("@active", "True");
                case "Inactive Only":
                    return new SQLiteParameter("@active", "False");
                default:
                    return null;
            }
        }

        // this function return the filer of whether to include members who owe
money, members who do not owe money, or both in the report
        private int getAmtOwedParam()
        {
            switch (comboBoxAmountOwed.Text)
            {
                case "Amount Owed":
                    return 1;
                case "No Amount Owed":
                    return 0;
                default:
                    return -1;
            }
        }

        // this function returns the list of grades that the user wants to include
in the report
```

```
        private string getSchoolGradeList()
        {
            StringBuilder gradeList = new StringBuilder("");
            if (checkBoxFilterFreshmen.Checked)
                gradeList.Append("9,");
            if (checkBoxFilterSophomores.Checked)
                gradeList.Append("10,");
            if (checkBoxFilterJuniors.Checked)
                gradeList.Append("11,");
            if (checkBoxFilterSeniors.Checked)
                gradeList.Append("12");

            if (gradeList.Length > 0)
            {
                if (gradeList[gradeList.Length - 1] == ',')
                    gradeList.Remove(gradeList.Length - 1, 1);
            }

            return gradeList.ToString();
        }

        // this function returns which field was selected for sorting the report
data
        private string getOrderByField()
        {
            switch (comboBoxSortBy.Text)
            {
                case "State":
                    return "USstate";
                case "First Name":
                    return "firstName";
                case "Last Name":
                    return "lastName";
                case "School":
                    return "school";
                case "School Grade":
                    return "schoolGrade";
            }

            return ""; //Nothing was selected
        }

        // this function is used to update the columns selected if the "Select All
Columns" check box is checked / unchecked
        private void checkBoxSelectAllColumns_CheckedChanged(object sender,
EventArgs e)
        {
            //don't do anything because the form is in the middle of checking boxes
```

```
            if (checkingBoxes)
                return;

            checkingBoxes = true; //setting this flag so that the
checkBoxSelectAllCoumns_CheckedChanged does not execute
            checkBoxMembershipNumber.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxFirstName.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxLastName.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxEmail.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxSchool.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxSchoolGrade.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxState.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxAmountOwed.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxActiveInactive.Checked = checkBoxSelectAllColumns.Checked;
            checkBoxYearJoined.Checked = checkBoxSelectAllColumns.Checked;
            checkingBoxes = false; //I am done checking boxes
        }

        // this function is used to uncheck the "Select All Columns" check-box if
any other column check-box becomes unchecked
        private void checkBoxColumn_CheckedChanged(object sender, EventArgs e)
        {
            //don't do anything becuase the form is in the middle of checking boxes
            if (checkingBoxes)
                return;

            checkingBoxes = true; //setting this flag so that the
checkBoxSelectAllCoumns_CheckedChanged does not execute
            checkBoxSelectAllColumns.Checked = false;
            checkingBoxes = false; //I am done checking boxes
        }

        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }

        // this function allows the user to select a report preset, which includes
the two reports in the FBLA guidelines for this event
        private void comboBoxReportPresets_SelectedIndexChanged(object sender,
EventArgs e)
        {
            switch (comboBoxReportPresets.SelectedIndex)
            {
                case 0: //MasterList
                    checkingBoxes = true; //make sure the checking the boxes does
not trigger their event
                    comboBoxFilterState.SelectedIndex = 0; //all states
```

```
                    comboBoxSortBy.SelectedIndex = 0; //sort by State
                    comboBoxActiveInactive.SelectedIndex = 2; //All
                    comboBoxAmountOwed.SelectedIndex = 0; //Amount Owed
                    checkBoxSelectAllColumns.Checked = false;
                    checkBoxMembershipNumber.Checked = true;
                    checkBoxFirstName.Checked = true;
                    checkBoxLastName.Checked = true;
                    checkBoxSchoolGrade.Checked = true;
                    checkBoxState.Checked = true;
                    checkBoxAmountOwed.Checked = true;
                    checkBoxSchool.Checked = false;
                    checkBoxEmail.Checked = false;
                    checkBoxYearJoined.Checked = false;
                    checkBoxActiveInactive.Checked = false;
                    checkBoxFilterFreshmen.Checked = true;
                    checkBoxFilterSophomores.Checked = true;
                    checkBoxFilterJuniors.Checked = true;
                    checkBoxFilterSeniors.Checked = true;
                    checkingBoxes = false;
                    break;
                case 1: //List of seniors with email addresses
                    checkingBoxes = true; //make sure the checking the boxes does
not trigger their event
                    comboBoxFilterState.SelectedIndex = 0; //all states
                    comboBoxSortBy.SelectedIndex = 0; //sort by State
                    comboBoxActiveInactive.SelectedIndex = 2; //All
                    comboBoxAmountOwed.SelectedIndex = 2; //Any Amount Owed/not owed
                    checkBoxSelectAllColumns.Checked = false;
                    checkBoxMembershipNumber.Checked = true;
                    checkBoxFirstName.Checked = true;
                    checkBoxLastName.Checked = true;
                    checkBoxSchoolGrade.Checked = false;
                    checkBoxState.Checked = true;
                    checkBoxAmountOwed.Checked = false;
                    checkBoxSchool.Checked = false;
                    checkBoxEmail.Checked = true;
                    checkBoxYearJoined.Checked = false;
                    checkBoxActiveInactive.Checked = false;
                    checkBoxFilterFreshmen.Checked = false;
                    checkBoxFilterSophomores.Checked = false;
                    checkBoxFilterJuniors.Checked = false;
                    checkBoxFilterSeniors.Checked = true;
                    checkingBoxes = false;
                    break;
            }
        }

        private void labelAbout_Click(object sender, EventArgs e)
```

```
        {
            var aboutForm = new AboutCreateReport();
            aboutForm.ShowDialog();
        }
    }
}
```

```
using FBLA.Utils;
using MigraDoc.DocumentObjectModel;
using MigraDoc.DocumentObjectModel.Tables;
using MigraDoc.Rendering;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SQLite;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;

namespace FBLA
{
    // ReportForm class shows report data and allows saving/printing the report in
different formats
    public partial class ReportForm : Form
    {
        private DataTable Data; // holds entire data set for the report, to use for
export.
        private int totalRecordCount;
        private int totalPages;
        private int currentPageNum;  // keeps track of which page is currently
displayed

        private string sqlString;
        private string whereClause;
        private string orderByClause;
        private List<SQLiteParameter> ParamList;

        /****** Fields for PDF report*********/

        //Total width = 21 cm for letter size page. After 1 cm left and right
margin, usable space is 19 cm
        private double pdfPageWidth = 19;

        //These predefined column widths are percentages to use to layout the
document based on the fields selected
        private double colWidthMembershipNum = 0.108;
        private double colWidthFirstName = 0.128;
        private double colWidthLastName = 0.128;
        private double colWidthEmail = 0.157;
        private double colWidthState = 0.057;
        private double colWidthSchool = 0.148;
        private double colWidthSchoolGrade = 0.063;
        private double colWidthYearJoined = 0.063;
```

```
        private double colWidthActive = 0.063;
        private double colWidthAmountOwed = 0.085;

        //Knowing average space needed by each character will help in figuring out
if email string needs to be wrapped in pdf report
        private double avgCharWidthInCm = 0.13;

        //This field indicates how the field widths above should stretch on the
printed report to cover the whole width of the page
        private double stretchFactor = 1.0;

        // Constructor - requires all parts of query string and the parameters, to
generate the report data
        public ReportForm(string sql, string where, string orderBy,
List<SQLiteParameter> paramList)
        {
            sqlString = sql;
            whereClause = where;
            orderByClause = orderBy;
            ParamList = paramList;
            InitializeComponent();
            CenterToParent();

            // default setting for the various components of the form
            comboBoxRecordsPerPage.Text = "50"; //default 50 records per page
            currentPageNum = 1;
            textBoxCurrentPage.Text = currentPageNum.ToString();

            // show report totals at the bottom of the form
            getReportCounts();

            // show the report data in the data grid view
            ShowReportData(currentPageNum);
        }

        // this function executes the SQL query to retrieve data and show it in the
data grid view
        public void ShowReportData(int pageNumToShow)
        {
            // totalRecordCount is populated during the initialization of the form
            // use this record count to determine whether there will be report data
to show
            if (totalRecordCount > 0)
            {
                // figure out how many records to show in the grid view for use with
pagination
                int recordsPerPage = 0; // default is 0, means just show all records
in the grid view
```

```
            if (!String.IsNullOrWhiteSpace(comboBoxRecordsPerPage.Text))
            {
                if (comboBoxRecordsPerPage.Text != "ALL")
                    recordsPerPage =
Convert.ToInt32(comboBoxRecordsPerPage.Text);
            }

            // calculate total number of pages for the data grid view, based on
recordsPerPage
            if (recordsPerPage > 0)
            {
                totalPages = totalRecordCount / recordsPerPage;

                //When total records are not exactly divisible by the page size,
add an extra page for the remaining records
                if (totalRecordCount % recordsPerPage > 0)
                    totalPages++;
            }
            else
                totalPages = 1; //Records per page = 0, means all records, so
total pages equals 1

            // figure out which page to show
            if (pageNumToShow > totalPages)
                pageNumToShow = totalPages;

            if (pageNumToShow < 1)
                pageNumToShow = 1;

            currentPageNum = pageNumToShow;

            textBoxCurrentPage.Text = currentPageNum.ToString();
            labelTotalPages.Text = totalPages.ToString();

            // execute the SQL query to retrieve report data for the current
page of the data grid view
            // pagination is achieved using the LIMIT and OFFSET keywords in
SQLite
            StringBuilder sqlToRun = new StringBuilder(sqlString);

            if (!String.IsNullOrWhiteSpace(whereClause))
                sqlToRun.Append(" WHERE " + whereClause);

            if (!String.IsNullOrWhiteSpace(orderByClause))
                sqlToRun.Append(" ORDER BY " + orderByClause);

            if (recordsPerPage > 0)
                sqlToRun.Append(String.Format(" LIMIT {0} OFFSET {1}",
```

```
recordsPerPage, (recordsPerPage * (pageNumToShow - 1))));

                SQLiteDataAdapter ad = DButils.getDBData(sqlToRun.ToString(),
ParamList);

                Data = new DataTable();
                ad.Fill(Data);
                ShowData(Data);
            }
            else
                totalPages = 0; //No records to show, so total pages = 0

        }

        // this function displays the retrieved data
        private void ShowData(DataTable dt)
        {
            if (dt.Columns.Contains("id"))
            {
                dt.Columns["id"].ColumnMapping = MappingType.Hidden;
            }

            if (dt.Columns.Contains("membershipNumber"))
            {
                dt.Columns["membershipNumber"].ColumnName = "Membership Number";
            }
            if (dt.Columns.Contains("firstName"))
            {
                dt.Columns["firstName"].ColumnName = "First Name";
            }
            if (dt.Columns.Contains("lastName"))
            {
                dt.Columns["lastName"].ColumnName = "Last Name";
            }
            if (dt.Columns.Contains("email"))
            {
                dt.Columns["email"].ColumnName = "Email";
            }
            if (dt.Columns.Contains("schoolGrade"))
            {
                dt.Columns["schoolGrade"].ColumnName = "School Grade";
            }
            if (dt.Columns.Contains("school"))
            {
                dt.Columns["school"].ColumnName = "School";
            }
            if (dt.Columns.Contains("USstate"))
            {
```

```
            dt.Columns["USstate"].ColumnName = "State";
        }
        if (dt.Columns.Contains("yearJoined"))
        {
            dt.Columns["yearJoined"].ColumnName = "Year Joined";
        }
        if (dt.Columns.Contains("active"))
        {
            dt.Columns["active"].ColumnName = "Active?";
        }
        if (dt.Columns.Contains("amountOwed"))
        {
            dt.Columns["amountOwed"].ColumnName = "Amount Owed";
        }

        dataGridView1.DataSource = dt;

        if (dataGridView1.Columns.Contains("Amount Owed"))
        {
            dataGridView1.Columns["Amount Owed"].DefaultCellStyle.Format = "c";
// currency format for Amount Owed
        }

    }

    // This function calculates and displays the statistics for the report data
    private void getReportCounts()
    {
        //To get all report counts we must do a select *.  The user may not have
selected all fields for the report
        StringBuilder sqlToRun = new StringBuilder("SELECT * FROM Membership");

        if (!String.IsNullOrWhiteSpace(whereClause))
            sqlToRun.Append(" WHERE " + whereClause);

        if (!String.IsNullOrWhiteSpace(orderByClause))
            sqlToRun.Append(" ORDER BY " + orderByClause);

        SQLiteDataAdapter ad = DButils.getDBData(sqlToRun.ToString(),
ParamList);

        var dt = new DataTable();
        ad.Fill(dt);
        totalRecordCount = dt.Rows.Count;
        labelTotalRecords.Text = String.Format("{0}", totalRecordCount);
        int totalActiveMembers = 0;
        int totalInactiveMembers = 0;
        int totalMembersWithAmountOwed = 0;
```

```
            decimal totalAmountOwed = 0;
            foreach (DataRow row in dt.Rows)
            {
                if (row["active"].ToString() == "True")
                {
                    totalActiveMembers++;
                }
                else
                {
                    totalInactiveMembers++;
                }

                if (Convert.ToDecimal(row["amountOwed"]) > 0)
                {
                    totalMembersWithAmountOwed++;
                    totalAmountOwed += Convert.ToDecimal(row["amountOwed"]);
                }
            }

            labelTotalActiveMembers.Text = totalActiveMembers.ToString();
            labelTotalInactiveMembers.Text = totalInactiveMembers.ToString();
            labelTotalMembersWithAmountOwed.Text =
totalMembersWithAmountOwed.ToString();
            labelTotalAmountOwed.Text = totalAmountOwed.ToString("$#,##0.00");
        }

        // refresh the data grid view to show data for the specified page
        private void textBoxCurrentPage_Leave(object sender, EventArgs e)
        {
            refreshPageData();
        }

        private void textBoxCurrentPage_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == 13) // ENTER key pressed
                refreshPageData();
        }

        private void refreshPageData()
        {
            int pageNum = 1;
            if (Int32.TryParse(textBoxCurrentPage.Text, out pageNum))
                ShowReportData(pageNum);
        }

        private void comboBoxRecordsPerPage_SelectedIndexChanged(object sender,
EventArgs e)
        {
```

```
            //records per page has changed, re-run report at page 1
            ShowReportData(1);
        }

        private void buttonFirstPage_Click(object sender, EventArgs e)
        {
            ShowReportData(1);
        }

        private void buttonPreviousPage_Click(object sender, EventArgs e)
        {
            ShowReportData(currentPageNum - 1);
        }

        private void buttonNext_Click(object sender, EventArgs e)
        {
            ShowReportData(currentPageNum + 1);
        }

        private void buttonLast_Click(object sender, EventArgs e)
        {
            ShowReportData(totalPages); //last page
        }

        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }

        //*****************************************************
        //*          Excel and PDF export functions           *
        //*****************************************************

        // export the data to Excel (only works if Excel is installed on the user's
machine)
        private void buttonExportXls_Click(object sender, EventArgs e)
        {
            Cursor.Current = Cursors.WaitCursor;
            DialogResult saveResult = saveFileDialogExport.ShowDialog();
            if (saveResult == DialogResult.OK)
            {
                string savePath = Path.GetFullPath(saveFileDialogExport.FileName);
                ExportToExcel(savePath);
            }
            Cursor.Current = Cursors.Default;
        }
```

```
        // export data to a PDF and open the PDF
        private void buttonExportPDF_Click(object sender, EventArgs e)
        {
            Cursor.Current = Cursors.WaitCursor;
            string savePath = savePDF();
            if (!String.IsNullOrWhiteSpace(savePath))
                Process.Start(savePath);
            Cursor.Current = Cursors.Default;
        }

        // this function exports the report data to an Excel spreadsheet (.xls file)
        // *Note* this will only work if the user has Excel installed on his/her
machine
        private void ExportToExcel(string filePath)
        {
            try
            {
                if (Data == null || Data.Columns.Count == 0)
                    MessageBox.Show("There is no data to export", "Export to Excel",
MessageBoxButtons.OK);

                // load excel, and create a new workbook
                var excelApp = new Excel.Application();
                excelApp.Workbooks.Add();

                // add worksheet
                Excel._Worksheet workSheet = excelApp.ActiveSheet;

                // format the columns in Excel for each of the report fields
                int columnNumber = 1;
                foreach (DataColumn dataColumn in Data.Columns)
                {
                    switch (dataColumn.ColumnName)
                    {
                        case "Membership Number":
                        case "First Name":
                        case "Last Name":
                        case "Email":
                        case "School":
                        case "State":
                        case "Active?":
                            workSheet.Cells[1,
columnNumber].EntireColumn.NumberFormat = "@"; // text format
                            break;
                        case "School Grade":
                        case "Year Joined":
                            workSheet.Cells[1,
```

```
columnNumber].EntireColumn.NumberFormat = "General"; // general format
                            break;
                        case "Amount Owed":
                            workSheet.Cells[1,
columnNumber].EntireColumn.NumberFormat = "$#,##0.00"; // currency format
                            break;
                    }

                    // we will skip printing the id column in the excel file, so
don't increment the columnNumber
                    if (dataColumn.ColumnName != "id")
                        columnNumber++;
                }

                // add column headings, skip the id column
                for (int columnNum = 1; columnNum < Data.Columns.Count; columnNum++)
                {
                    workSheet.Cells[1, (columnNum)] =
Data.Columns[columnNum].ColumnName;
                }

                // add rows with the report data
                // In order to export all records, change the records per page to
"ALL" for the export, then change it back to original selection
                // this process makes sure that all records in the report gets added
to the file
                string recordsPerPage = comboBoxRecordsPerPage.Text;
                comboBoxRecordsPerPage.Text = "ALL";
                for (int rowNum = 0; rowNum < Data.Rows.Count; rowNum++)
                {
                    //exclude the id column, start at index = 1
                    for (int columnNum = 1; columnNum < Data.Columns.Count;
columnNum++)
                    {
                        workSheet.Cells[(rowNum + 2), (columnNum)] =
Data.Rows[rowNum][columnNum];
                    }
                }
                comboBoxRecordsPerPage.Text = recordsPerPage;

                // check file path to try to save the Excel spreadsheet
                if (!String.IsNullOrWhiteSpace(filePath))
                {
                    try
                    {
                        workSheet.SaveAs(filePath);
                        excelApp.Quit();
                        MessageBox.Show("Excel file saved successfully!", "Export to
```

```
Excel", MessageBoxButtons.OK);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("Failed to save Excel file. \nPlease make
sure you have access to the file path and that the file is not already in use.",
                            "Export to Excel", MessageBoxButtons.OK);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Failed to save Excel file. \nPlease make sure you
have access to the file path and that the file is not already in use.",
                    "Export to Excel", MessageBoxButtons.OK);
            }
        }

        // this function creates the PDF of the data retrieved for the report
        private Document CreateDocument()
        {
            // create a new MigraDoc document
            Document doc = new Document();
            doc.Info.Title = "FBLA Report";
            doc.Info.Subject = "For FBLA Desktop App Programming competition 2016";
            doc.Info.Author = "Suraj Masand";

            // each MigraDoc document needs at least one section/page
            AddNewPage(doc);
            DefineStyles(doc);
            AddReportHeader(doc);
            AddDataToPage(doc);
            AddReportFooter(doc);

            return doc;
        }

        // this function creates the styles that we will use for the content inside
the PDF
        private void DefineStyles(Document doc)
        {
            // get the predefined style Normal.
            Style style = doc.Styles["Normal"];
            style.Font.Name = "Arial Narrow";
            style = doc.Styles[StyleNames.Header];
            style.ParagraphFormat.AddTabStop("16cm",
MigraDoc.DocumentObjectModel.TabAlignment.Right);
            style = doc.Styles[StyleNames.Footer];
```

```
            style.ParagraphFormat.AddTabStop("8cm",
MigraDoc.DocumentObjectModel.TabAlignment.Center);

            // create a new style called Table based on style Normal
            style = doc.Styles.AddStyle("Table", "Normal");
            style.Font.Name = "Arial Narrow";
            style.Font.Size = 9;

            // create a new style called Reference based on style Normal
            style = doc.Styles.AddStyle("Reference", "Normal");
            style.ParagraphFormat.SpaceBefore = "2mm";
            style.ParagraphFormat.SpaceBefore = "2mm";
            style.ParagraphFormat.TabStops.AddTabStop("19cm",
MigraDoc.DocumentObjectModel.TabAlignment.Right);
        }

        // this function adds a header to the the top of the PDF
        // header contains the date / time of when the PDF report was generated
        private void AddReportHeader(Document doc)
        {
            var section = doc.LastSection; // gives us the page

            // add the print date field
            var paragraph = section.AddParagraph();
            paragraph.Style = "Reference";
            paragraph.AddTab();
            paragraph.AddText("Report Run Date: ");
            paragraph.AddDateField("dd.MM.yyyy hh:mm tt");

        }

        // this function creates a table in the page, and writes the report data to
the table
        // a maximum of 50 records are written to each page
        // a new page are created after every 50 records, and the process repeats
        private void AddDataToPage(Document doc)
        {
            int totalRowsToPrint = Data.Rows.Count;
            int totalRowsPrinted = 0;

            // we will use email field width to figure out if we need to wrap the
email address
            double spaceAvailableForEmail = colWidthEmail * stretchFactor *
pdfPageWidth;

            var section = doc.LastSection; //gives us the page
            var table = createTable(section);
```

```
            // before we can add a row, we must define the columns
            SetupPDFReportColumns(table);

            int pageRowCount = 0; // we need to reset every 50 rows, only show 50
rows per page
            foreach (DataRow dataRow in Data.Rows)
            {
                var row = table.AddRow();
                pageRowCount++;

                // once the number of records reaches 50, add a new page and setup
the columns again
                if (pageRowCount >= 50 && totalRowsPrinted < totalRowsToPrint)
                {
                    totalRowsPrinted += pageRowCount;
                    pageRowCount = 0;
                    table = createTable(AddNewPage(doc));
                    SetupPDFReportColumns(table);
                }

                // place the data in the correct columns within the table
                int columnNumber = 0;
                foreach (DataColumn dataColumn in Data.Columns)
                {
                    switch (dataColumn.ColumnName)
                    {
                        case "Membership Number":

row.Cells[columnNumber].AddParagraph(dataRow[columnNumber+1].ToString());
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "First Name":

row.Cells[columnNumber].AddParagraph(dataRow[columnNumber+1].ToString());
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "Last Name":

row.Cells[columnNumber].AddParagraph(dataRow[columnNumber+1].ToString());
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
```

```
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "Email":
                            var emailString = dataRow[columnNumber + 1].ToString();
                            double spaceNeeded = emailString.Length *
avgCharWidthInCm;
                            // add a space in the email string so MigraDoc can wrap
it in the pdf
                            if (spaceNeeded > spaceAvailableForEmail)
                                emailString = emailString.Replace("@", " @"); //
space allows wrapping email string
                            row.Cells[columnNumber].AddParagraph(emailString);
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "School Grade":

row.Cells[columnNumber].AddParagraph(dataRow[columnNumber+1].ToString());
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "School":
                            var schoolName = dataRow[columnNumber + 1].ToString();
                            // truncate school name to 33 chars to fit nicely in the
pdf
                            if (schoolName.Length > 33)
                                schoolName = schoolName.Substring(0, 33) + "...";
                            row.Cells[columnNumber].AddParagraph(schoolName);
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "State":

row.Cells[columnNumber].AddParagraph(dataRow[columnNumber+1].ToString());
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "Year Joined":
```

```
row.Cells[columnNumber].AddParagraph(dataRow[columnNumber+1].ToString());
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "Active?":

row.Cells[columnNumber].AddParagraph(dataRow[columnNumber+1].ToString());
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                        case "Amount Owed":

row.Cells[columnNumber].AddParagraph(Convert.ToDecimal(dataRow[columnNumber+1].ToStr
ing()).ToString("$##0.00")); // currency format
                            row.Cells[columnNumber].Format.Font.Bold = false;
                            row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                            columnNumber++;
                            break;
                    }
                }

            }

        }

        // this function adds a new page to the document and sets up the document
margins
        private Section AddNewPage(Document doc)
        {
            var section = doc.Sections.AddSection();
            section.PageSetup = doc.DefaultPageSetup.Clone();
            section.PageSetup.PageFormat = PageFormat.Letter;
            section.PageSetup.TopMargin = "2cm";
            section.PageSetup.LeftMargin = "1cm";
            section.PageSetup.RightMargin = "1cm";
            section.PageSetup.BottomMargin = "2cm";

            return section;
        }

        // this function creates a new table with the correct borders and style
        private Table createTable(Section section)
        {
```

```
            // create the table
            var table = section.AddTable();
            table.Style = "Table";
            table.Borders.Width = 0.25;
            table.Borders.Left.Width = 0.5;
            table.Borders.Right.Width = 0.5;
            table.Rows.LeftIndent = 0;
            return table;
        }

        // this function sets up the columns in the table so that data rows can be
added later on
        // the function checks which columns are used in the report and sets up the
columns to the appropriate width
        private void SetupPDFReportColumns(Table table)
        {

            // Since the number of columns to show in report can vary based on what
the user selected,
            // this logic calculates the factor by which to stretch each column to
fit nicely on the printed page
            double totalWidthUsed = 0.0;
            foreach (DataColumn dataColumn in Data.Columns)
            {
                switch (dataColumn.ColumnName)
                {
                    case "Membership Number":
                        totalWidthUsed += colWidthMembershipNum;
                        break;
                    case "First Name":
                        totalWidthUsed += colWidthFirstName;
                        break;
                    case "Last Name":
                        totalWidthUsed += colWidthLastName;
                        break;
                    case "Email":
                        totalWidthUsed += colWidthEmail;
                        break;
                    case "School Grade":
                        totalWidthUsed += colWidthSchoolGrade;
                        break;
                    case "School":
                        totalWidthUsed += colWidthSchool;
                        break;
                    case "State":
                        totalWidthUsed += colWidthState;
                        break;
                    case "Year Joined":
```

```
                    totalWidthUsed += colWidthYearJoined;
                    break;
                case "Active?":
                    totalWidthUsed += colWidthActive;
                    break;
                case "Amount Owed":
                    totalWidthUsed += colWidthAmountOwed;
                    break;
            }
        }

        if (totalWidthUsed > 0.00)
            stretchFactor = 1 / totalWidthUsed;
        else
            stretchFactor = 1;

        // before creating a row in the table, all the columns need to be
defined and added
        Column column = null;
        foreach (DataColumn dataColumn in Data.Columns)
        {
            switch (dataColumn.ColumnName)
            {
                case "Membership Number":
                    column = table.AddColumn(String.Format("{0}cm",
colWidthMembershipNum * stretchFactor * pdfPageWidth));
                    column.Format.Alignment = ParagraphAlignment.Center;
                    break;
                case "First Name":
                    column = table.AddColumn(String.Format("{0}cm",
colWidthFirstName * stretchFactor * pdfPageWidth));
                    column.Format.Alignment = ParagraphAlignment.Center;
                    break;
                case "Last Name":
                    column = table.AddColumn(String.Format("{0}cm",
colWidthLastName * stretchFactor * pdfPageWidth));
                    column.Format.Alignment = ParagraphAlignment.Center;
                    break;
                case "Email":
                    column = table.AddColumn(String.Format("{0}cm",
colWidthEmail * stretchFactor * pdfPageWidth));
                    column.Format.Alignment = ParagraphAlignment.Center;
                    break;
                case "School Grade":
                    column = table.AddColumn(String.Format("{0}cm",
colWidthSchoolGrade * stretchFactor * pdfPageWidth));
                    column.Format.Alignment = ParagraphAlignment.Center;
                    break;
```

```
                    case "School":
                        column = table.AddColumn(String.Format("{0}cm",
colWidthSchool * stretchFactor * pdfPageWidth));
                        column.Format.Alignment = ParagraphAlignment.Center;
                        break;
                    case "State":
                        column = table.AddColumn(String.Format("{0}cm",
colWidthState * stretchFactor * pdfPageWidth));
                        column.Format.Alignment = ParagraphAlignment.Center;
                        break;
                    case "Year Joined":
                        column = table.AddColumn(String.Format("{0}cm",
colWidthYearJoined * stretchFactor * pdfPageWidth));
                        column.Format.Alignment = ParagraphAlignment.Center;
                        break;
                    case "Active?":
                        column = table.AddColumn(String.Format("{0}cm",
colWidthActive * stretchFactor * pdfPageWidth));
                        column.Format.Alignment = ParagraphAlignment.Center;
                        break;
                    case "Amount Owed":
                        column = table.AddColumn(String.Format("{0}cm",
colWidthAmountOwed * stretchFactor * pdfPageWidth));
                        column.Format.Alignment = ParagraphAlignment.Right;
                        break;
                }
            }

            // now create the row and add headings
            Row row = table.AddRow();
            row.HeadingFormat = true;
            row.Format.Alignment = ParagraphAlignment.Center;
            row.Format.Font.Bold = true;
            row.Shading.Color = MigraDoc.DocumentObjectModel.Colors.LightGray;

            int columnNumber = 0;
            foreach (DataColumn dataColumn in Data.Columns)
            {
                switch (dataColumn.ColumnName)
                {
                    // before creating a header row, columns need to be defined
                    case "Membership Number":
                        row.Cells[columnNumber].AddParagraph("Membership #");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
```

```
                    case "First Name":
                        row.Cells[columnNumber].AddParagraph("First Name");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                    case "Last Name":
                        row.Cells[columnNumber].AddParagraph("Last Name");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                    case "Email":
                        row.Cells[columnNumber].AddParagraph("Email");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                    case "School Grade":
                        row.Cells[columnNumber].AddParagraph("School Grade");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                    case "School":
                        row.Cells[columnNumber].AddParagraph("School");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                    case "State":
                        row.Cells[columnNumber].AddParagraph("State");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                    case "Year Joined":
                        row.Cells[columnNumber].AddParagraph("Yr. Joined");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
```

```
                        break;
                    case "Active?":
                        row.Cells[columnNumber].AddParagraph("Active?");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                    case "Amount Owed":
                        row.Cells[columnNumber].AddParagraph("Amount Owed");
                        row.Cells[columnNumber].Format.Font.Bold = false;
                        row.Cells[columnNumber].VerticalAlignment =
VerticalAlignment.Center;
                        columnNumber++;
                        break;
                }
            }
        }

        // this function adds the footer data to the end of the report
        private void AddReportFooter(Document doc)
        {
            // create footer
            Paragraph paragraph = doc.LastSection.AddParagraph();
            paragraph.AddLineBreak();
            paragraph.AddLineBreak(); // Add space between table and footer
            string footerData = "Total Number of Members in Report: " +
labelTotalRecords.Text + "\t" +
                "Total Active Members: " + labelTotalActiveMembers.Text + "\t" +
                "Total Inactive Members: " + labelTotalInactiveMembers.Text + "\n" +

                "Total Members With Amount Owed: " +
labelTotalMembersWithAmountOwed.Text + "\t" +
                "Total Amount Owed: " + labelTotalAmountOwed.Text;
            paragraph.AddText(footerData);
            paragraph.Format.Font.Size = 9;
            paragraph.Format.Alignment = ParagraphAlignment.Center;
        }

        // this function allows the user to export / save the PDF report
        private string savePDF()
        {
            // store currently selected records per page, change to "ALL" for the
export, change it back to original selection
            // this process makes sure that all records in the report gets added to
the PDF
            string recordsPerPage = comboBoxRecordsPerPage.Text;
            comboBoxRecordsPerPage.Text = "ALL";
```

```
            var doc = CreateDocument();
            comboBoxRecordsPerPage.Text = recordsPerPage;

            PdfDocumentRenderer renderer = new PdfDocumentRenderer(true,
PdfSharp.Pdf.PdfFontEmbedding.Always);
            renderer.Document = doc;
            renderer.RenderDocument();

            // save the document
            DialogResult saveResult = saveFileDialogPDF.ShowDialog();
            if (saveResult == DialogResult.OK)
            {
                string savePath = Path.GetFullPath(saveFileDialogPDF.FileName);

                try
                {
                    renderer.PdfDocument.Save(savePath);
                    return savePath;
                }
                catch
                {
                    MessageBox.Show("There was an error when trying to save the
file." +
                    "\nThe file may be in use by another application.", "Save
Error", MessageBoxButtons.OK);
                }
            }

            return "";
        }

        private void labelAbout_Click(object sender, EventArgs e)
        {
            var aboutForm = new AboutGeneratedReport();
            aboutForm.ShowDialog();
        }
    }
}
```

```
using FBLA.Utils;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SQLite;

namespace FBLA
{
    // FBLAMember class represents the member object for each member in the system
    public class FBLAMember
    {
        // Properties of the FBLAMember class
        public int Id { get; set; }
        public string MembershipNumber { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string School { get; set; }
        public string USstate { get; set; }
        public int SchoolGrade { get; set; }
        public string Active { get; set; }
        public int YearJoined { get; set; }
        public decimal AmountOwed { get; set; }

        // Gets the member object by the membership number
        public static FBLAMember getMemberByMembershipNumber(string
membershipNumber)
        {
            FBLAMember member = null;

            string queryString = "SELECT * FROM Membership WHERE membershipNumber =
@membershipNumber";

            List<SQLiteParameter> paramList = new List<SQLiteParameter>();
            paramList.Add(new SQLiteParameter("@membershipNumber",
membershipNumber));

            SQLiteDataAdapter ad = DButils.getDBData(queryString.ToString(),
paramList);

            DataSet ds = new DataSet();
            ad.Fill(ds, "fblaMember");
            if (ds.Tables["fblaMember"].Rows.Count == 1) // we found one record with
this membership number
            {
                member = new FBLAMember();
                member.Id = Convert.ToInt32(ds.Tables["fblaMember"].Rows[0]["id"]);
                member.MembershipNumber =
```

```
ds.Tables["fblaMember"].Rows[0]["membershipNumber"].ToString();
                member.FirstName =
ds.Tables["fblaMember"].Rows[0]["firstName"].ToString();
                member.LastName =
ds.Tables["fblaMember"].Rows[0]["lastName"].ToString();
                member.Email = ds.Tables["fblaMember"].Rows[0]["email"].ToString();
                member.School =
ds.Tables["fblaMember"].Rows[0]["school"].ToString();
                member.USstate =
ds.Tables["fblaMember"].Rows[0]["USstate"].ToString();
                member.SchoolGrade =
Convert.ToInt32(ds.Tables["fblaMember"].Rows[0]["schoolGrade"]);
                member.AmountOwed =
Convert.ToDecimal(ds.Tables["fblaMember"].Rows[0]["amountOwed"]);
                member.Active =
ds.Tables["fblaMember"].Rows[0]["active"].ToString();
                member.YearJoined =
Convert.ToInt32(ds.Tables["fblaMember"].Rows[0]["yearJoined"]);
            }

            return member;
        }

        // Adds a new member to the database
        public static bool addMember(FBLAMember newMember)
        {
            string insertNewMemberQuery = "INSERT INTO Membership " +
                "(membershipNumber, firstName, lastName, email, schoolGrade, school,
USstate, yearJoined, active, amountOwed) " +
                "VALUES (@memberNum, @fName, @lName, @email, @schoolGrade,
@school,@USstate, @yearJoined, @active, @amountOwed)";

            List<SQLiteParameter> paramList = new List<SQLiteParameter>();
            paramList.Add(new SQLiteParameter("@memberNum",
newMember.MembershipNumber));
            paramList.Add(new SQLiteParameter("@fName", newMember.FirstName));
            paramList.Add(new SQLiteParameter("@lName", newMember.LastName));
            paramList.Add(new SQLiteParameter("@email", newMember.Email));
            paramList.Add(new SQLiteParameter("@schoolGrade",
newMember.SchoolGrade));
            paramList.Add(new SQLiteParameter("@school", newMember.School));
            paramList.Add(new SQLiteParameter("@USstate", newMember.USstate));
            paramList.Add(new SQLiteParameter("@yearJoined", newMember.YearJoined));
            paramList.Add(new SQLiteParameter("@active", newMember.Active));
            paramList.Add(new SQLiteParameter("@amountOwed", newMember.AmountOwed));

            return DButils.storeData(insertNewMemberQuery, paramList);
        }
```

```
        // Updates an existing member in the database
        public static bool updateMember(FBLAMember newMember)
        {
            string updateMemberQuery = "UPDATE Membership SET membershipNumber =
@memberNum, " +
            "firstName = @fName, " +
            "lastName = @lName, " +
            "email = @email, " +
            "schoolGrade = @schoolGrade, " +
            "school = @school, " +
            "USstate = @USstate, " +
            "yearJoined = @yearJoined, " +
            "active = @active, " +
            "amountOwed = @amountOwed " +
            "WHERE id = @id";

            List<SQLiteParameter> paramList = new List<SQLiteParameter>();
            paramList.Add(new SQLiteParameter("@memberNum",
newMember.MembershipNumber));
            paramList.Add(new SQLiteParameter("@fName", newMember.FirstName));
            paramList.Add(new SQLiteParameter("@lName", newMember.LastName));
            paramList.Add(new SQLiteParameter("@email", newMember.Email));
            paramList.Add(new SQLiteParameter("@schoolGrade",
newMember.SchoolGrade));
            paramList.Add(new SQLiteParameter("@school", newMember.School));
            paramList.Add(new SQLiteParameter("@USstate", newMember.USstate));
            paramList.Add(new SQLiteParameter("@yearJoined", newMember.YearJoined));
            paramList.Add(new SQLiteParameter("@active", newMember.Active));
            paramList.Add(new SQLiteParameter("@amountOwed", newMember.AmountOwed));
            paramList.Add(new SQLiteParameter("@id", newMember.Id));

            return DButils.storeData(updateMemberQuery, paramList);
        }

        // Deletes a member from the database using the internal ID number
        public static bool deleteMember(int memberId)
        {
            string deleteMemberQuery = "DELETE FROM Membership WHERE id = @id";

            List<SQLiteParameter> paramList = new List<SQLiteParameter>();
            paramList.Add(new SQLiteParameter("@id", memberId));

            return DButils.storeData(deleteMemberQuery, paramList);
        }
    }
}
```

```
using System.Collections.Generic;
using System.Data.SQLite;

namespace FBLA.Utils
{
    // DButils class provides data manipulation methods as utility functions for use
throughout the application
    static class DButils
    {
        // the connection string to use for connecting to the FBLA Membership
database
        private const string connectionString = @"data source='FBLAmembership.db';
Version=3;";

        // this function executes the supplied command, along with parameters, and
returns the data as a DataAdapter object
        public static SQLiteDataAdapter getDBData(string cmdText,
List<SQLiteParameter> paramList)
        {
            var cmd = openDBConnection();
            cmd.CommandText = cmdText;

            // Add any parameters that have been supplied for executing this command
            if (paramList != null && paramList.Count > 0)
            {
                foreach (var param in paramList)
                {
                    cmd.Parameters.Add(param);
                }
            }
            SQLiteDataAdapter ad = new SQLiteDataAdapter(cmd);
            return ad;
        }

        // this function executes the supplied command, along with parameters, to
store data in the database
        public static bool storeData(string cmdText, List<SQLiteParameter>
paramList)
        {
            SQLiteCommand cmd = openDBConnection();
            cmd.CommandText = cmdText;

            if (paramList != null && paramList.Count > 0)
            {
                foreach (var param in paramList)
                {
                    cmd.Parameters.Add(param);
                }
```

```
            }

            // ExecuteNonQuery is for executing queries that update, insert, or
delete data
            int rowsAffected = cmd.ExecuteNonQuery();

            return (rowsAffected > 0);
        }

        // opens a connection to the database and returns the command object used to
execute queries
        private static SQLiteCommand openDBConnection()
        {
             var connection = new SQLiteConnection(connectionString);
             connection.Open();

             SQLiteCommand cmd = connection.CreateCommand();
             return cmd;
        }

    }
}
```

```
using System;
using System.Globalization;

namespace FBLA
{
    // Extensions class contains extension methods (for string and decimal
variables)
    public static class Extensions
    {

        // extension method - captilizes only the first letter in the string
        public static string ToTitleCase(this string str)
        {
            return CultureInfo.CurrentCulture.TextInfo.ToTitleCase(str.ToLower());
        }

        // extension method - converts number to a decimal rounded to two decimal
places
        public static decimal ToCurrency(this decimal num)
        {
            return decimal.Round(num, 2, MidpointRounding.AwayFromZero);
        }

    }
}
```

```
using System;
using System.Windows.Forms;

namespace FBLA
{
    // AboutForm class provides information about the application - identical to the
README document
    public partial class AboutForm : Form
    {
        public AboutForm()
        {
            InitializeComponent();
            CenterToParent();
        }

        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FBLA
{
    public partial class AboutAddEditMember : Form
    {
        public AboutAddEditMember()
        {
            InitializeComponent();
            CenterToParent();
        }

        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FBLA
{
    public partial class AboutCreateReport : Form
    {
        public AboutCreateReport()
        {
            InitializeComponent();
            CenterToParent();
        }

        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FBLA
{
    public partial class AboutGeneratedReport : Form
    {
        public AboutGeneratedReport()
        {
            InitializeComponent();
            CenterToParent();
        }

        private void buttonClose_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```