

APPLICATION REPORT

Report and analysis of the Finance Calculator created for
Assignment 1 of CO1109 Business and Financial Computing



Tom Cooper

Contents

Introduction	2
Language and IDE.....	2
Assumptions.....	2
Source Code and Design	3
Overview	3
Main	4
Description	4
Improvements.....	6
Form 1 – Main Menu	6
Description	6
Improvements.....	6
Form 2 – Value Calculator.....	7
Description	7
Improvements.....	8
Form 3 – Ratio Calculator.....	9
Description	9
Improvements.....	10
General Improvements	10
Testing.....	10
Conclusion.....	11
Appendix	11
Appendix 1 Source Code	11
Main Class	11
Form 1 – Main Menu	15
Form 2 – Value Calculator.....	16
Form 3 – Ratio Calculator.....	18
Appendix 2 Screen Captures	23
Form 1- Main menu	23
Form 2- Value Calculator	24
Form 3 – Ratio Calculator.....	25
Buttons and Messages	26

Introduction

I have been tasked with designing and creating a small application with an appropriate Graphical User Interface (GUI), where a user will have the ability to calculate simple financial formulas. The application has been designed in such a way that there are two main displays shown to the user. The first display manages 'Time Value of Money' (referred to as 'Value Calculator' in the application), where the two calculations that can be made are the present value and future value. The second display manages basic financial ratios (referred to as 'Ratio Calculator' in the application), where the current ratio, the working capital ratio, the debt-to-equity ratio, and the gross profit margin can be calculated.

This document starts by examining the early decisions made and any assumptions made in initial design. The document will then review the source code in detail, class by class, as well as improvements that were identified. Comments on testing methods and results will be made towards the end, as well as a short conclusion. [Appendix 1](#) and [2](#) contain the source code in full and some screen captures. In-document links may be found in this document for self-referral as well as some links to external internet URL's for referencing.

Language and IDE

The initial choice for the programming language and IDE was Java and IntelliJ. The reasons for this were subjective as practice was needed with this language. However, it was soon realised that it might take longer the time frame given to implement a successful program. This was mainly due to inexperience regarding GUI's with Java, despite the use of two Java frameworks designed to ease the programming of a GUI (JavaFX and Swing).

After this realisation, it was decided to use the Visual Basic alongside with the IDE Visual Studio. This was chosen as there was previous experience with them both. The main reason for this choice, however, was the ease of use of implemented a GUI. In Visual Studio, it is a simple case that controls, and elements of the GUI are dragged and dropped from a built-in toolbox. There is then automatic creating and handling of these elements regarding the code. There is also a properties menu for each element which makes the changes and customisation of the GUI far easier than searching and changing code. Having a visual design overview of the forms and display is also very useful to ensure that the GUI is presented in a way that is intended, without trial and error of writing code. Although, it should be noted there may be similar features with other IDE that are not known.

Another advantage using Visual Studio was the inherit compatibility with the Windows OS and using the .NET framework, however on test publishing errors were encountered, discussed further in [testing](#). A final advantage was the clear and expansive documentation for the Visual Basic language, as well as the support that Microsoft offers.

There was, however, some disadvantages using Visual Studio. The main issue I encountered when using Visual Studio was generally buggy behaviours. Occasionally application errors would be thrown, with seemingly no fix other than to close and restart the IDE with one such event causing hours of lost work. Visual Studio also seemed to be run slow, with edits on the GUI designer and debugging taking a few seconds to run or execute. These disadvantages could be associated with the workstation used to create the program; however, it is a powerful, new, and fully updated computer system which encounters no issues using other IDE's and software applications.

Assumptions

By far, the biggest assumption that is being made regarding this application is the device and operating system that the user will be using. It has been assumed that the user will be using a

Microsoft Windows OS (XP or later), and therefore will be on a desktop computer, laptop, or tablet running this OS. This assumption is somewhat justified, however, as it was outlined in the design brief that the application must run in Windows XP or later.

Whilst efforts have been made to provide functionality for exporting data, it is also being assumed that the user will not wish to save the data and the calculations made will be stand alone to any other application. Following this, it is also assumed that the user will not want to integrate the application into any others and the application will be used completely independently of any other.

Regarding the dimensions of the application display, it is being assumed that the user has a display of no less than 800 x 600 pixels as this is set size for each form.

With the input to the application, it is being assumed that the user has a keyboard and mouse to provide input in their computer system and therefore the application (although it can technically be operated with a keyboard alone).

A final assumption being made is that the user has the technical knowledge to be able to locate an .exe file in a directory and open it. This is because there is no “installation wizard” to run the application for them, although this could certainly be an [improvement](#) on later versions.

Source Code and Design

A copy of the source code can be found in [appendix 1](#), although some direct references and illustrations may be found in this section. I will give an overview of the application works then go into detail regarding each class and form. Improvement points specific to each form will be given, however many of the improvements can be found in the [general improvements](#) section.

Overview

The code, by design of VB/VS is event driven programming and the code is written in each form. Each control or element that is placed in the GUI must have code associated it when the control is used. As previously discussed, the form and control loading are handled by VS and not directly by the programmer. Therefore, the code that has been written is in four main sections to correspond with the 3 forms used and a Main class. The application is made up of each form/class (.vb file) and is combined with the .resx files (embedded resource).

The ability to view and display all the calculations is handled by the implementation of data tables which provide a data source for the corresponding data grid view control. It has been decided that any navigation between forms will result in the data table being wiped and a new data table created for the calculator being navigated to. This is to ensure robust programming and reduce bugs as issues in previous applications have arose from “left over” data in forms. With further knowledge there may better ways this can be achieved and [improved](#).

With the value calculator, it was relatively easy to contain both future and present value in the same data table and data view. This was achieved by having a column that indicates either P or V. However, the amount of different column header names that would be required for the ratio calculators, it was decided that each calculation should have its own data table and respective data view.

There is also functionality through the [export](#) subroutine to save data externally to the application.

On every form, the ESC key will call the Quitter subroutine, whilst ENTER will act as though the calculate button has been pressed.

The TAB key can also be used to cycle through select buttons and controls in the application.

Main

Description

This is the main class that handles functionality that is universal (mostly) through all other classes and forms. It begins by importing System.IO, this is essential for the creation of the data tables that are used by all calculators. The other import, ClosedXML.Excel, is for the exporting of data to a .xls file (Excel spreadsheet). "ClosedXML is a .NET library for reading, manipulating, and writing Excel 2007+ (.xlsx, .xlsm) files. It aims to provide an intuitive and user-friendly interface to dealing with the underlying OpenXML API" ([URL](#) to developer).

Variables are then declared, in this case being the 5 data tables used in the application to store (temporarily) the calculations performed.

The first subroutine found in the Main class is DtValueCreate. This creates the data table for both the value calculators. It also assigns the data table to the source of the data grid view control. This function is called every time a user navigates to the value calculator. The second function, DtRatioCreate, is similar. However, it creates each data table for the ratio calculators. For both functions, the data grids are forced hidden. This is to ensure that the user is presented with a clean GUI when entering the calculator display. The appropriate column headers will appear when the user selects which calculator they wish to use.

Next, there is a function to check if the user wished to navigate to a different form. It takes in the data grid and data table as parameters. This function, CheckMove, is called on every button that navigates the user away from their current form or calculator. This function is needed as the application is designed to wipe all data and input when navigating between calculators. The check occurs by a pop-up message box that gives the option to either choose "No" (Returning false) or "Yes" (Returning True).

TableWipe is a subroutine that iterates through every column in the passed in data table and removes it. It will also clear all rows and columns in the corresponding grid (passed in).

The Quitter function is called on every quit button on the application and quits the applications when pressed. Like other buttons, the function further calls CheckMove to check that user truly wishes to move. If the function returns True, the application closes, otherwise the subroutine is halted (user is returned to a state that is the same as before the button was pressed).

The MoveToRatioCalc subroutine is next and it is called when ever the user is moving to the ratio calculator form. Calling previously discussed functions, it will wipe all data from the value calculator, create a new data table for the ratio calculations, hide the current form (passed in as a parameter) and show the ratio calculator form. It will check, however if the user is on the main display (form 1). In this case, it will not call CheckMove, as there is no data that would be lost on navigation.

The MoveToValueCalc subroutine is functionally the same as MoveToRatioCalc but in this case it will wipe the ratio data and create a new value table.

Returner is a subroutine for when the user selects the return button. Functionally, it is like the Quitter function, however, will return the user to the main menu, rather than quitting the application.

Exporting data

ExportXLS is the subroutine which holds the majority of the development time. This was due to a trial and error of suitable libraries and implementation of those, until ClosedXML was found. Other libraires were either far too complex or required the user to have required files in the Windows Global Assembly Cache (namely a specific type of installation of Microsoft Office). Immediately, it should be noted that the subroutine name should be changed as it also deals with the exportation to a .txt file as well.

This subroutine starts by declaring the variables required. The `fileCounter` for keeping check of how many files have been exported. The `defaultName` to provide a name if the user does not write one. The `inputPath` that the file will be saved to, and a `checkNum` for the implementation of a select case.

It will first check if there is any data to export. This is achieved through checking the data table that has been passed in. If there are 0 rows of data in it, it will provide a message to the user that there is nothing to export. It will then end the subroutine.

Next, it will assign a value to the `checkNum` based on certain criteria established through If statements. If the value of `btnToggle` (function to switch between present and future), is either true or false it will assign a value to `checkNum`. The next 4 values establish which ratio table is being used by Boolean values set in [form 3](#).

A select case is then used to set the input path (and therefore creating of folder if needed) and default name to the corresponding calculation, previously established by the If statements and the `checkNum` provided.

The subroutine will then copy the data table passed into a new data table (`dt`), this to adhere to the instructions of the library being used, however it may not entirely be necessary.

A message box (with a prompt) is then provided to the user to write in a file name. If the user selects “cancel” or “ok” with the field blank, a previously established default file name will be used. The file counter is also incremented by 1, as well as the any potential input from the user being trimmed of whitespace.

The folder path is then created (if it is required). The folder path is determined by the select case and therefore the calculation being made. This means that there is a logical folder directory that the user can find all calculations made from the application with appropriate names.

The subroutine then checks if the export will be to a .xls file format or a .txt format. This is achieved, rather crudely, by a passed in value of either “.txt” or “XLS” which is determined depending on which button the user has selected.

If it is a .txt export, the sub routine will create a StreamWriter class. The parameters passed in is the file name (which will be the folder path, file name, and the extension), and False value indicating that it is not to use specified encoding and use the default buffer size. This part of the code was heavily influenced by the [Microsoft document page](#) and [Microsoft Forum post's](#). Whilst the general concept (each data row for the passed in data table is written to a new line and saved in the stated folder path) is understood, this section was a product of luck and a lot of trail and error to get working as intended- rather than a deep understanding how to use StreamWriter.

If the export is to be in .xls format, the subroutine will instead create a new excel workbook through the ClosedXML library function. It will then add a sheet to the workbook and save in the folder path previously established.

Once either export has occurred a message box will prompt the user to show that the export was successful.

Improvements

The first improvement in the class is within the ExportXLS subroutine, specifically with saving. As the application currently stands, it is not a very intelligent in the file saving process. For example, it will not challenge or warn the user if a file is about to be overwritten. It will also not give the opportunity for the user to select a file save path. Also, the application does not save the file counter variable. This means, for example, if a user saves a calculation with the default name then closes the application and restarts it and saves again in the same manner, the original file will be overwritten. A quick solution to this would be for the application to search for already existing filenames and counters and adjust, as necessary. In future versions, however, it may be worthwhile to implement a whole new save file system within the application to handle all the issues in a more efficient manner. This might also mean that data might not need wiped on every navigation change.

Following on with the export and save function, the message box provided for the user to fill in the file name is not fit for purpose. The “cancel” and “ok” button are functionally the same if the field is left blank or filled and the application will save the file regardless which is pressed.

Staying with the export subroutine, although functional, the solution for adapting the subroutine depending on which calculator is being used may be incorrect (or rather not the most efficient way). An attempt was made to use If statements alone, however the select case had to be introduced to ensure correct functionality. It may also be useful for the user to combine all calculations into one file, if desired.

It may also be useful to implement built in functionality to the application where data is always saved on the application exit and automatically recovered the next time the application is executed.

The functions to move to either the value or ratio calculator could be combined in some way, although a solution to this has not been found. The current subroutines instinctively do not feel like a generally correct solution to the problem of moving from one calculator to another, despite their correct functionality.

Form 1 – Main Menu

Description

[This](#) form is the main display the user will see when the application is loaded. It contains welcome text, a logo, and three buttons. The Value Calculator takes the user to the value calculator. The Ratio Calculator, to the ratio calculator and the Quit button closes the application.

Each of these buttons has a function (discussed in [Main](#)) that is held in the Main class. These are the MoveToValueCalc, MoveToRatioCalc, and Quitter.

Improvements

The first improvement that could be made with this form is the GUI design, it is somewhat bland. Specifically, the image and welcome text. However, if this application were to ever be sold this may

be an advantage as a company could change the logo and welcome text that suits their brand, and they would not be influenced by any designs by the programmer.

Functionally, other than a help button, there may not need to be any other improvements.

Form 2 – Value Calculator

Description

[This](#) form is deals with the value calculator, both present and future.

Firstly, all variables required are declared.

Line 6 – Raw input of number of years.

Line 7 – Raw input of amount.

Line 8 – Raw input of annual interest.

Line 9 – Boolean to show if fields are empty or not.

Line 10 - Boolean to show if input for number of years is correct.

Line 11 - Boolean to show if input for amount is correct.

Line 12 - Boolean to show if input for number of years is correct.

Line 13 – Integer that increments after every calculation. Starting at 1.

Line 14 – Boolean to show if user is making a present or value calculation.

Line 15 – Amount to be displayed in data grid.

Line 16 - Interest to be displayed in data grid.

Line 17 – String of “P” for use in populating and displaying data in grid.

Line 18 – String of “F” for use in populating and displaying data in grid.

Line 19 – Data table to be passed into the export subroutine.

```
6 Dim inputNumOfYears As Decimal
7 Dim inputAmount As Decimal
8 Dim inputAnnualInterest As Decimal
9 Dim validateEmpty As Boolean
10 Dim validateNumOfYears As Boolean
11 Dim validateAmount As Boolean
12 Dim validateInterest As Boolean
13 Dim entryNum As Integer = 1
14 Public btnToggle As Boolean = True
15 Dim displayAmount As String
16 Dim displayInterest As String
17 Dim pReturn As String = "P"
18 Dim fReturn As String = "F"
19 Dim toExport As DataTable
```

The next important subroutine is that of the button “Calculate”.

```
0 references
Public Sub Button1_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
```

This handles most of the processing in this form and it driven by the event of the user clicking the “Calculate” button.

First, it will check if any of the input fields are empty, if so, it will show a message to the user that the fields must be complete. If they are filled (for the moment with any values) it will turn the `validateEmpty` to `True`, indicating that the fields have been validated as not being empty. This is

important to reduce any unnecessarily code being ran if the user has not entered anything in the fields.

If, and only if, the fields have values, this subroutine will then attempt to parse (changing the field input values to a decimal format). If this is not successful for any reason (user has typed in anything other than a decimal number), a message will be displayed to the user that they must type in a decimal number. It will also provide specific examples of what they should be entering. For those values that it applies (e.g. amount) a separate display variable will take the input value and convert it to a currency value to be displayed. This is useful as it means the application could be global (to English speaking countries) as the currency value is a localised value based on Windows settings.

Next, if all conditions of validation (empty, data validation) have been satisfied, the subroutine will create a data row to be appended to the corresponding data table. It will display on the grid the entry number, the inputted data (formatted if necessary), the result of the calculation, and increment the entry number by one. All inputs and input values will be reset or “cleaned” through the `CleanInputs` subroutine.

The `CleanInputs` subroutine specifically resets all values of the input fields, the input values (those that have been parsed), and any formatted display variables. It is executed when the user presses the clear button.

A short `CheckPorF` Subroutine is also used in the creation of the data rows and will either return “P” or “F” based on the user selection. This is determined by `btnToggle` value.

Concerning the calculation of the result value, this is handled by the function `CalculatorValue`. It declares two variables. One is the calculated value, and one is the return value. First, it will check if `btnToggle` is `True` (user is calculating future value). If so, it will make the correct calculation and that will be assigned to the calculated value. If `False` (user is calculating present value), it will change the calculation to be made. The result will then be formatted to currency and returned to be added in the data row.

`btnToggle` is changed by the user clicking the “Change to x” where x can be either be present or future. Other values that change depending on the value of this value is the label indicating to the user what calculation they are making, the label above one of the input fields, and the text in the button itself. The [export](#) subroutine also makes use of this value.

Other buttons in this form include:

- Export to Txt – starts export subroutine, passing in txt.
- Export to XLS - starts export subroutine, passing in XLS.
- Return – returns to main menu.
- Quit – quits application.
- Ratio Calculation – navigates to the ratio calculator.

These buttons are simply using functions previous discussed in the [main](#) section.

Improvements

The `CheckPorF` function may not be necessary, although it does allow for the application to grow as it saves future programmers having to reuse code. Similarly, there may be a clearer way to show whether the user is calculating present or future value to the application (instead of using the `btnToggle`).

An overflow exception may be thrown for extremely large calculations regarding the future value of money. This is not caught in the programming and in the published version the application simply crashes with an error.

Form 3 – Ratio Calculator

Description

[This](#) form deals with the ratio calculator and the 4 different type of calculation.

The code for this calculator will be like form 2. The declared variables are similar. The difference is:

- Only two inputs
- 4 entry numbers instead of just the 1 for the value calculator
- 4 Boolean to represent which calculator is being used instead of the toggle button previously used.

```
3 Public ratioCurrent As Boolean = False
4 Public ratioDebt As Boolean = False
5 Public ratioProfit As Boolean = False
6 Public ratioCap As Boolean = False
7 Dim input1 As Decimal
8 Dim input2 As Decimal
9 Dim validateEmpty As Boolean
10 Dim validateInput1 As Boolean
11 Dim validateInput2 As Boolean
12 Dim entryNumCurrent As Integer = 1
13 Dim entryNumDebt As Integer = 1
14 Dim entryNumGross As Integer = 1
15 Dim entryNumCap As Integer = 1
16 Dim displayInput1 As String
17 Dim displayInput2 As String
18 Dim toExport As DataTable
```

There is a very similar implementation of the calculation button with the key difference being in the calculator. This time it is named `CalculatorRatio` and will calculate the formula based on the value of the variables in line 3 to 6. In this case, however, the current ratio, debt to equity, and the gross profit margin use the same calculation. Despite this the calculate function is written in such a way that each calculation still has its own line within the calculate function. This is so, if the need arises, a different formula can be used (such as a new type of calculation replaces others in future version).

The selection of the calculator that the user wants to use is also different. In this case, it is a combo box. It handled by:

```
93 Sub cmbSelector_SelectedIndexChanged(sender As Object, e As EventArgs) Handles cmbSelector.SelectedIndexChanged
```

It is executed when a selection is made on the combo box. Once a selection is made, the picture of the arrow guiding the user where to start is removed. Dependant on what selection is made, it will then change the value of the `ratioX` (see line 3 to 6) to `True` and the others to `False`, where `x` is the name of the calculator selected. It will also hide the other data grids relating to the other calculators and show the user the data grid that corresponds to what they have selected. Labels for the input fields and the main label to indicate to the user what calculator they are currently using.

There is also an `CleanInputs` subroutine and works in a very similar way to the one used in Form 2. It is also activated when the user clicks the clear button.

Other buttons include:

- Export to Txt – starts export subroutine, passing in txt.
- Export to XLS - starts export subroutine, passing in XLS.
- Return – returns to main menu.
- Quit – quits application.

- Value Calculation – navigates to the value calculator.

Improvements

The ratio calculator need not be as complex as it is. This was an error whilst writing the code. It was assumed each formula would be difference and complex, so it was written in such a way that it would be able to handle this. However, when it came to writing the calculator itself, only 2 different formulae were used.

Although not tried, another way to improve this calculator is to have all data grids shown in the same one. The user will then not have to switch between each one to see the calculations made.

General Improvements

The first general improvement would be how classes have been implemented, specifically how the Main class has been used to write all general functions and subroutines. Whilst functional, there may be far better solutions. For example, there could be a class of calculator that is created depending on what the user wishes to calculate. There then could be sub classes for each specific calculator. The data table and data grid views could then adapt to whatever sub class of calculator is being used.

A second general improvement would be for the export buttons to be a single button. The user would press this button then be prompted if they wished to export in either .txt or .XLS format. An attempt was made to implement this, however custom buttons in message boxes are above currently knowledge and experience to implement. Alternately, another form could be used. In similar vein to this, it might also be useful to give the user the option to export when leaving any form. This could be combined with the current solution of warning the user their data will be lost.

For all data grid views, the columns do not fill the space appropriately. Many fixes were tried to solve this; however, a correct solution was not found.

How inputs and general user errors are handled may be suboptimal. Instead, it may be far more useful to implement exception handling.

The return button that is used may be quite confusing to the user. They may think they are returning to the previous form, rather than returning to the main menu. Another improvement would be a help page. This could be accessible on every page and could either give information regarding the application overall or be related to wherever the user is in the application. Copies of the formulas used may also be shown.

Testing

This application has mainly been tested and debugged on a single computer system. This is the same computer system used to write the program and most of the testing has been continuous in the writing of the application. When a suitable version was ready, the application was published to be sent to two volunteer users. The first issue encountered was the publishing itself. Attempts were made to publish a single file, although whilst publishing was successful, when the executable was ran, nothing would happen. The solution was to remove the single file publishing. From the two testers it was also made apparent that the published application had to be “self-contained” with all the frameworks required.

The testers were instructed to try all functions of the application and try to “break it” and cause errors. They both have reported that they were unable to cause a crash except for the overflow exception mentioned in [form 2 improvements](#).

One of the testers also complained about form size and control size being out of proportion. For example, the grid data control was covering the whole form. Attempts were made to solve this by forcing properties on the controls, rather than allowing them to go to default values based on the user’s Windows OS settings.

Another issue is the certifications for anti-virus software. Both testers received multiple warnings that the application could be harmful and in one case, simply deleted it.

For future versions, testing will need to be a lot more expansive, with as many testers as possible attempting to use the application on different versions of Windows OS and settings. It will also be useful to create a testing table to ensure every possible variable and value is tested robustly. However, the brief for this application did not require it.

Conclusion

Whilst there are improvements that can be made to the application, the application is currently in a position where future versions could quite easily be created and improve on the application. There is scope for a professional UI designer to offer expertise, as well as improving the branding and the general professional feel of the application.

The greatest challenge for future versions will be in the publishing issues. An installation wizard would greatly improve the user experience, as well as a more functional save state system. Issues regarding certificates for the application or re-writing of the code in such a way that anti-virus software recognises the application as safe would also be necessary.

Appendix

Appendix 1 Source Code

Main Class

```
Imports System.IO
```

```
Imports ClosedXML.Excel
```

```
Public Class Main
```

```
    Public Shared interestTable As New DataTable
```

```
    Public Shared ratioTableCurrent As New DataTable
```

```
    Public Shared ratioTableDebt As New DataTable
```

```
    Public Shared ratioTableGross As New DataTable
```

```
    Public Shared ratioTableCap As New DataTable
```

```
    'Creates value datatable
```

```
    Public Shared Sub DtValueCreate()
```

```
        With interestTable
```

```
            .Columns.Add("Entry #", System.Type.GetType("System.String"))
```

```
            .Columns.Add("Number of Years", System.Type.GetType("System.String"))
```

```
            .Columns.Add("Amount", System.Type.GetType("System.String"))
```

```
            .Columns.Add("Annual Interest", System.Type.GetType("System.String"))
```

```
            .Columns.Add("P or F", System.Type.GetType("System.String"))
```

```
            .Columns.Add("Value", System.Type.GetType("System.String"))
```

```
        End With
```

```

        Form2.grid1.DataSource = interestTable
    End Sub

    'Creates ratio datatable ---- put into movement buttons
    Public Shared Sub DtRatioCreate()
        With ratioTableCurrent
            .Columns.Add("Entry #", System.Type.GetType("System.String"))
            .Columns.Add("Type", System.Type.GetType("System.String"))
            .Columns.Add("Total Current Assets", System.Type.GetType("System.String"))
            .Columns.Add("Total Current Liabilities",
System.Type.GetType("System.String"))
            .Columns.Add("Result", System.Type.GetType("System.String"))
        End With
        Form3.grid1.DataSource = ratioTableCurrent

        With ratioTableDebt
            .Columns.Add("Entry #", System.Type.GetType("System.String"))
            .Columns.Add("Type", System.Type.GetType("System.String"))
            .Columns.Add("Total Debt", System.Type.GetType("System.String"))
            .Columns.Add("Total Equity", System.Type.GetType("System.String"))
            .Columns.Add("Result", System.Type.GetType("System.String"))
        End With
        Form3.grid2.DataSource = ratioTableDebt

        With ratioTableGross
            .Columns.Add("Entry #", System.Type.GetType("System.String"))
            .Columns.Add("Type", System.Type.GetType("System.String"))
            .Columns.Add("Gross Profit", System.Type.GetType("System.String"))
            .Columns.Add("Revenue", System.Type.GetType("System.String"))
            .Columns.Add("Result", System.Type.GetType("System.String"))
        End With
        Form3.grid3.DataSource = ratioTableGross

        With ratioTableCap
            .Columns.Add("Entry #", System.Type.GetType("System.String"))
            .Columns.Add("Type", System.Type.GetType("System.String"))
            .Columns.Add("Current Assets", System.Type.GetType("System.String"))
            .Columns.Add("Current Liabilities", System.Type.GetType("System.String"))
            .Columns.Add("Result", System.Type.GetType("System.String"))
        End With
        Form3.grid4.DataSource = ratioTableCap

        Form3.grid1.Hide()
        Form3.grid2.Hide()
        Form3.grid3.Hide()
        Form3.grid4.Hide()
    End Sub

    'Confirms if the user truly wishes to move page, warns of data loss if move
    proceeds
    Public Shared Function CheckMove()
        Dim answer As Int32
        answer = MsgBox("Are you sure you want to leave?" & vbCrLf & "All unexported
data will be lost", vbCritical + MsgBoxStyle.YesNo + vbDefaultButton2, "Leaving
page!")
        If answer = vbYes Then
            Return True
        Else
            Return False
        End If
    End Function

```

```

End Function

'Table wipe when calc pages
Public Shared Sub TableWipe(grid, table)
    Dim i As Int16
    i = table.Columns.Count - 1
    While i >= 0
        table.Columns.RemoveAt(i)
        i -= 1
    End While
    table = Nothing
    grid.DataSource = Nothing
    grid.Rows.Clear()
    grid.Columns.Clear()
    grid.Refresh()
End Sub

' Sub for quit buttons
Public Shared Sub Quitter(form)
    If Not Main.CheckMove() Then
        Exit Sub
    End If
    form.Close()
    Application.Exit()
End Sub

' Sub for movement to Ratio Calc
Public Shared Sub MoveToRatioCalc(form)
    If form Is Form1 Then
        TableWipe(Form2.grid1, interestTable)
        DtRatioCreate()
        form.Hide()
        Form3.Show()
    ElseIf Not Main.CheckMove() Then
        Exit Sub
    Else
        TableWipe(Form2.grid1, interestTable)
        DtRatioCreate()
        form.Hide()
        Form3.Show()
    End If
End Sub

' Sub for movement to Value Calc
Public Shared Sub MoveToValueCalc(form)
    If form Is Form1 Then
        TableWipe(Form3.grid1, ratioTableCurrent)
        TableWipe(Form3.grid2, ratioTableDebt)
        TableWipe(Form3.grid3, ratioTableGross)
        DtValueCreate()
        form.Hide()
        Form2.Show()
    ElseIf Not Main.CheckMove() Then
        Exit Sub
    Else
        TableWipe(Form3.grid1, ratioTableCurrent)
        TableWipe(Form3.grid2, ratioTableDebt)
        TableWipe(Form3.grid3, ratioTableGross)
        DtValueCreate()
        form.Hide()
        Form2.Show()
    End If
End Sub

```

```

    End If
End Sub

' Sub for button when returning to main menu
Public Shared Sub Returner(form)
    If Not CheckMove() Then
        Exit Sub
    End If
    TableWipe(Form2.grid1, interestTable)
    TableWipe(Form3.grid1, ratioTableCurrent)
    TableWipe(Form3.grid2, ratioTableDebt)
    TableWipe(Form3.grid3, ratioTableGross)
    form.Close()
    Form1.Show()
End Sub

'Function to export data to an Excel fileformat.
Public Shared Sub ExportXLS(inputDataTable, type)
    Dim fileCounter As Int32 = 1
    Dim defaultName As String = ""
    Dim inputPath As String = ""
    Dim checkNum As Int16

    If inputDataTable.rows.Count <= 0 Then
        MsgBox("Nothing to export", vbExclamation, "Empty Table")
        Exit Sub
    End If

    If Form2.btnToggle Then
        checkNum = 0
    End If

    If Not Form2.btnToggle Then
        checkNum = 1
    End If

    If Form3.ratioCurrent Then
        checkNum = 2
    End If
    If Form3.ratioDebt Then
        checkNum = 3
    End If
    If Form3.ratioProfit Then
        checkNum = 4
    End If
    If Form3.ratioCap Then
        checkNum = 5
    End If

    Select Case checkNum
        Case 0
            defaultName = "FutureCalc"
            inputPath = "C:\FinanceCalc\TimeValue\"
        Case 1
            defaultName = "PresetCalc"
            inputPath = "C:\FinanceCalc\TimeValue\"
        Case 2
            defaultName = "CurrentRatioCalc"
            inputPath = "C:\FinanceCalc\RatioCalc\"
        Case 3
            defaultName = "DebtCalc"

```

```

        inputPath = "C:\FinanceCalc\RatioCalc\"
    Case 4
        defaultName = "GrossMarginCalc"
        inputPath = "C:\FinanceCalc\RatioCalc\"
    Case 5
        defaultName = "WorkingCap"
        inputPath = "C:\FinanceCalc\RatioCalc\"

End Select

' Takes input data table and assings to dt
Dim dt As New DataTable()
dt = inputDataTable

' Asks user from the desired filename, if left blank will create default name
of "calcuation" plus the number (to be incremented to prevent overwrite)
Dim fileName As String = InputBox("Enter filename or leave blank")
fileName.Trim()
If fileName = "" Then
    fileName = defaultName & fileCounter
    fileCounter += 1
End If

' Creates (if required) a folder in C:\FinanceCalc\ for file to save to.
Dim folderPath As String = inputPath
If Not Directory.Exists(folderPath) Then
    Directory.CreateDirectory(folderPath)
End If

If type Is "txt" Then
    ' Creates and saves txt file
    Dim txtfileName As String = folderPath + fileName + ".txt"
    Using sw As New StreamWriter(txtfileName, False)
        For Each drow As DataRow In inputDataTable.Rows
            Dim lineoftext = String.Join(", ",
drow.ItemArray.Select(Function(s) s.ToString).ToArray)
            sw.WriteLine(lineoftext)
        Next
    End Using
    MsgBox("File Saved Successfully in " & folderPath, vbInformation,
"Sucessful")
End If

If type Is "XLS" Then
    ' Creates a new XL workbook using CloseXML library, add dt to it and saves
it at previously determined dir and filename. Alerts user of sucess and file location
    Using wb As New XLWorkbook()
        wb.Worksheets.Add(dt, "Sheet")
        wb.SaveAs(folderPath & Convert.ToString(fileName & ".xlsx"))
    End Using
    MsgBox("File Saved Successfully in " & folderPath, vbInformation,
"Sucessful")
End If
End Sub
End Class

```

Form 1 – Main Menu

```
Public Class Form1
```

```
    Public Sub btnValue_Click(sender As Object, e As EventArgs) Handles btnValue.Click
```



```

        Main.MoveToValueCalc(Me)
    End Sub

    Private Sub btnRatio_Click(sender As Object, e As EventArgs) Handles
btnRatio.Click
        Main.MoveToRatioCalc(Me)
    End Sub

    Private Sub btnQuit_Click(sender As Object, e As EventArgs) Handles btnQuit.Click
        Main.Quitter(Me)
    End Sub
End Class

```

Form 2 – Value Calculator

```

Public Class Form2

    Dim inputNumOfYears As Decimal
    Dim inputAmount As Decimal
    Dim inputAnnualInterest As Decimal
    Dim validateEmpty As Boolean
    Dim validateNumOfYears As Boolean
    Dim validateAmount As Boolean
    Dim validateInterest As Boolean
    Dim entryNum As Integer = 1
    Public btnToggle As Boolean = True
    Dim displayAmount As String
    Dim displayInterest As String
    Dim pReturn As String = "P"
    Dim fReturn As String = "F"
    Dim toExport As DataTable

    'Function to calculate time money value
    Private Function CalculatorValue(years, amount, interest)
        Dim calcvalue As Decimal
        Dim toReturn As String
        If btnToggle Then
            calcvalue = amount * ((1 + (interest)) ^ years)
        Else
            calcvalue = amount / ((1 + (interest)) ^ years)
        End If
        toReturn = FormatCurrency(calcvalue)
        Return toReturn
    End Function

    'Wipes input fields
    Private Sub CleanInputs()
        boxAmount.Text = ""
        boxInterest.Text = ""
        boxNumOfYear.Text = ""
        inputNumOfYears = Nothing
        inputAmount = Nothing
        inputAnnualInterest = Nothing
        displayAmount = ""
        displayInterest = ""
    End Sub

    Public Function CheckPorF()
        If btnToggle Then
            Return fReturn
        Else

```

```

        Return pReturn
    End If
End Function

Public Sub Button1_Click(sender As Object, e As EventArgs) Handles btnCalc.Click

    If String.IsNullOrEmpty(boxAmount.Text) Or
String.IsNullOrEmpty(boxInterest.Text) Or String.IsNullOrEmpty(boxNumOfYear.Text) Then
        MsgBox("All fields must be complete to make a calculation", vbExclamation,
"Fields not full")
        Exit Sub
    Else
        validateEmpty = True
    End If

    If validateEmpty Then
        If Decimal.TryParse(boxNumOfYear.Text, inputNumOfYears) Then
            validateNumOfYears = True
        Else
            MsgBox("The number of years must a decimal number" & vbCrLf & vbCrLf &
"Example: 2 or 0.5", vbExclamation, "Entered incorrectly")
            Exit Sub
        End If

        If Decimal.TryParse(boxAmount.Text, inputAmount) Then
            displayAmount = FormatCurrency(inputAmount)
            validateAmount = True
        Else
            MsgBox("The amount must a decimal number." & vbCrLf & vbCrLf &
"Example: 1546 or 124.214", vbExclamation, "Entered incorrectly")
            Exit Sub
        End If

        If Decimal.TryParse(boxInterest.Text, inputAnnualInterest) Then
            inputAnnualInterest /= 100
            displayInterest = FormatPercent(inputAnnualInterest)
            validateInterest = True
        Else
            MsgBox("The interest must a decimal number, with no %" & vbCrLf &
vbCrLf & "Example: 1 or 0.5", vbExclamation, "Entered incorrectly")
            Exit Sub
        End If
    End If

    If validateEmpty And validateNumOfYears And validateAmount And
validateInterest Then

        Dim newrow As DataRow = Main.interestTable.NewRow
        newrow("Entry #") = entryNum
        newrow("Number of Years") = inputNumOfYears
        newrow("Amount") = displayAmount
        newrow("Annual Interest") = displayInterest
        newrow("P or F") = CheckPorF()
        newrow("Value") = CalculatorValue(inputNumOfYears, inputAmount,
inputAnnualInterest)
        entryNum += 1
        Main.interestTable.Rows.Add(newrow)
        CleanInputs()
        newrow = Nothing
    End If
End Sub

```

```

        Else
            Exit Sub

        End If

    End Sub

    Private Sub btnQuit_Click(sender As Object, e As EventArgs) Handles btnQuit.Click
        Main.Quitter(Me)
    End Sub

    Private Sub btnRtn_Click(sender As Object, e As EventArgs) Handles btnRtn.Click
        Main.Returner(Me)
    End Sub

    Private Sub btnWipe_Click(sender As Object, e As EventArgs) Handles btnWipe.Click
        CleanInputs()
    End Sub
    Private Sub btnChange_Click(sender As Object, e As EventArgs) Handles
btnChange.Click

        If btnToggle Then
            lblIndicator.Text = "Present"
            btnChange.Text = "Change to future"
            lblAmount.Text = "Enter the future value"
            btnToggle = False

        ElseIf Not btnToggle Then
            lblIndicator.Text = "Future"
            btnChange.Text = "Change to Present"
            lblAmount.Text = "Enter the current amount"
            btnToggle = True
        End If

    End Sub

    Private Sub btnExport_Click(sender As Object, e As EventArgs) Handles
btnExportTxt.Click
        toExport = Main.interestTable
        Main.ExportXLS(toExport, "txt")
    End Sub
    Private Sub btnExportXLS_Click(sender As Object, e As EventArgs) Handles
btnExportXLS.Click
        toExport = Main.interestTable
        Main.ExportXLS(toExport, "XLS")
    End Sub

    Private Sub btnRatio_Click(sender As Object, e As EventArgs) Handles
btnRatio.Click
        Main.MovetoRatioCalc(Me)
    End Sub

End Class

```

Form 3 – Ratio Calculator

```

Public Class Form3

    Public ratioCurrent As Boolean = False
    Public ratioDebt As Boolean = False

```

```

Public ratioProfit As Boolean = False
Public ratioCap As Boolean = False
Dim input1 As Decimal
Dim input2 As Decimal
Dim validateEmpty As Boolean
Dim validateInput1 As Boolean
Dim validateInput2 As Boolean
Dim entryNumCurrent As Integer = 1
Dim entryNumDebt As Integer = 1
Dim entryNumGross As Integer = 1
Dim entryNumCap As Integer = 1
Dim displayInput1 As String
Dim displayInput2 As String
Dim toExport As DataTable

Private Sub CleanInputs()
    boxInput1.Text = ""
    boxInput2.Text = ""
    input1 = Nothing
    input2 = Nothing
End Sub

Private Function CalculatorRatio(input1, input2)
    Dim calcvalue As Decimal
    Dim toReturn As String

    If ratioCurrent Then
        calcvalue = input1 / input2
    ElseIf ratioDebt Then
        calcvalue = input1 / input2
    ElseIf ratioProfit Then
        calcvalue = input1 / input2
    ElseIf ratioCap Then
        calcvalue = input1 - input2
    End If

    If ratioCap Then
        toReturn = FormatCurrency(calcvalue)
    Else
        toReturn = Math.Round(calcvalue, 5)
    End If

    Return toReturn
End Function

Private Sub btnExportTxt_Click(sender As Object, e As EventArgs) Handles
    btnExportTxt.Click

    If ratioCurrent Then
        toExport = Main.ratioTableCurrent
    ElseIf ratioDebt Then
        toExport = Main.ratioTableDebt
    ElseIf ratioProfit Then
        toExport = Main.ratioTableGross
    End If

    Main.ExportXLS(toExport, "txt")
End Sub

```

```

Private Sub btnExportXLS_Click(sender As Object, e As EventArgs) Handles
btnExportXLS.Click

    If ratioCurrent Then
        toExport = Main.ratioTableCurrent
    ElseIf ratioDebt Then
        toExport = Main.ratioTableDebt
    ElseIf ratioProfit Then
        toExport = Main.ratioTableGross
    ElseIf ratioCap Then
        toExport = Main.ratioTableCap
    End If

    Main.ExportXLS(toExport, "XLS")
End Sub

Private Sub btnQuit_Click(sender As Object, e As EventArgs) Handles btnQuit.Click
    Main.Quitter(Me)
End Sub

Private Sub btnValue_Click(sender As Object, e As EventArgs) Handles
btnVaule.Click
    Main.MoveToValueCalc(Me)
End Sub

Private Sub btnRtn_Click(sender As Object, e As EventArgs) Handles btnRtn.Click
    Main.Returner(Me)
End Sub

Private Sub cmbSelector_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cmbSelector.SelectedIndexChanged
    picArrow.Visible = False
    If cmbSelector.SelectedIndex = 0 Then
        ratioCurrent = True
        ratioDebt = False
        ratioProfit = False
        lblIndicator.Text = "Current Ratio"
        lblInput1.Text = "Total current assets"
        lblInput2.Text = "Total current liabilities"
        grid1.Show()
        grid2.Hide()
        grid3.Hide()
        grid4.Hide()

    ElseIf cmbSelector.SelectedIndex = 1 Then
        ratioCurrent = False
        ratioDebt = True
        ratioProfit = False
        lblIndicator.Text = "Debt To Equity"
        lblInput1.Text = "Total debt"
        lblInput2.Text = "Total equity"
        grid1.Hide()
        grid2.Show()
        grid3.Hide()
        grid4.Hide()

    ElseIf cmbSelector.SelectedIndex = 2 Then
        ratioCurrent = False
        ratioDebt = False
        ratioProfit = True
        lblIndicator.Text = "Gross Profit Margin"

```

```

        lblInput1.Text = "Gross Profit"
        lblInput2.Text = "Revenue"
        grid1.Hide()
        grid2.Hide()
        grid3.Show()
        grid4.Hide()

    ElseIf cmbSelector.SelectedIndex = 3 Then
        ratioCurrent = False
        ratioDebt = False
        ratioProfit = False
        ratioCap = True
        lblIndicator.Text = "Working Capital"
        lblInput1.Text = "Current Assets"
        lblInput2.Text = "Current Liabilities"
        grid1.Hide()
        grid2.Hide()
        grid3.Hide()
        grid4.Show()

    End If

End Sub

Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click

    If String.IsNullOrEmpty(boxInput1.Text) Or
String.IsNullOrEmpty(boxInput2.Text) Then
        MsgBox("All fields must be complete to make a calculation", vbExclamation,
"Fields not full")
        Exit Sub
    Else
        validateEmpty = True
    End If

    If validateEmpty Then
        If Decimal.TryParse(boxInput1.Text, input1) Then
            validateInput1 = True
            displayInput1 = FormatCurrency(input1)

        Else
            MsgBox("The amount must a decimal number." & vbCrLf & vbCrLf &
"Example: 1546 or 124.214", vbExclamation, "Entered incorrectly")
            Exit Sub
        End If

        If Decimal.TryParse(boxInput2.Text, input2) Then
            validateInput2 = True
            displayInput2 = FormatCurrency(input2)
        Else
            MsgBox("The amount must a decimal number." & vbCrLf & vbCrLf &
"Example: 1546 or 124.214")
            Exit Sub
        End If

    End If

    If validateEmpty And validateInput1 And validateInput2 Then

```

```

If ratioCurrent Then

    Dim newrow As DataRow = Main.ratioTableCurrent.NewRow
    newrow("Entry #") = entryNumCurrent
    newrow("Type") = "Current Ratio"
    newrow("Total Current Assets") = displayInput1
    newrow("Total Current Liabilities") = displayInput2
    newrow("Result") = CalculatorRatio(input1, input2)

    entryNumCurrent += 1
    Main.ratioTableCurrent.Rows.Add(newrow)
    CleanInputs()
    newrow = Nothing

ElseIf ratioDebt Then

    Dim newrow As DataRow = Main.ratioTableDebt.NewRow
    newrow("Entry #") = entryNumDebt
    newrow("Type") = "Debt To Equity"
    newrow("Total Debt") = displayInput1
    newrow("Total Equity") = displayInput2
    newrow("Result") = CalculatorRatio(input1, input2)

    entryNumDebt += 1
    Main.ratioTableDebt.Rows.Add(newrow)
    CleanInputs()
    newrow = Nothing

ElseIf ratioProfit Then

    Dim newrow As DataRow = Main.ratioTableGross.NewRow
    newrow("Entry #") = entryNumGross
    newrow("Type") = "Gross Profit Margin"
    newrow("Gross Profit") = displayInput1
    newrow("Revenue") = displayInput2
    newrow("Result") = CalculatorRatio(input1, input2)

    entryNumDebt += 1
    Main.ratioTableGross.Rows.Add(newrow)
    CleanInputs()
    newrow = Nothing

ElseIf ratioCap Then

    Dim newrow As DataRow = Main.ratioTableCap.NewRow
    newrow("Entry #") = entryNumGross
    newrow("Type") = "Working Capital Ratio"
    newrow("Current Assets") = displayInput1
    newrow("Current Liabilities") = displayInput2
    newrow("Result") = CalculatorRatio(input1, input2)

    entryNumCap += 1
    Main.ratioTableCap.Rows.Add(newrow)
    CleanInputs()
    newrow = Nothing

End If
Else
    Exit Sub
End If

```

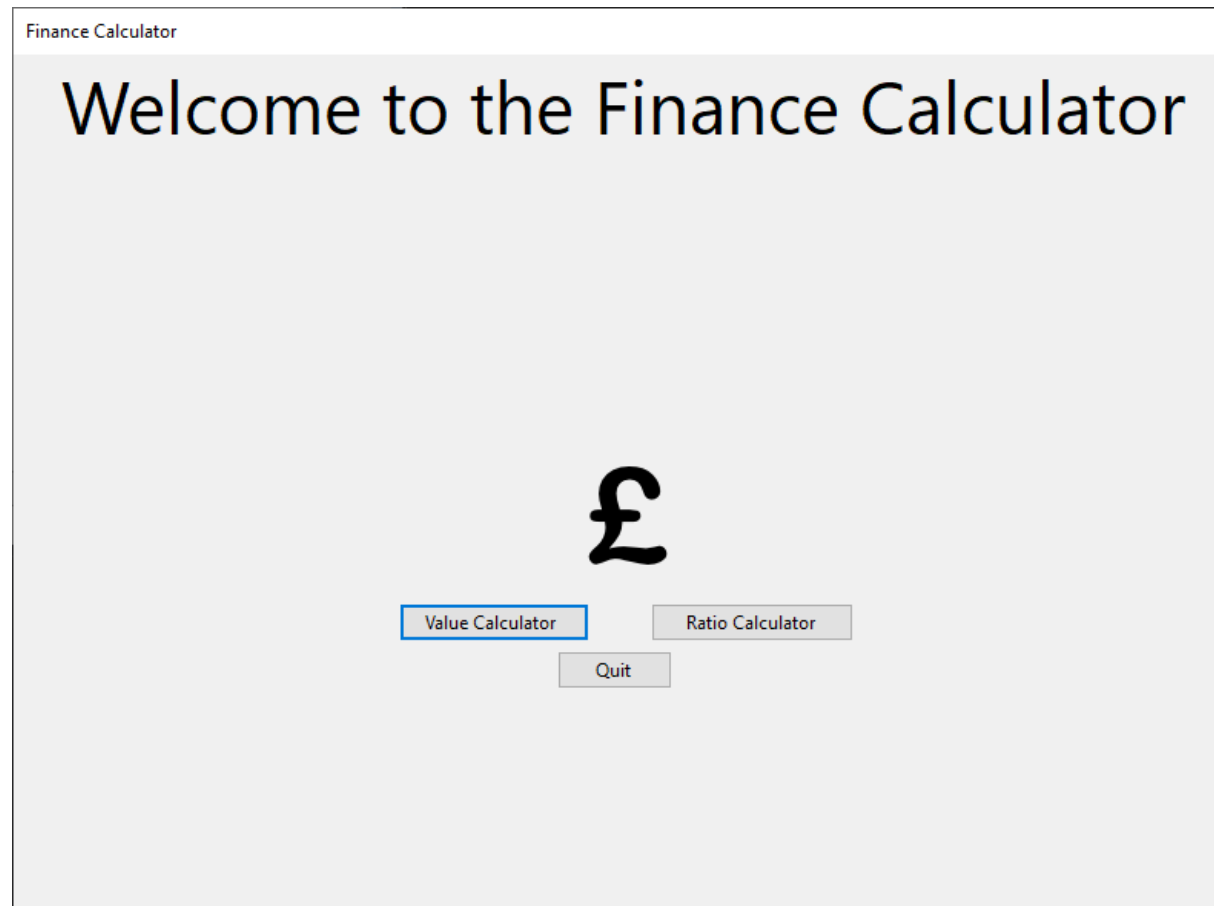
```
End Sub
```

```
Private Sub btnWipe_Click(sender As Object, e As EventArgs) Handles btnWipe.Click  
    CleanInputs()  
End Sub
```

```
End Class
```

Appendix 2 Screen Captures

Form 1- Main menu



Form 2- Value Calculator

Finance Calculator

Present

Enter period of years

Enter the future value

Enter the annual interest

Change to future

Calculate

Clear

	Entry #	Number of Years	Amount	Annual Interest	P or F	Value
▶	1	1	£100.00	2.00%	F	£102.00
	2	2	£200.00	1.00%	F	£204.02
	3	5	£500.00	1.00%	P	£475.73

Return

Ratio Calculator

Export to Txt


Export to XLS

Quit

Ratio Calc

Current Ratio

Select type:

Current ratio 

Total current assets

Total current liabilities

Calculate

Clear

	Entry #	Type	Total Current Assets	Total Current Liabilities	Result
▶	1	Current Ratio	£200,000.00	£12,000.00	16.66667
	2	Current Ratio	£35,411,110.00	£255,541.26	138.57297

Return

Vaule Calculator

Export to Txt

Export to XLS

Quit

Ratio Calc

Gross Profit Margin

Select type:
 Gross profit margin ▾

Gross Profit
 Revenue

Calculate
 Clear

	Entry #	Type	Gross Profit	Revenue	Result
▶	1	Gross Profit Margin	£2,555,441.00	£12,220.00	209.11956
	1	Gross Profit Margin	£998,547.00	£655,547.00	1.52323

Return
 Vaule Calculator
 Export to Txt
 Export to XLS
 Quit

Buttons and Messages

Gross Profit
 Revenue

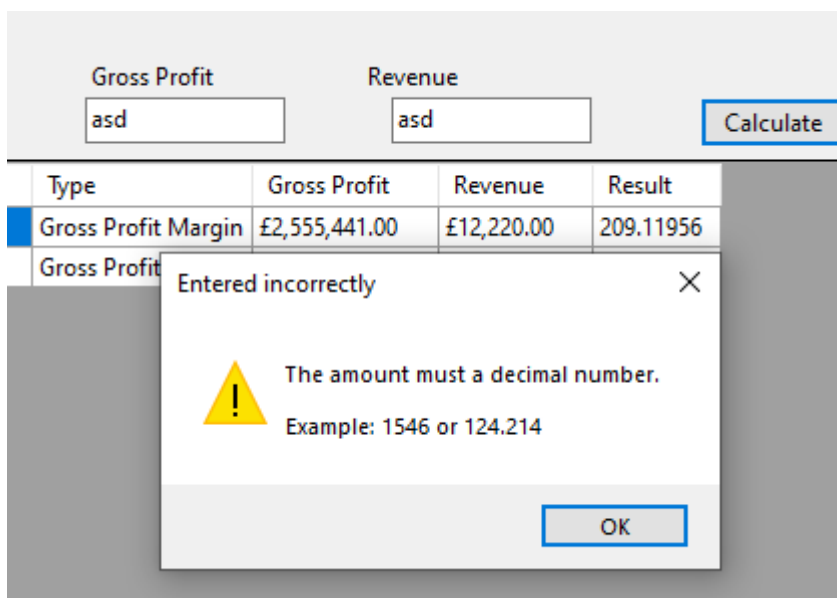
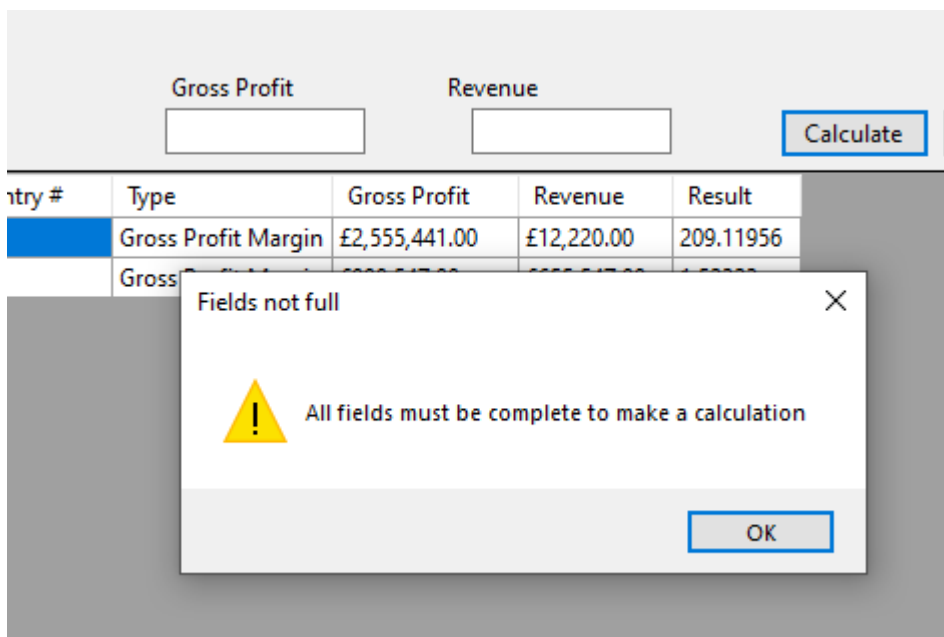
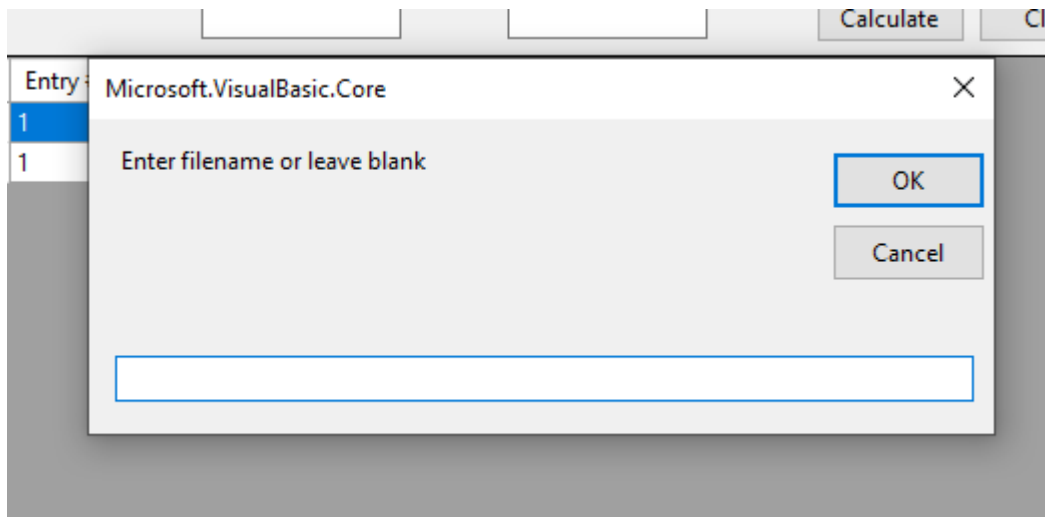
Calculate
 Clear

Entry #	Type	Gross Profit	Revenue	Result
	Gross Profit Margin	£2,555,441.00	£12,220.00	209.11956
	Gross Profit Margin	£998,547.00	£655,547.00	1.52323

Leaving page!

Are you sure you want to leave?
 All unexported data will be lost

Yes
 No



Working Capital

Select
Worki

Current Assets

Current Liabilities

Calculate

Clear

Entry #	Type	Current Assets	Current Liabilities	Result
---------	------	----------------	---------------------	--------

Empty Table



Nothing to export

OK