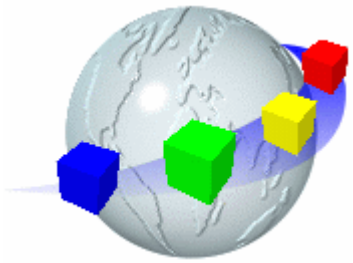
WebCab Portfolio for .NET v5.0

WebCab
Components



Preface

This documentation accompanies the WebCab Portfolio .NET Service. The purpose of this documentation is to provide a clear and concise description of all aspects that are likely to be encountered in real life applications by developers and users of this .NET Service. WebCab Portfolio contains methods which use the Markowitz and Capital Asset Pricing Model to construct a portfolio which for a given level of risk offers the greatest expected return.

The first chapter of this documentation contains a brief introduction to the most important implemented features and related system requirements. In chapter two we let the developer quickly get started with deploying the component by detailing deployment techniques. The third chapter contains the mathematical documentation, which represents the theoretical background of this component's implemented business functionality. Chapters four to seven contains the programmer's guide containing a road map for developers to take advantage of every feature and capability of this product for use with Microsoft's Office, Visual Studio 6, Borland's C++ Builder and the .NET Framework respectively. In chapter eight we provide some examples to illustrate how features detailed within the programmer's guides can be applied in practice. Finally, we introduce WebCab Components, its philosophy and approach to serving the .NET development community with robust and powerful enterprise applications.

Good luck with your project and thank you for your interest in our components.

The WebCab Components Team

Contents

Chapter 1

Introduction

1.1 Product Description

1.1.1 Overview

Apply Markowitz Theory and Capital Asset Pricing Model (CAPM) to analyze and construct the optimal portfolio with/without asset weight constraints with respect to Markowitz Theory by giving the risk, return or investors utility function; or with respect to CAPM by given the risk, return or Market Portfolio weighting. Also includes Performance Evaluation, extensive auxiliary classes/methods including equation solve and interpolation procedures, analysis of Efficient Frontier, Market Portfolio and CML.

1.1.2 Details

This suite includes the following features:

- **Markowitz Model** - Construct optimally diversified portfolios.
  - **Efficient Frontier** - Construct the Efficient Frontier with or without constraints on the asset weights.
  - **Utility Function** - Discover and set the investors utility function.
  - **Optimal Portfolio** - Select the optimal portfolio or set of portfolios by providing the expected return desired, the maximum risk or the investors utility function.
- **Capital Asset Pricing Model (CAPM)** - Construct optimally diversified portfolios with can hold or borrow cash.
  - **Efficient Frontier** - Construct the Efficient Frontier with or without constraints on the asset weights.
  - **Market Portfolio** - Find the Market Portfolio which offer the greater expected return per unit of risk.

  - **Capital Market Line (CML)** - Construct the CML with contains the optimal portfolio with respect to the CAPM.
  - **Selecting Optimal Portfolio** - Select the optimal portfolio by given expected return, risk or the Market Portfolio weighting.
  - **Analysis of Optimal Portfolio** - Evaluate the risk, expected return or Market Portfolio weighting of the optimal portfolio whenever one of these three properties is known.

- Auxiliary Classes
  - **Interpolation** - Cubic spline and general polynomial interpolation procedures to assist in the study and manipulation of curves such as the Efficient Frontier which are evaluated at a finite number of points.
  - **SolveFrontier** - Solve the Efficient Frontier with respect to the risk, return, or the investors utility function which may be given as a function of the risk or the expected return.
  - **TwoAssetPortfolio** - Evaluate of the optimal weighting of a portfolio with two assets. This functionality can be used to analyze the effect of a single purchase or sale from an arbitrary portfolio
  - **AssetParameters** - Evaluation of the covariance matrix, expected return, volatility, portfolio risk/variance, ARCH model for expected price.
  - **MaxRange** - Evaluates the maximum range of the values of the expected return for which Efficient Frontier should be considered when the historical data set does is not consistent within the assumptions of Markowitz Theory and CAPM.
  - **Performance Evaluation** - Offers a number of procedures for accessing the return and risk adjusted return (Treynors Measure, Sharpes Ratio).

This product also has the following technology aspects:

- **3-in-1: .NET, COM, and XML Web services** - Three DLLs, Three API Docs, Three Sets of Client Examples all in 1 product. Offering a 1st class .NET, COM, and XML Web service product implementation.
- **Extensive Client Examples** - Multiple client examples including .NET (C#, VB.NET, C++.NET), COM and XML Web services (C#, VB.NET)
- **ADO Mediator** - The ADO Mediator assists the .NET developer in writing DBMS enabled applications by transparently combining the financial and mathematical functionality of our .NET components with the ADO.NET Database Connectivity model.
- **Compatible Containers** - Visual Studio 6 (incl. Visual Basic 6, Visual C++ 6), Visual Studio .NET (incl. Visual Basic .NET, Visual C#.NET, and Visual C++.NET), Borland's C++ Builder (incl. C++Builder, C++BuilderX, C++ 2005), Borland Delphi 3 - 2005, Office 97/2000/XP/2003.

- **ASP.NET Web Application Examples** - We provide an ASP.NET Web Application example which enables you to quickly test the functionality within this .NET Service.
- **ASP.NET Examples with Synthetic ADO.NET** - we use a ASP.NET service to perform component calculations on SQL database columns from a remote DBMS. We apply a component's function to certain rows from the database and list the output in HTML format. This is a powerful feature since it allows you to perform calculations in a DBMS manner without having to code the C# to SQL database transaction yourself as it is all done by the ASP within the .NET Framework managed server side environment.

1.2 Package Details

This .NET Service package contains the following:

- Introductory Text File (README.TXT)
- License Agreement
- Documentation in PDF Format
  - Product Description
  - System Requirements
  - Compatibility Issues
  - Deployment Guide (How to get started?)
  - Mathematical Documentation
  - Programmer's Guide
  - Examples
  - Guide to WebCab Components
- Class Documentation
  - Class Descriptions
  - Methods Descriptions
- Deployment Files (`DLLs`)
- Examples and Related Source Code Files
- WebCab Components Brochure

1.3 Prerequisites and Compatibility

1.3.1 System Requirements

- Microsoft Windows® XP/2000/2003 family
- Pentium III 500 MHz
- 64MB RAM
- .NET Framework 1.0 (or higher)

1.3.2 Compatibility

Component Type

- ASP.NET XML Web service
- .NET Class Library
- COM Component

Built Using

- Microsoft .NET Framework SDK

Chapter 2

Where do I Start?

Start using the Portfolio for .NET v5.0 .NET Service right away by following the few quick and simple steps described in this chapter. If you require additional information or have problems with the installation and use of this .NET Service then please do not hesitate to contact us via our support forum at: http://www.webcabcomponents.com/support/index.php

2.1 What Type of User Are You?

.NET Components or XML Web Services

The prerequisite Windows operating system components you will need to have installed on your local machine will depend on the way in which you intend to use our .NET Service. In particular, there are two distinct deployment architectures in which our product can be used:

- **Class Library** - those wishing the use our .NET Components functionality directly within there .NET Applications. This category includes those who wish to use our component within Microsoft's Visual Studio .NET or Borland's C#Builder projects, or those who simply want to register the DLL class library onto there local machine for consumption by local or remote applications.
- **XML Web service/ASP.NET** - those wishing to deploy either an ASP.NET based Client server application and an XML Web service.

Chapter Overview

Within the remainder of this chapter we will detail how to install and configure the necessary Windows components, deploy the .NET Components or XML Web Services and finally how to run examples associated within these two deployment scenarios[1].

[1]Later within this PDF documentation within the Programmers Guide we will cover development techniques which can be employed for either composing larger applications or building clients for these deployed components and Web services

The remainder of this chapter is structured in the following way, please feel free to skip any sections which are not relevant to you:

1. **Configuration** - Describes how to configure the Windows deployment/development platform for .NET based applications, including Web Services.
2. **Deployment** - Details the deployment of the .NET Service as a .NET Component or as a XML Web Service.
3. **Using the DLLs inside an IDE** - We detail here how to import the .NET Class Libraries DLLs into Microsoft's Visual Studio .NET and Borland's C# Builder.
4. **Clients** - Explains client examples provided with this .NET Service
5. **Live Demos** - Details the functionality of the Online Web Application Demos.

2.2 What Do I Need to Install and Where Can I Get It?

In order to deploy and then use a .NET Component within your applications you are required to have (or install) the following Windows component onto your Windows operating system:

- .NET Framework

If you wish to deploy either ASP.NET or XML Web services then you will also be required to install:

- Microsoft's Internet Information Server (IIS)

In order to develop .NET based applications (i.e. compile C#.NET source code) you will also need to have the following Windows Components installed:

- .NET Framework SDK

Getting the .NET Platform

The .NET Framework's installation package comes in the following two flavors:

- .NET Framework SDK (approx. 110MB)
- .NET Framework, Redistribution package (approx. 23MB)

Those wishing to develop applications using the .NET Framework will require the .NET Framework and the .NET Framework SDK installed onto the development machine. Whereas those who only wishing to deploy .NET Applications and Services will require only the .NET Framework (also known as the .NET Framework Redistribution package). The latest version of these packages can be downloaded from http://msdn.microsoft.com/downloads.

In order to run our ASP.NET and Web service demos you are also required to have Microsoft's IIS Web server installed and started.

Installing the .NET Framework

Through the **Add/Remove Programs** feature of the Control Panel you are able to discover which version (if any) of the .NET Framework you are currently running. If you intend to upgrade to a later version of the .NET Framework then you are presently running then we strongly suggest that you first uninstall all previous versions of the .NET Framework first. This is due to the fact that some applications will use the earliest installed version of the .NET Framework rather than the latest edition.

You can then proceed to install the latest version of the .NET Framework by just executing the .NET Framework installation package by following through the dialog within the installation program.

Remark Please note that Microsoft will ensure backward compatibility of the .NET Framework. That is, application developed under v1.0 for example will run and behave in a similar fashion under later releases of the .NET Framework.

2.2.1 Installing the IIS Web server

In order to deploy and run ASP.NET or XML Web services you are required to have the relevant pieces of the "Applications Server" stack installed onto the deployment Windows machine. The key piece of this infrastructure stack is Microsoft's IIS Web server.

On any Windows machine you can check to see whether the IIS Web server is installed and running by opening the following page within a web browser http://localhost. If the IIS Web server is installed and running then you will be presented with either a placeholder page similar to the following:

If you already have a web site deployed into the root directory of the IIS Web server then the home page of this web site will be displayed.

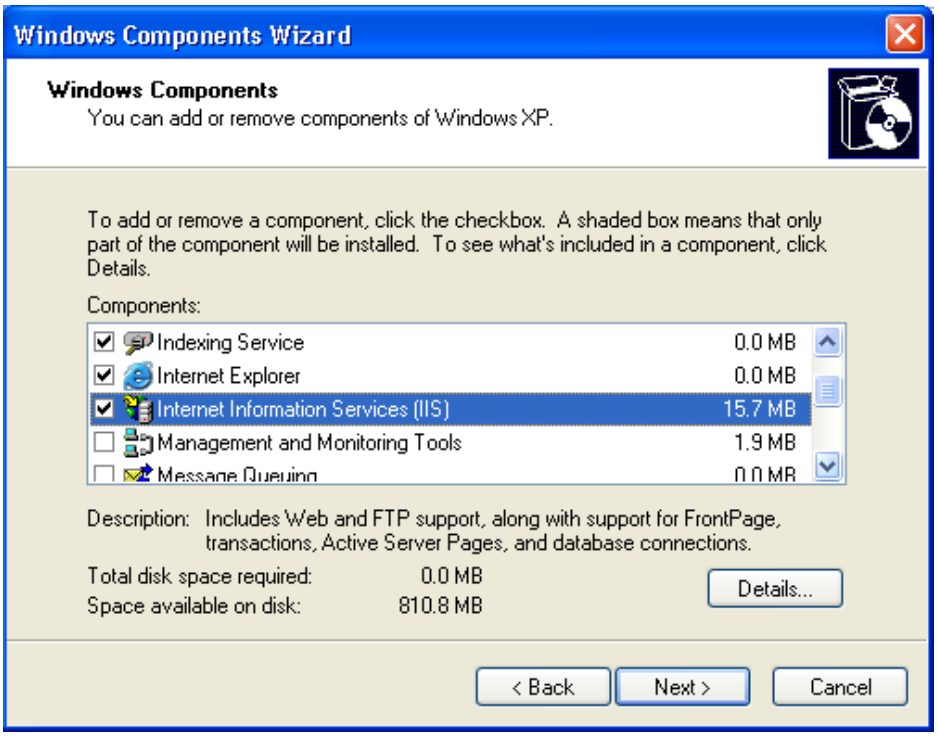
Remark By requesting the page http://localhost, the local IIS Web server will be automatically restarted if it is not already running.

Installation Process

If the IIS Web server is not installed onto your local machine then you may install it through using the Windows Program installation tool. To access this tool select:

Settings > Control Panel > Add or Remove Programs

Now click on the button, **Add\Remove Windows Components**. A window should pop-up in which a number of Windows components are listed, one of these items should read **Internet Information Services (IIS)**. You should select the click box for this service and install it by clicking the **Next** button.



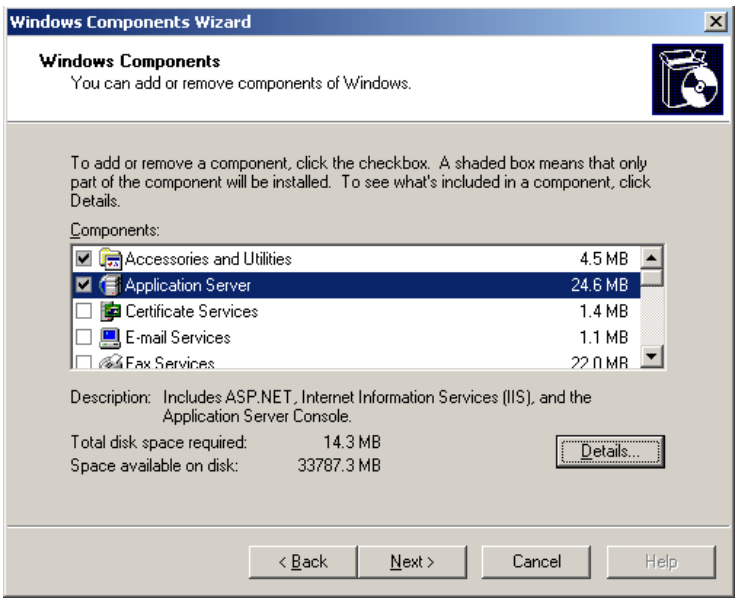
2.2.2 What Do I Need if I am Using Windows 2003?

Microsoft's Windows 2003 is the first Windows operating system that has been designed explicitly to host .NET Applications. Though, Windows 2003 has the .NET Framework v1.1 embedded within the operating system you will still need to configure the IIS Web server or the .NET Framework SDK if you intend to either deploy ASP.NET based Applications or use the Windows 2003 machine to develop .NET based Applications and Services.

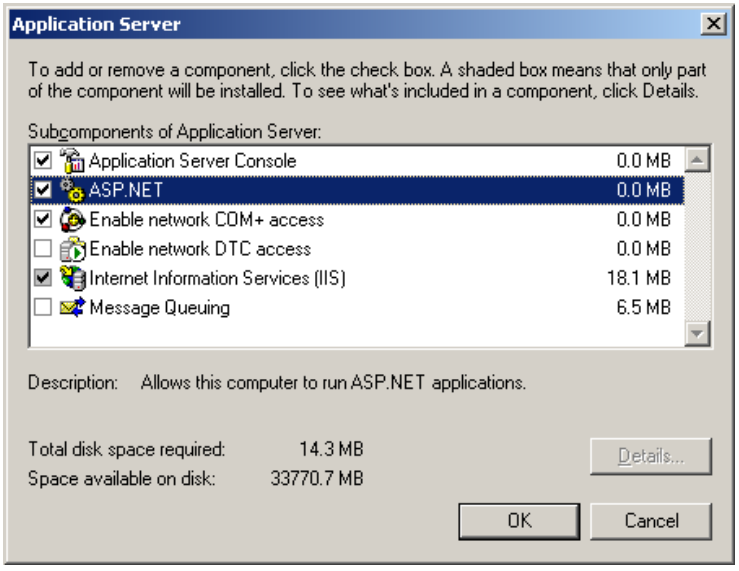
Installing the IIS Web server onto Windows 2003

The IIS Web server for the Windows 2003 operating system is not installed by default. Therefore, if you did not explicitly request the deployment of the system component known as 'Application Server' during the installation of Windows 2003. You will need to manually install this Windows 2003, component if you wish to deploy ASP.NET or XML Web services to your local machine. For all Windows platforms the deployment process of the IIS Web server proceeds in a similar fashion via the control panel interface.

In order to deploy the 'Applications Server' components first select **Add/Remove Windows Components** from the **Add/Remove Programs** control panel interface.



Once the 'Application Server' is selected by clicking on the **Details** button you will be presented with:



Remarks

- As you can see from the above screen shot by default the **ASP.NET**, **Enable network COM+ access** and **Internet Information Services (IIS)** have been selected which is sufficient for the deployment of nearly all .NET Framework based Applications and Services.
- The Windows 2003 platform contains a thoroughly updated infrastructure stack concerning the delivery of ASP.NET and XML Web services. This includes a new ASP.NET container which is closely integrated with the IIS Web server. This update contains several enterprise level .NET related features concerning the management of services (automatic startup, shutdown of services and the server itself), security (lockdown of ports and unused services), scalability (load balancing) and easier management (Component services and IIS management console). In our opinion the Windows 2003 platform offers a significantly more robust platform than early versions of the Windows platform in which to deploy .NET Applications and Services.

2.3 Deploying the .NET Service

If you installed this product using the MSI installer then the .NET Class Libraries DLLs should already be registered within your global assembly cache. In order to use the corresponding XML Web Service implementation you will need to follow the deployment guide below.

In case you installed using a Zip package or wish to install the .NET Class Libraries onto another Windows machine we explain below how to manually deploy our .NET Class Libraries.

2.3.1 Deploying a Component

Once the .NET Framework has been installed onto the deployment machine the component (i.e. the DLL), can be deployed and registered. By registering the component it will become available to local and remote .NET Applications and Services.

Making your components (locally) available

The easiest and quickest means to deploy the DLL component files, is to simply copy them to one of the following two directories on your Windows machine:

- **WINDOWS\System**
- **Application directory** - the directory in which the .NET Application which will consume the component is located.

When your Application is executed Windows will automatically look in one of these locations for the DLL file with it wishes to import.

Making the Components Globally available?

The above deployment technique is straightforward but has the restriction that is does not allow for any custom configuration of the deployment environment which may be required by the .NET Application which will consume the component. Moreover, such deployment techniques do not allow the components (i.e. DLL's) to be effectively shared between two or more applications. In order to allow the components to be used by several applications you must deploy the components within the **Global Assembly Cache**.

Remark The **Global Assembly Cache** within the .NET Framework plays a similar role to that of the CLASSPATH environment variable within the Sun Java platform.

A Component can be deployed to the **Global Assembly Cache** in one of two ways:

- Microsoft Windows Installer 2.0 - Recommended for use on production servers.
- Global Assembly Cache tool (Gacutil.exe) - Recommended only for use on development or test servers.

Please note, in order to use the Assembly Cache tool you are required to have the .NET Framework SDK installed and to create a Windows installer you will need the Windows Installer SDK v2.0. For further details concerning the Windows Installer technology we refer the interested reader to the Visual Studio .NET documentation or the MSDN section of Microsoft's website.

The Global Assembly Cache Tool is run from the DOS command prompt. In order to deploy the assembly WebCab.Libraries.PortfolioDemo.dll, contained within the present directory you must input the command:

```
Gacutil.exe /i WebCab.Libraries.PortfolioDemo.dll
```

Using the DLL within your own Applications

Within the **Library** directory of the installation package you will find the class library assembly (DLL) for Portfolio. There are several ways you may choose to use this DLL within your applications.

2.3.2 Deploying an XML Web Service

Once the .NET Framework and the IIS related infrastructure is installed the actual deployment of an ASP.NET Application or an XML Web service just involves copying a DLL file and several ASMX files into given directories within the IIS Web server root file directory.

Installation Process

Within the XML Web service folder within this installation package are the resources for deploying the Web service of the Portfolio. In order to install and start using the XML Web service functionality you will need to perform the following steps:

- Deploy the XML Web service assembly (DLL) onto your local IIS Web server. This is accomplished by copying the contents of the 'bin' folder of the current directory to the 'bin' folder of the local IIS root directory, which under the default installation is C:\Inetpub\wwwroot.
- Deploy all **.asmx** pages inside the IIS Web server, by copying all the folders located under the **XML Web service** folder located under the current directory to the IIS root directory.
- Run the WSDL tool in order to generate all XML Web service proxies. This is done by running the following line at the command prompt:

  ```
  wsdl http://localhost/<directory name>/<XML Web service>.asmx
  ```

  where <directory name> is the name of the subdirectory of the IIS root in which the XML Web service is deployed and <XML Web service> is the name of the Web service which you wish to generate a proxy for.

Remarks

- All generated proxies are used as described in the CHM documentation of this product.
- If the Web service is not being locally run then the 'localhost' should be replaced with the host name of the computer where the Web service is deployed and running.
- You may be using another equivalent ASP.NET Web server, in which case the above deployment procedure may differ slightly in detail but will still involve the same basic steps.

Testing the Deployment

You can easily test an XML Web service deployment by using any compatible web browser. That is, to test the deployment open a compatible web browser (for example Internet Explorer) and open the following page:

```
http://localhost/<directory name>/<filename>.asmx
```

where <directory name> is the name of the subdirectory of the IIS directory root where the Web services are located and <filename> is the name of one of the **.asmx** files located inside the <directory name> folder. As soon as the browser displays the page correctly you are sure that the corresponding XML Web service has been properly installed.

2.4 Using the DLLs inside an IDE

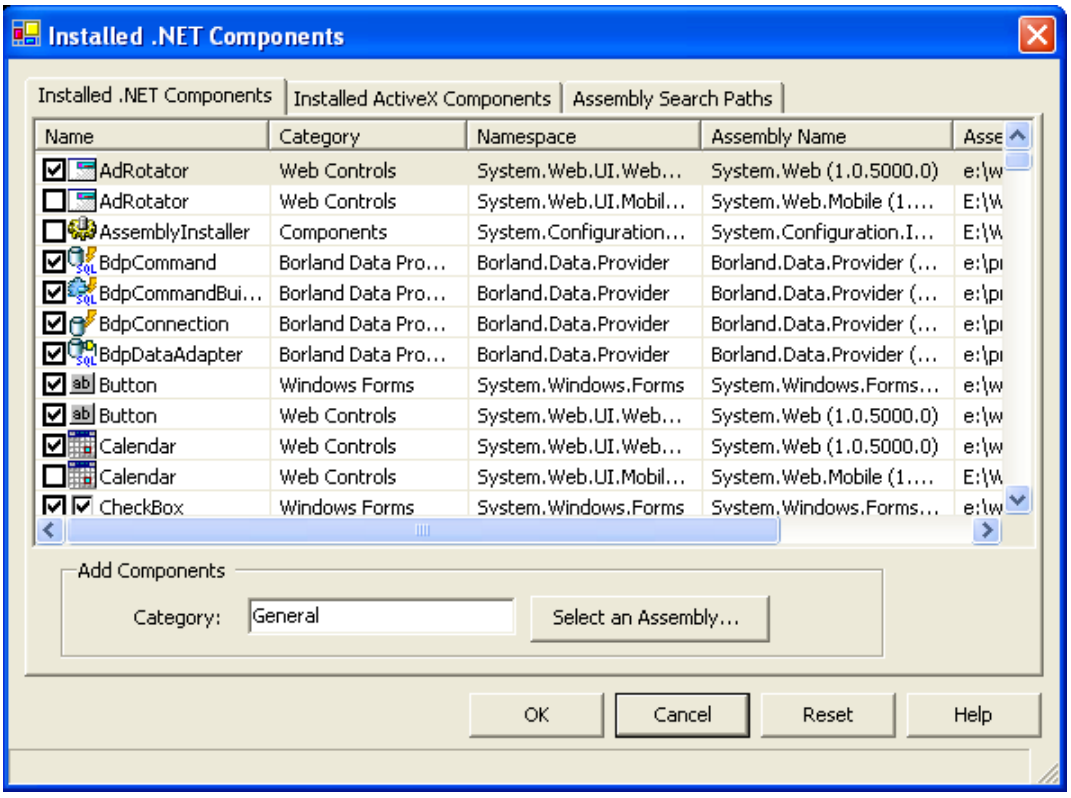
2.4.1 Using the DLL within a Visual Studio .NET project

If you are developing a Visual Studio .NET project and wish to use our DLLs components functionality then you may add the DLL to the project reference. This is performed by using the **Project** > **Add Reference...** menu item.

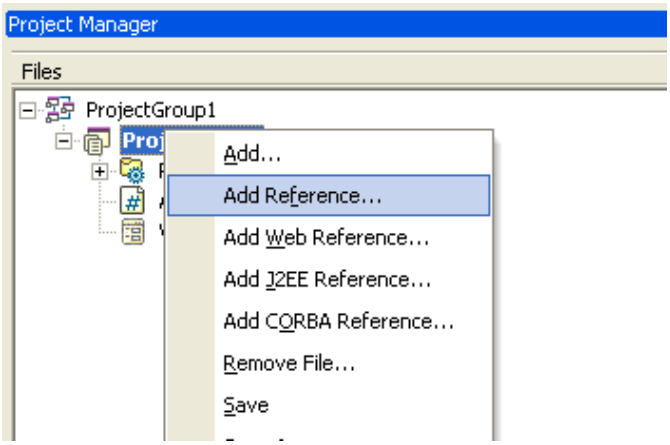
2.4.2 Using the DLL within a Borland's C# Builder project

If you are developing a C#Builder project and wish to use our DLLs components functionality then you may deploy and then add the assembly to your project solely from within C# Builder.

To deploy the DLL select from the menu, Component > Installed .NET Components Then click on the 'Select an Assembly', to select the DLL from the file system.



Once the DLL component is deployed you may add the DLL to your C# project by right clicking on your project within the "Project Manager" and selecting "Add reference" as show below:



Alternatively, you could use the equivalent link with the menu, Project > Add Reference.

Now an 'Add reference' window will pop-up, in which you should select the assembly you wish to add to your project and then click OK.

2.5 Client Examples

Within this section we describe the clients which have been provided with this .NET Service. In particular, we will describe:

1. **Console Client Examples**
2. **ASP.NET Client Examples**
3. **Web Service Client Examples**

2.5.1 Running the Console Client Example

In order to run the console demonstration examples you must start a DOS Command Prompt. On a Windows XP machine this can be achieved by:

START > Programs > Accessories > Command Prompt

Once the command prompt is started you should navigate to the **Librares\Client** subdirectory of the installation directory[2] using the cd command. The **Client** subdirectory is structured into subdirectories corresponding to client examples. Every client example should be accompanied by a **compile.cmd** file, which when run will compile its corresponding client. After compilation, the only thing left to do is launch into execution the generated executable file(s).

For further technical details you should read the **README.TXT** files located inside the **Client** subdirectories.

[2]Usually `C:\Program Files\WebCab Components\Portfolio`.

2.5.2 ASP.NET Client Example

Within the **\NET Libraries\ASP.NET Examples**, folder of this package we have provided a set of tailor made ASP.NET examples for the WebCab Portfolio for .NET v5.0 .NET Service.

In order to start using these ASP.NET examples, please go through the following steps:

1. **Deploy the ASP.NET Pages and Resources**

   (a) Copy the contents of the 'Portfolio ASP.NET Examples' folder located in the current directory to a location under your IIS (or another compatible ASP.NET web server) root directory. This is usually 'C:\Inetpub\wwwroot' or a similar path with a different drive letter.

   (b) Secondly, copy the contents of the 'bin' subfolder of the current directory to the 'bin' subfolder of your IIS root directory. If the 'bin' directory does not exist inside the IIS root directory, you will be required to create it.

2. **Run the ASP.NET Examples**

   (a) Open an Internet browser[3] and type in the following address:

   ```
   http://localhost/Portfolio ASP.NET Examples/index.html
   ```

   In case you chose to copy the 'Portfolio ASP.NET Examples' directory to a subdirectory of your IIS root directory, you will need to include the full path to it inside the above URL. Also, if you have deployed these ASP.NET examples to another .NET machine, you should replace 'localhost' with the name of that machine.

Remark An online version of these ASP.NET examples can be found at:

webcabcomponents.com/dotNET

Using the Example

Further explanation regarding the use of this example can be found within the 'Accessing the Online Demo', section of this guide which details the use of the online version of this ASP.NET example.

2.5.3 XML Web service examples

Within the directory **\XML Web Services\Client**, you will find C# application client examples which make use of the financial and mathematical functionality provided by the XML Web services implementation of the WebCab Portfolio .NET Service.

[3] E.g. Internet Explorer, Opera, Mozilla.

Remark In order to run and access the Web service examples it is necessary to have the IIS web server and a compatible internet browser (for example Internet Explorer 5 or higher) installed onto your local machine.

Please go through the following steps in order to test these client examples:

1. Before running or even compiling these XML Web service client examples you will have to deploy all Portfolio XML Web services located one directory level above to your local IIS server or an IIS server your machine can connect to. This is accomplished by copying the contents of the 'bin' folder, located one level above in the directory structure, to the 'bin' folder of the IIS root directory, usually `C:\Inetpubwwwroot`. You will also need to copy the 'PortfolioDemo' folder to the IIS root directory.

2. Browse through each subfolder of the current directory and locate every '`compile.cmd`' compilation script file. There is one compilation script for every client example. Every client example requires certain XML Web services to connect to, which you may determine by running the compilation script. The reported errors should correspond to the XML Web services the client example requires.

3. Run the WSDL tool in order to generate XML Web service proxies for every XML Web service required by each client example. This is done by running the following line at command prompt:

   ```
   wsdl http://localhost/PortfolioDemo/<XMLWebservice>.asmx
   ```

   where <XMLWebservice> is the name of the required XML Web service and '`localhost`' should is the host name of the computer where the IIS server is running, in case it's not this machine.

   Make sure you are running this command from directory containing the compilation script.

4. Run the '`compile.cmd`' compilation script file again. If the right XML Web service proxies have been generated for its client examples, the compilation script will compile all source code files and generate a .NET executable file (.exe). If the script still results in an error, please go through the previous step again, making sure you generate the XML Web service proxy classes required by the client example corresponding to this compilation script.

5. Run the generated executable files. Running the executable files will allow you to test the client examples.

You may compile the client examples (after having generated the XML Web service proxies) and run their executable files as you go through each subfolder, since every client example is independent of each other.

If you wish to customize a particular client example after having run it, please feel free to adapt its source code files accordingly and run the compilation script again. Note that you will not need to regenerate the XML Web service proxy classes.

2.6 Testing the Component

2.6.1 Accessing the Online ASP.NET Demo

By far the easiest way to test the functionality of this .NET Service is to view the online demo. The online demo can be accessed from our .NET Homepage by clicking the '[ASP.NET]' link corresponding to this Application. Once you click on the link the following pop-up Window will appear:

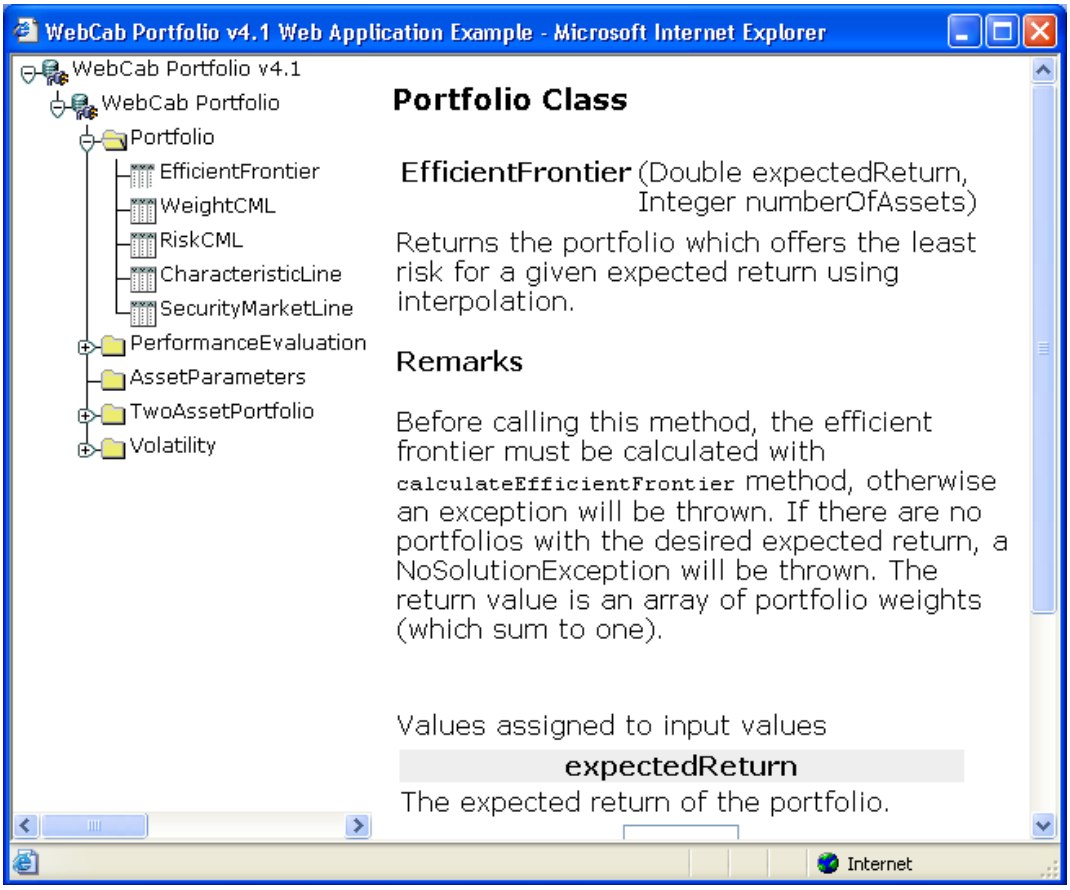


Fig: Interface of the ASP.NET online Web Application

Compatible Web Containers

This demo runs inside an online web container and demonstrates the ASP.NET technology. The demo can be accessed through a web browser and is compatible with Internet Explorer 5, Netscape Navigator 4, Netscape 6, Opera 5 and higher.

Selecting a method to test

After clicking on the '[ASP.NET]' link for this application from our home page the Web demo will launch within a new browser window. To order to select a method from this application use the drop-down menu on the left hand side of the screen to navigate through all implemented functions. The menu lists all the components on the first level and their corresponding functions on the second and third level. Click on the plus icon in order to expand the component menu and then click a function item to select it.

Inputting data

The selected function will be displayed on the right hand panel of the new window accompanied by its description and parameter characterization. Once the nature of the method is clear you may test the method by inputting the parameters within the text boxes provided or associating a database table field using our database management tool.

Running the demo using text boxes

Please input the value of each parameters in to the corresponding text box while paying attention to each parameter description. When all the parameters has been input, press the "Get Result" button on the right-hand side and towards the bottom of the screen in order to request the solution from the server-side component. If by chance any parameters are out of range you will be prompted to enter a correct value before proceeding.

2.6.2 Online XML Web service examples

The XML Web Services contained within this package have also been deployed online. The Web service demo is accessed through our .NET Homepage by clicking on the '[WSDL]' link corresponding to this Service. Once you click on the link the following pop-up Window will appear:

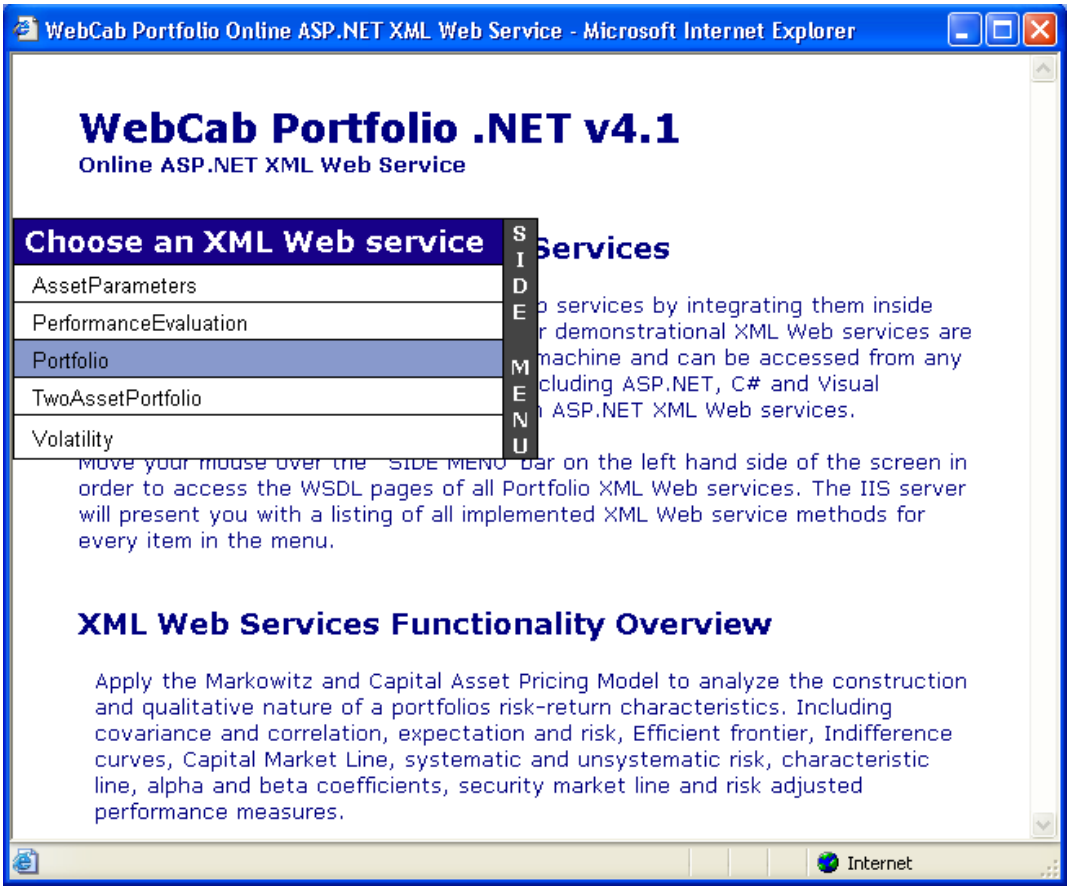


Fig: Interface of the Online Web Services

Remark In order to view the online demos you will require a compatible internet browser (for example Internet Explorer 5 or higher) with JavaScript enabled.

2.6.3 Using .NET Web Service Studio

You may wish to test the functionality of our XML Web services by using one of Microsoft's .NET tools made for testing XML Web services, *.NET Web Service Studio 2.0*. A link to where you can directly download this tool is available on our online web site at webcabcomponents.com/dotNET. Download and unzip the pack and run the `WebServiceStudio\build.bat` file. In order to start up the tool double-click the generated `WebServiceStudio.exe` .NET executable file.

Deciding on an XML Web service

Click here in order to open the .NET Homepage of our web site. At the bottom of the page you will notice a list of *Online .NET Web Services* with their corresponding ASP.NET Example page and their WSDL description page links. Click on the *WSDL* link next to :fullnamewithoutdotnet:. Move your mouse pointer above the left hand *SIDE MENU* and click on an XML Web service name.

A new browser window will open and present you with the default IIS 6.0 ASP.NET XML Web service page. Inside the 'Address' bar of your browser you will be able to see the fully qualified URL to the online WSDL for the corresponding XML Web service.

Connecting to an XML Web service

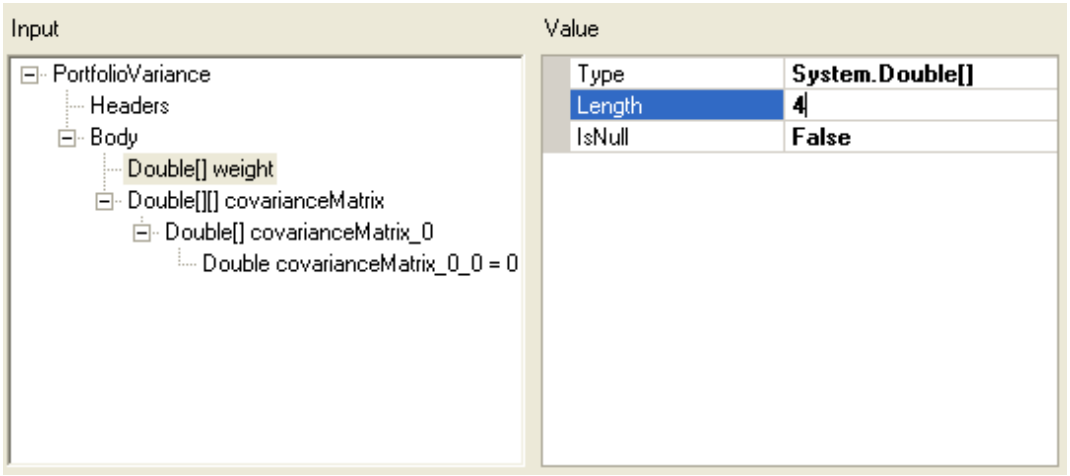
Copy from your browser's Address bar the WSDL URL to the XML Web service you have chosen to test and paste it into the `WSDL EndPoint` text field inside WebService Studio. Click the `Get` button next to this text field and wait for the tool to establish a connection to the XML Web service.

As soon as the connection has been established, you can start going through every available webmethod our XML Web service has to offer by clicking on the items in the left hand side of the tool's window. After clicking a web method's name, look inside the *Input* panel for required parameters. You may recognize these parameters from the available API CHM documentation we have included inside this package[4].

Sending in Web Method Parameters

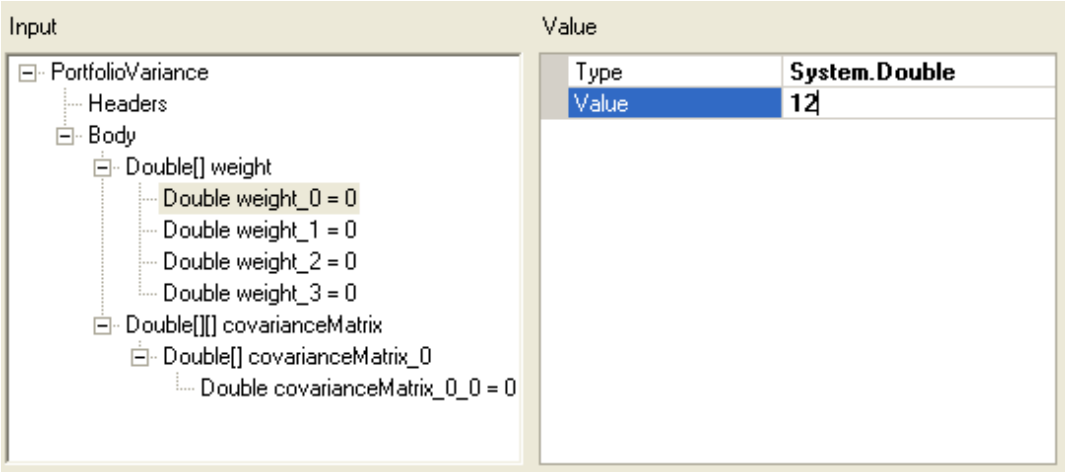
Inside the *Body* tree of the *Input* panel you may click on every parameter type and then set its value inside the *Value* panel to its right. This is accomplished by editing the right side of the *Value* column in the right hand side *Value* panel.

If you are dealing with parameters of an array type (say a `Double[]` array), you will first set its *Length* in the *Value* panel and then go through every of its tree children inside the *Input* panel and set their *Value* field back into the *Value* panel. The following two screen shots explain this process.



WebService Studio: Setting the Length of a Double Array to 4

[4]Browse the API Reference by double-clicking the CHM file located inside the `Documentation` directory of this Package.



WebService Studio: Setting the Value of its First Element to 12

Invoking a Web Method

After having sent in the right parameter values[5] click the `Invoke` button right below the *Value* panel on the right hand side of the window.

The *Output* panel below the *Input* panel will present you with the results of invoking the corresponding Web Method of our online XML Web service over the Internet.

[5]You should refer to the corresponding API HTML Help documentation even when running this test tool. You could however send in some arbitrary values just to try out some of our simpler Web Methods.

Chapter 3

Mathematical Documentation

Within this .NET Service we offer a through and flexible implementation of a number of ideas mainly concerned with the selection and analysis of the optimal portfolio in accordance with the assumptions of Portfolio Theory. Portfolio Theory, in general, deals with the interrelation between risk[1] and return[2] for Portfolios constructed from a given collection of assets. In particular, within our component we allow the user to apply Markowitz Theory and the Capital Asset Pricing Model (CAPM). Within this chapter of the PDF documentation we describe the theoretical content and application of these theories.

3.1 Assumptions, Questions, Functionality and Problems

The construction of Markowitz Theory and the Capital Asset Pricing Model (CAPM) is based upon a number of assumptions concerning the investment market, the behavior of the participants in these markets and the assets from which the investor's portfolio can be constructed.

3.1.1 Assumptions of Markowitz Theory

The assumptions underlying Markowitz Theory from which the model in derived is as follows:

1. Investors seek to maximize the expected return of total wealth.
2. All investors have the same expected single period investment horizon.
3. All investors are risk-adverse, that is they will only accept greater risk if they are compensated with a higher expected return.

[1] Risk is also know as volatility; it is a measure that determines the degree to which an asset's return (or sometimes price) fluctuates.

[2] The return of an asset is the absolute or relative (i.e. percentage) change of the price of an asset.

4. Investors base their investment decisions on the expected return and risk (i.e. the standard deviation of an assets historical returns).

5. All markets are perfectly efficient (e.g. no taxes and no transaction costs).

3.1.2 Assumptions of Capital Asset Pricing Model (CAPM)

The CAPM is an extension of the Markowitz Theory and hence requires all of its assumptions. The addition to these assumptions it also requires that the investor is able to lend or borrow (risk free) cash from the market in order to leverage or un-leverage a portfolio. In particular, we require the following provisions:

1. **Lend excess capital at the Market Rate:** The investor may lend money at the prevailing market rate. This constitutes the ability to hold within the portfolio a risk free asset which will provide the prevailing return available on cash. In practice, such assets are often referred to as a money market accounts and will yield a return in the region of LIBOR.

2. **Borrow capital at the Market Rate:** The investor may borrow money from the market at the prevailing market rate in order to invest within (risky) assets. This will increase the expected return of the original capital base and also increase its risk. In practice, the rate at which money can be lent from the market by a fund will be in the region of LIBOR (at least if the fund structures the loan as a REPO type agreement).

Remark The rate of which money can be borrowed or lend from or to the market is the same.

3.1.3 What problems do these Models address

Though it may not be obvious from the assumptions of these models, using the theories based upon these assumptions, we are able to select the optimal portfolio which balances the risk/reward profile in accordance with the investor's risk tolerance or reward requirements. The principle questions which we are able to address within each of the models are as follows:

- **Markowitz Theory** - Allows the optimal portfolio constructed from a collection of assets to be selected when the risk, expected return or the investor's utility function is known.

    1. What is the portfolio that can be constructed from a given set of assets which has the lowest risk for a given value of the expected return? (See the Portfolio.Markowitz.efficientFrontier(double, int) and Portfolio.Markowitz.efficientFrontier(double, double[][], double[], double) methods from the Markowitz class).

2. What is the portfolio which can be constructed from a given set of assets that has the greatest expected return for a given value of the total risk? (See the Portfolio.SolveFrontier class and in particular the methods Portfolio.SolveFrontier.findReturn(double, double[], double[]) and Portfolio.SolveFrontier.findReturn(double[], double[], double[], double[], double) of that class)
3. What is the portfolio(s) that can be constructed from a given set of assets which is optimal with respect to the investor's risk/reward utility function? (See the Portfolio.Markowitz.optimalPortfolio and Portfolio.Markowitz.optimalPortfolioMaxExpected methods from the Markowitz class).

- **Capital Asset Pricing Model (CAPM)** - Allows the optimal portfolio constructed from a collection of assets with the option of either lending or borrowing cash, to be selected when the risk, expected return, weighting of the Market Portfolio are known.

1. What is the portfolio which can be constructed from a given set of assets with the option of borrowing or lending cash at the prevailing market rate, such that the total risk is minimized for a given value of the expected return? (See Portfolio.CapitalMarket.weightCML to construct the optimal portfolio and Portfolio.CapitalMarket.riskCML to evaluate its total risk from the Portfolio.CapitalMarket class).
2. What is the portfolio that can be constructed from a given set of assets with the option of borrowing or lending cash at the prevailing market rate, such that expected return is maximized for a given value of the total risk? (See Portfolio.CapitalMarket.returnCML to evaluate associated expected return and then Portfolio.CapitalMarket.weightCML to construct the optimal portfolio from the Portfolio.CapitalMarket class).
3. What is the weighting of the Market Portfolio (i.e. non-cash part) within the optimal portfolio given by its expected return when the portfolio can be constructed from a given set of assets with the option of borrowing or lending cash at the prevailing market rate? (See Portfolio.CapitalMarket.weightCML in order to evaluate the weighting and then Portfolio.CapitalMarket.riskCML to evaluate its risk from the Portfolio.CapitalMarket class).

3.1.4 Range of Functionality contained within the classes

This Component contains the following business classes:

- Portfolio.AssetParameters - Auxiliary class that offers methods to assist in the evaluation and estimation of various parameters which are then used within the methods of the main classes.

- Portfolio.CapitalMarket - Implements the Capital Asset Pricing Model (CAPM). The CAPM is an extension of the Markowitz theory in that in the construction of an

optimal portfolio along with (risky) assets you may borrow or lend (zero risk) cash at the prevailing market rate.

- Portfolio.Interpolation - Offers methods by which the Efficient Frontier can be constructed from a finite set of points.
- Portfolio.Markowitz - Implements the Markowitz model, that is, we offer method that allow the portfolio with the least return to be constructed from a collection of assets.
- Portfolio.PerformanceEvaluation - Offers a number of procedures for accessing the return and risk adjusted return (Treynors Measure, Sharpes Ratio).
- Portfolio.SolveFrontier - This class complements the methods found within the Markowitz class which allow you to find for a given expected return the corresponded portfolio on the Efficient Frontier. Here we also allow yo to provide the value of the total risk and we will find the corresponding values of the expected return of the portfolio on the efficient frontier.
- Portfolio.TwoAssetPortfolio - Evaluation of the optimal weighting of a portfolio with two assets which can be used to analysis the effect of a single purchase or sale from an arbitrary portfolio.
- Portfolio.Volatility - Auxiliary class that offers methods to assist in the evaluation of the volatility, variance and covariance of the assets.

3.1.5 Problems with the Application of the Markowitz Theory and CAPM

When applying the Markowitz Theory and CAPM to real world problems concerning the construction of portfolios, we will in general be faced with the following difficulties.

Estimation of the Parameters

Problem: To apply any of these models we require knowledge of the expectation and variance of the return for every available investment and the covariance between every pair of investments. This information in practice is not obtainable. To get around this we use the historical rates as estimates, the idea being that in the near term at least, the rates will not vary significantly from there historical rates.

Our Approach: Within our Component we have offered a number of utility classes which assist in the evaluation and estimation of these quantities. The major of these auxiliary methods will be found within the Portfolio.AssetParameters class. With this class we offer methods for the estimation of the expected returns, variance and covariance in accordance with a historical and scenario based approach.

Scaling as the Number of Assets considered increases

Problem: Both the Markowitz Theory and CAPM application becomes computationally unyielding when a large number of investments are considered. For instance, on a typical desktop machine when 1 or 2 hundred assets are used then the application of these models will take a number of minutes; however if thousands of assets are used then the application of the model will take hours or even days.

Our Approach: Both the Markowitz and CAPM require the maximum of a given function to be determined. This requires that a low level multi-dimensional optimization algorithm is applied. These algorithms by there very are computational intension and hence the computational intensity of the application of the Markowitz and the CAPM cannot to completely avoided. However, we have mitigated the computational requirements of the application of this Component by:

1. **Refined Algorithm** - The optimization algorithm used here is an adapted version of Rosen's Algorithm that has been optimized to handle the particular optimization problems which are generated from the Markowitz Theory and CAPM. This custom algorithm was developed from the basis of our experience and developed algorithms found within the WebCab Optimization Component.

2. **This Components Design** - The Component is designed so that the Efficient Frontier and the Market Portfolio are constructed at a finite number of points after which the interpolation methods are applied. By designing the Component in this way you are able to call the optimization algorithms maybe 5 or 10 times in order to be able to construct the optimal portfolio for a wide range of possible values of the risk, expected return or Investors risk/rewards preferences.

   Moreover, the Component is actually designed so that the computation of the Efficient Frontier and the Market Portfolio which depend on the optimization algorithms can be completed at the beginning of the application of these models, and these algorithms will not need to be used during the subsequent application of the models. Therefore, within real life situations the closing prices and other market data can be collected from which the Efficient Frontier and Market Portfolio can be constructed using an over-night batch process. Once these objects have been constructed the application of the Markowitz and CAPM even for portfolios which can be constructed from 1,000's of assets can be carried out using a desktop machine in essentially real time.

Determining the Risk/Reward preferences of the Investor

Problem: Within the application of the Markowitz Theory when selecting the optimal portfolio with respect the Investor's risk tolerance it is necessary to provide the Investors Utility function which describes there riskpreferences. This Utility function allows a unique portfolio to be selected from the collection of portfolios which offer the least risk for a given

expected return. The difficulty here like with other non-observable quantities is in our ability to estimate this function from information which is provided by the investor.

Our Approach: The difficulty in obtaining the Investors Utility Function is in providing a framework which is both comprehensive and user friendly. Ideally, the Investors would be in a position to submit a function (in analytic form or otherwise) which relates the risk and expected return in accordance with his preferences. However, this requirement and resulting construction of the Utility function if not sufficiently intuitive in order to be widely applied. The approach we have taken here is to allow the Investor to given his risk/reward preferences at a finite number of sweet stops around which we will interpolate in order to obtain the Utility function itself. Clearly, one of the assumptions we are forced to make in this approach is that the Utility function is smoothly varying between the interpolation points given.

Remark The interval between and the location of the interpolation points should reflect nature of the investors preferences. In particular, in regions where the cubic spline interpolation function is likely to differ most from the (true) Utility function more interpolation points should be requested from the investor.

For further details concerning the application of this (and other) approaches please see the section 'Discovering the Investors risk - return profile').

Practicalities of Diversification

Problem: Though this problem is not particular to our (or others) implementations it is important to point out that in practice costs will be incurred when an asset is brought or sold, and the effects of diversification (in accordance with the Markowitz Theory and CAPM) will diminish as the number of assets considered within the collection of asset increases. Therefore in practice one should balance the benefits of diversification with the costs incurred in re-balancing a portfolio.

Solution: The level of costs will vary according to the type and quantity of assets being held within the portfolio. Generally speaking the more liquid the asset the lower the relative transaction costs will be for a given transaction size. Moreover, several studies have shown that it is possible to derive most of the benefits of diversification with a portfolio consisting of 12 to 18 holdings. That is, to be adequately diversified does not require 200 holdings in a portfolio which could incur significant trading costs.

Another approach to deducing the effects of trading costs is to use derivatives on the underlying assets rather than the assets themselves in order to re-balance the portfolios. The reason being that derivatives trading generally has significantly lower costs than trading in the underlying security.

3.2 Preliminaries and Auxiliary classes

Within this section we detail a number of auxiliary notions and the procedures that will be required by the main functionality offered by this Component.

Within the Portfolio.AssetParameters class we provide procedures for the evaluation of various quantities which are required within the application of this Component. These parameters include:

- Estimate of the Volatility of an asset prices:
  1. Historical Approach: Portfolio.AssetParameters.volatility(double[])
  2. Scenario Approach: Portfolio.AssetParameters.volatility(double[], double[])
- Estimate Expected Return of an asset:
  1. Historical Approach: Portfolio.AssetParameters.expectedReturn(double[])
  2. Scenario Approach: Portfolio.AssetParameters.expectedReturn(double[], double[])
  3. ARCH based Approach: Portfolio.AssetParameters.archExpectedPriceEstimate
- Estimate Covariance Matrix of the collection assets:
  1. Historical Approach: Portfolio.AssetParameters.covarianceMatrix(double[][])
  2. Scenario Approach: Portfolio.AssetParameters.covarianceMatrix(double[], double[][])

Note Further methods for the estimation of the Volatility are provided within the Portfolio.Volatility class.

3.2.1 Choices in Approach and other issues

As listed about the main two approaches by which the expected return, volatility and covariance are estimates is the historical and scenarios based approaches. Here we motivate the essential differences between these methods. We also give a brief explanation of the ARCH procedure for the estimation of the expected returns along with advice concerning the number of historical values which should be used within the application of the historical estimate.

When should the scenario approach be used?

One such instance is when a takeover of a quoted company has been announced and the share price converges to almost the offer price in anticipation of the takeover being completed. In this scenario the more likely the takeover will be completed the closer the price will converge to the offer price. However, if the takeover breaks down then the price is likely to experience sharp moves to the price level found prior to the intended takeover

being announced. By estimating the probability of each scenario and the likely level of volatility resulting we are able to give a realistic forward looking estimate.

When should the historical approach be used?

If the market under consideration goes through seasonal or business cycles, or if a given company has transformed itself then the observations used in order to estimate the expected volatility should reflect these issues. For example, if company which was a diversified general industrial company has since refocused on certain key areas, then in terms of estimating its expected volatility from historical values it is reasonable to only consider the period after the company refocused.

The number of historical values which should be used

The number of historical values used here in order to estimate the volatility should reflect the length of the period over which a reliable estimate of the volatility is required. For example, if an estimate of the 1-month volatility is sought then it is reasonable to use at least the last 1-month's historical values up to a few years of historical values.

ARCH type models for Estimate Returns and Volatility

The ARCH model in the general sense can be thought of as a (linear) weighted scheme. Here we allow the past historical asset prices to have a weighted effect, along with a weighted measurement of the expected value in accordance with the assets characteristic line. Note, that when we set all the weights of the historical values to zero this estimate reduces to an estimate based on extrapolation of the characteristic line.

When to use the ARCH approach

This method assumes that the asset under consideration exhibits a long term drift (in particular, linear correlation) with respect to some market index. Therefore, this estimate is particularly applicable to assets which have been observed to exhibit significant linear correlation to market indexes such as the FT100, S&P500 etc.

3.2.2 Basic Formulae for the Return, Covariance, Correlation and Risk

Within this subsection we collect together a number of formulae which illustrate how the scenario and the historical approach can be used in order to either define or investigate relationships between the return, covariance, correlation or risk of a portfolio and its assets.

Definition of the Return of a Portfolio

Before we move onto the Markowitz Theory we should define exactly what we mean by the return of a portfolio. The return of a portfolio over a given period is simply the weighted

arithmetic average of the returns of the individual assets. That is, the return r_p of a portfolio over t periods is given by:

$$r_p = \sum_{i=1}^{N} x_i r_i(t) \tag{3.2.1}$$

where x_i is the weight for the asset i and $r_i(t)$ is the return on the asset i over the t periods. Note that by definition the weights x_i, for any portfolio must add up to 1, i.e. $\sum_{i=1}^{N} x_i = 1$.

Remark Note that this definition depends on knowledge in the returns of the individual assets. Therefore, if we use the forwarding looking scenario based approach to estimating the return of the asset then the above formulae will evaluate the forwarding looking (expected) value of the (expected) return of the portfolio. Similarly, if we had evaluated the historical returns using historical values then the above formulae would refer to the evaluation of the historical value of the return of the portfolio.

Evaluating the Covariance of Returns (Scenario Approach)

The covariance of returns is a statistical measure of how the returns of two assets are correlated. That is, to what degree do they move together. (In the following section we offer more motivation and show how the covariance and correlation are connected).

Using the forward looking scenario approach the covariance of returns σ_{ij}, between two assets i and j is given by:

$$\sigma_{ij} = \sum_{s=1}^{N} P_s[r_{is} - E(r_i)][r_{js} - E(r_j)] \tag{3.2.2}$$

where,

- r_{is} (respectively r_{js}) denotes the rate of return for the asset i (respectively j) in the state s
- $E(r_i)$ (respectively $E(r_j)$) is the expected return for the asset i (respectively j)
- P_s is the probability of the state s occurring

Our Implementation: An approach to estimating all the covariances between all the pairs of a collection of asset (i.e. the covariance matrix has been implemented within the method Portfolio.AssetParameters.covarianceMatrix(double[], double[][]).

Correlation, Covariance and the Volatility (aka standard deviation)

Though knowledge of the Correlation is not directly used within the full application[3] of the Markowitz and CAPM, the notion does play an important role. The principle reason

[3] In the case of a portfolio with two assets some of the results are phrased in terms of the correlation.

being that the effects of 'diversification' (explained below) which Portfolio Theory uses can be influenced by the level of correlation between the assets from which the portfolios can be constructed. In particular, from the perspective of the Portfolio Theory if two assets are perfectly correlated then one would expect them to have the same level of expected return since otherwise the asset with the lower expected return could be excluded from the collection of assets. Below we further illustrate these points by giving the primary relation between the Correlation, covariance and standard deviation (or volatility).

The correlation coefficient ρ_{ij} is another measure of the relationship between two assets i and j. The correlation coefficient is a measure which lies within the closed interval $[-1, 1]$, where a value of -1 which depicts that the assets are perfectly negatively correlation meaning that their prices always move in opposite directions and 1 which depicts that the assets are perfectly positively correlated, meaning that their prices always move in the same direction.

The Correlation, covariance and Volatility (also know as the standard deviation) are related by the following equation:

$$\rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j} \tag{3.2.3}$$

where σ_i and σ_j is the standard deviations of the returns of the assets i and j, and σ_{ij} is the covariance of the returns between the assets i and j.

Scenario Approach to the Correlation Coefficient and the Covariance

Since the covariance is given by (3.2.2), it only remains to find the two standard deviations (or volatility) σ_i and σ_j, in order to evaluate (3.2.3). The standard deviation of an asset is given by the following equation:

$$\sigma = \sqrt{\sum_{t=1}^{N} P_t [r_t - E(r)]^2} \tag{3.2.4}$$

where P_t is the probability of the t-th state occurring, r_t denotes the historical average return when the t-th state occurs and $E(r)$ is the expected (future) return of the asset.

Remark If we are considering only two assets A and B then a computationally efficient way to calculate the correlation coefficient between A and B over N periods is by using the following expression:

$$\sigma_{ij} = \frac{N \sum_{t=1}^{N} A_t B_t - \sum_{t=1}^{N} A_t \sum_{t=1}^{N} B_t}{\sqrt{\left[(N \sum_{t=1}^{N} A_t^2) - (\sum_{t=1}^{N} A_t)^2\right] \left[(N \sum_{t=1}^{N} B_t)^2\right]}} \tag{3.2.5}$$

where A_t (respectively B_t) is the percentage increase of the asset A (respectively B) in the t^{th} period.

Evaluating the Expected Return(s) (Scenario Approach)

The following expression is often used to define (average) expected return $E(r)$ of an asset over some (future) time period:

$$E(r) = \sum_{s=1}^{N} P_s r_s \tag{3.2.6}$$

where P_s is the probability of the state s occurring and r_s is the corresponding expected return of that given state.

Our Implementation: Such an approach has been implemented within AssetParameters.expectedReturn(double[], double[]).

Covariance of the Expected Return (Historical Approach)

Similarly, we may use an historical approach in order to estimate the covariance between the expected return of assets. That is, if we know the historical rates of return then the covariance is given by the expression:

$$\sigma_{ij} = \frac{1}{N} \sum_{t=1}^{N} (r_{it} - \bar{r}_i)(r_{jt} - \bar{r}_j) \tag{3.2.7}$$

where,

- N is the number of equally likely paired observations
- r_{it} (respectively r_{jt}) is the return of the asset i (respectively j) in the t^{th} period
- $\bar{r}_i$ (respectively $\bar{r}_j$) is the historical average return for the asset i (respectively j)

Our Implementation: The allow the covariance to be evaluated/estimated uses a scenario or historical approach within Portfolio.AssetParameters.covariance(double[], double[], double[]), or Portfolio.AssetParameters.covariance(double[], double[]) respectively. The evaluation of the covariance is only required when portfolio with only two assets are required. For portfolio with more than two assets you will be required to evaluate the covariance matrix of the collection of assets from which the portfolio can be constructed. Within the Portfolio.AssetParameters class we provide the methods Portfolio.AssetParameters.covarianceMatrix(double[][]), and Portfolio.AssetParameters.covariance(double[], double[][]), which allow the covariance matrix to be evaluated via historical or a scenario based approach respectively.

Variance and Expected Return of a Portfolio with n holdings

We give the general formula for the **variance** $\sigma^2(r_p)$, of the returns for a portfolio of N assets:

$$\text{var}(r_p) = \sum_{i=1}^{N} \sum_{j=1}^{N} x_i x_j \sigma_{ij} \tag{3.2.8}$$

where x_i and x_j are the weights for the assets i and j, σ_{ii} is the variance for the i^{th} asset, σ_{ij} is the covariance of the assets i and j.

Remark The standard deviation of the portfolio σ_p, is always the square root of the variance (i.e. $\sigma_p = \sqrt{\text{var}(r_p)}$).

The **expected return** $E(r_p)$, is given by:

$$E(r_p) = \sum_{i=1}^{n} x_i E(r_i) \tag{3.2.9}$$

where $E(r_i)$ is the expected return of the i^{th} asset and x_i is the proportion of the portfolios value invested in the i^{th} asset.

Remarks

- If we hold a positive quantity of the asset i (i.e. go long) then clearly $x_i > 0$. We are able to model going short on an asset i, by letting $x_i < 0$. Therefore, if we set $x_i \geq 0$ then no short selling is allowed.
- For a portfolio with a large number of holdings the covariance terms will dominate the variance terms. Hence, the risk of the portfolio will depend more on the average covariance between the investments than on the risk of the investments themselves.

3.3 Expected Return and Risk from a Portfolio with two Assets

In the preceding discussion we have been setting up the language in which to express Portfolio Theory. Here we will start our discussion of Portfolio Theory itself. In particular, we will discuss formulae which detail the interaction between the risk and return of a portfolio and how it can be constructed to ensure the minimal risk. Here we will deal with the special case of portfolios which consist of two assets.

3.3.1 Motivation

This case is particularly useful because the relevant formulae take a simple form and we are able to calculate the effect of a single purchase (or sale) has to a portfolio by viewing the portfolio (or portfolio minus a holding) itself as a single asset.

In the case of portfolios with only two assets, evaluation of the portfolio risk, expected return and weights for the portfolio with the minimal risk are all closed formulae. As mentioned above the usefulness of these closed formulae is that the effect of a single purchase (or sale) has to a portfolios risk/reward profile can be studied by viewing the portfolio (or portfolio minus a holding) itself as a single asset. Note that this portfolio which is

viewed as a single asset could however have been constructed by use of the Markowitz (see Portfolio.Markowitz class) or CAPM (see Portfolio.CapitalMarket class) Theories.

3.3.2 Formulation for a Portfolio of two assets

If we can construct portfolios from two assets A and B, then the return of the **general portfolio** P, will take the form:

$$P = \alpha A + (1 - \alpha)B$$

where $\{\alpha : 0 \leq \alpha \leq 1\}$, determines the weighting of the portfolio between the two assets A and B.

If the corresponding **expected returns** of the two asset from which portfolio can be construct are $E(A)$ and $E(B)$, then the expected return of the general portfolio $E(P)$ is:

$$E(P) = \alpha E(A) + (1 - \alpha)E(B) \tag{3.3.10}$$

with a **standard deviation** of:

$$\sigma_p = \sqrt{\alpha^2\sigma_A^2 + (1 - \alpha)^2\sigma_B^2 + 2\alpha(1 - \alpha)\sigma_{AB}} \tag{3.3.11}$$

By using the identity (3.2.3), $\sigma_{AB} = \sigma_A\sigma_B\rho_{AB}$, we can rewrite (3.3.11) as:

$$\sigma_p^2 = \alpha^2\sigma_A^2 + (1 - \alpha)^2\sigma_B^2 + 2\alpha(1 - \alpha)\sigma_A\sigma_B\rho_{AB} \tag{3.3.12}$$

3.3.3 Minimum risk of a portfolio with two assets

By a simple application of calculus the portfolio with the minimum risk for some corresponding value of the expected return occurs when the weighting of the asset A (α) is equal to:

$$\alpha = \frac{\sigma_B^2 - \rho_{AB}\sigma_A\sigma_B}{\sigma_A^2 + \sigma_B^2 - 2\rho_{AB}\sigma_A\sigma_B}$$

where α is the percentage (in decimal format) invested in asset A within the two asset portfolio. Hence the percentage invested in the other asset B is $(1 - \alpha)$.

Please note that the weights given above are not assumed to lie within the interval $[0, 1]$, and in fact can be any real number. If the weight of the first asset is greater than one, say w, then it means that the portfolio with the minimal risk which can be constructed from the two assets consists of a leveraged position in the first asset which is funded by going short in the second asset (i.e. with a weight $1 - w$). Similarly, if the weight of the first asset is negative then the portfolio with the minimal risk consists of a short position in the first asset and a geared position in the second.

Remark Please note that often in practice it is necessary to provide margin for short

positions and hence in practice the portfolio with the minimal risk which contains a short position may require extra capital.

In particular, for the special cases when the assets are uncorrelated (i.e. knowledge of the movement of one asset either up or down does not imply anything concerning the movement of the other) and perfectly negatively correlated (i.e. the assets always move of opposite directions)we have:

- A and B are uncorrelated (i.e. $\sigma_{AB} = 0$) is given by:

$$\alpha = \frac{\sigma_B^2}{\sigma_A^2 + \sigma_B^2} \tag{3.3.13}$$

- A and B are perfectly negatively correlated (i.e. $\sigma_{AB} = -1$) is given when:

$$\alpha = \frac{\sigma_B}{\sigma_A + \sigma_B} \tag{3.3.14}$$

Concluding Remarks: Above we have constructed the portfolio under various conditions which exhibits the minimal risk. However, often an investor will wish to balance the risk against the return which is the main question we will consider the following sections.

3.4 Markowitz Theory

The Markowitz Model is used to analyze the construction and qualitative nature of a portfolio's risk-return characteristics. In particular, we offer methods by which the continuum (in risk and expected return) of the portfolios (known as the Efficient Frontier) which consists of the portfolios with the minimum risk for a given value of the expected return constructed from a given collection of assets. Moreover, from this collection of Portfolios a unique optimal portfolio can be selected with respect to a given value of the expected return, risk or an investors risk - return profile given in terms of a utility function.

3.4.1 Overview of Markowitz Theory Implemented

The Efficient Frontier is the collection of portfolios constructed from the given set of assets which have the lowest possible risk for a given level of the expected return. Note that the weights of the assets making up the portfolio may themselves be subject to constraints (for example, no one asset can have a weighting more than 20 percent or less the 5 percent) which we will deal with below.

Once the Efficient Frontier is known, we are able to select from this continuum a unique portfolio which represent the optimal portfolio with respect to an investors risk - return profile. The return profile of the investor may be given in three distinct ways and the correspond optimal portfolio can them be constructed. The by one of the following means:

1. With respect to the investors risk - reward utility function, see Portfolio.Markowitz.setUtilityFunct Portfolio.Markowitz.setUtilityFunctionPoly.

2. Maximum Risk - the investor gives the (maximum) risk which they are prepared to accept and then the corresponding portfolio with the highest expected return for that given level of risk is constructed.

3. Expected Return - the investor gives the expected return which they desire and the portfolio with the least risk for that given level of expected return is constructed.

3.4.2 Construction of the Efficient Frontier

For a given collection of securities with various risk profiles, an investor can construct a whole range of portfolios with various risk-return characteristics. In particular, there exists a continuous family of portfolios within the range of the risk and expectation of the individual securities. The investor will wish to choose a portfolio from this family which offers the lowest level of risk for a given expected return, according to the assumptions of the Markowitz model.

We consider the family of all possible portfolios formed from a combination of the available securities. Within this family we choose a locus of points by selecting the portfolios which offer the least risk for a given level of the expected return. Such optimal portfolios will exist within the range of the expected returns. This locus of points is referred to as the **Efficient Frontier**.

The Efficient Frontier is constructed by the following steps:

1. Evaluated the Efficient Frontier at a finite number of points. That is, find the portfolio which exhibits the lowest risk for a given expected return.

2. Interpolate about these points using cubic spline (or some other method) in order to construct the Efficient Frontier.

The points on the Efficient Frontier are portfolios constructed from the set of assets considered which exhibit the lowest risk for a given expected return. These portfolio are described by the following three properties:

1. Expected Return - The expected return of the portfolio which is estimated from the historical returns of the assets within the portfolio.

2. Total Risk - The total risk of the portfolio which is estimated from the historical returns of the assets within the portfolio.

3. Asset Weights - the weights of the collection of assets from which the portfolio can be constructed.

It is important to point out that the Efficient Frontier in monotonically increasing function in risk and expected return. This means that if we are given a value of the expected return then there will correspond a unique portfolio on the Efficient Frontier with a given total risk. Conversely, if we are given the total risk of the portfolio then there will exist a unique portfolio on the Efficient Frontier with a corresponding value its expected return.

Constraining the Weights of the assets of the Efficient Frontier's Portfolios

Within our implementation we offer the possibility to constrain the weights of the assets from which the portfolios on the Efficient Frontier are constructed. The constraints on the weights on the portfolios are set by using the method setConstraints. We illustrate the use constraints with the following example.

Say an investor requires a portfolio selected from n asset which has the lowest risk for a given expected return but also has the requirement that all of the assets must have a weight between 0.05 and 0.1 (i.e. between 5 and 10 percent). In this instance we would need to set the constraints on the assets to be:

lowerBounds = { 0.05, 0.05, 0.05,...., 0.05 }
upperBounds = { 0.1, 0.1, 0.1, ..., 0.1 }

where each of the arrays above has the same number of terms as the number of assets from which the portfolio can be constructed. Constraints can be set on the assets from which the Efficient Frontier is constructed by using Portfolio.Markowitz.setConstraints, constraints can also be added within the context of the evaluation of the Efficient Frontier within the CAPM using Portfolio.Markowitz.setConstraints.

Remark: If the constraints are not set then they will take there default values which are 0 and 1, for the lower bound respectively upper bound of each asset. That is, they will remain as weights in the usual sense.

3.4.3 Constraints effect on the range of the Efficient Frontier

The placing of constraints on the weights of the assets effects the range of expected returns for which the resulting portfolios can be constructed. Since the (constrained) Efficient Frontier is just a collection of portfolios subject also subject to the constraints which minimize the risk for a given level of the expected return. The range of values over which the Efficient Frontier exists must correspond to the range of expected returns of the possible constructed portfolios.

Within the Markowitz and the CapitalMarket classes we provide the methods `maxFrontierReturn`, and `minFrontierReturn` we allow the maximum and respectively minimum values of the expected return over which the (possibly constrained) Efficient Frontier exists. We also offer two associated methods `maxFrontierReturnWeights` and `minFrontierReturnWeights`,

which evaluate the assets weights of the portfolio at these two ends points. These two methods which construct the Portfolios on the Efficient Frontier at its end points have the significant advantage of having almost no computational overhead, unlike the construction of the portfolios on the Efficient Frontier at other points.

Also, note that since the Efficient Frontier is monotonically increase in risk and expected return, the therefore the point at which the maximum and minimum expected return on the Efficient Frontier occur is also the points at which the maximum respectively minimum of the risk of the portfolios on the Efficient Frontier occur.

Performance Issues

The introduction of constraints on the weights of the portfolios which form the Efficient Frontier will have the following consequences with regards to overall performance:

- **Upper Bounds** - the presence of upper bounds (not all equal to 1) on the asset weights will result in the construction of the corresponding Efficient Frontier requiring significantly more computational resources. If however the upper bounds are all set to 1 (i.e. there default value), then they will not effect the computational demands required to evaluate the points on the Efficient Frontier. This allows lower bounds to be set on the asset weight without reducing the performance of this class.
- **Lower Bounds** - the presence of lower bounds on the asset weights will have no significant effects on the efficiency of the construction and analysis of portfolio on the Efficient Frontier.

Notes on the Evaluation of the Efficient Frontier and the Optimal Portfolio

To calculate the Efficient Frontier, Rosen's gradient projection optimization algorithm is used. If you directly try to evaluate the optimal portfolio with respect to an investors utility function then you will need numerous applications of Rosen's algorithm which will become computationally intensive. Therefore, we designed this class so that this would not be necessary by allowing the computation at the beginning a number of points on the Efficient Frontier, from which the other points will be deduced (in fact, estimated) through the use of cubic spline interpolation. These interpolation points are determined by calculateEfficientFrontier, which must be called prior to any subsequent method which depends on the Efficient Frontier being known.

The use of this approach will also allow us to deal with the possible situation.

3.4.4 Consistency of the Assets and Effects on the Efficient Frontier

This section deals with a rather technical issue concerning how the consistency of the collection of assets from which the Portfolio on the Efficient Frontier can be constructed. By

consistency we mean consistency with the assumptions of Markowitz Theory and CAPM, and how these inconsistencies effect are ability the construct optimal portfolio in a neighborhood of the lower bound of the expected returns over which the Efficient Frontier exists. That is, how appropriate the portfolios constructed from the available ("inconsistent") assets are to issues concerning the selection of the optimal portfolio. Hence the selection of either portfolios with the lowest risk for a given expected return or portfolio with the highest expected return with a given maximum risk.

When these effects occur

Note that the effects we will consider with this section will have relevance in the following situations:

- Only effect a neighborhood of the lower bound of the expected returns over which the Efficient Frontier exists.
- Will only occur when the portfolio considered are constructed from collections of assets which are not consistent (see note below) the assumptions of Markowitz Theory and CAPM.

Remark Similar effects do not occur in a neighborhood of the upper bound of the expected return of the Efficient Frontier because at the upper bound by definition there do not exist any portfolios with a higher expected returns so for that level of returns there are no preferred portfolios.

Advice to the Reader

We advise the reader to skip this section at the first reading, and refer back if you encounter such effects with the collection of assets you are using in order to construct the portfolios within your given application.

Nature of Inconsistency

The way in which the collection of assets can be inconsistent lies in the fact that the Markowitz Theory (and as we will see later the CAPM) assume that the investor will only take on additional risk for a higher level of expected return. Therefore, in order for a collection of assets to be consistent with these assumptions there must exist a portfolio with a greater return if it has a higher risk. Unfortunately, this may not be the case if the assets do not obey the rule that for a higher level of the expected return the risk increases.

If such a portfolio exists then the Efficient Frontier should only be considered on a subset over the entire range of expected values on which it exists when selecting an optimal portfolio. In particular, we will need to increase the lower bound of the expected returns over which the Efficient Frontier is considered. The precise construction of the lower bound will in general involve analysis of the differential of the Efficient Frontier. We implement these procedures within the `MaxRange` class.

When Inconsistency will effect the Efficient Frontier application

He we detail when the effects of inconsistency of the historical source data will effect the Efficient Frontiers ability the select the optimal portfolio. We only offer conditions under which such effects are certain to take place. We refer the reader to the section on precise lower bounds for a precise statement under which these effects take place.

When the asset with the lowest expected return does not have an upper bound on its weight then this effect is certain to occur when:

> 'the asset with the lowest expected return does not have the lowest risk and does not have a constraint placed on its upper bound'

In the case when this asset does not have an upper bound on its asset weight then these effects are certain to take place if:

> 'The portfolio constructed by taking the maximum weight of the asset with the lowest expected return, then the maximum weight of the asset with the next to lowest expected return and so on... until the sum of the weights is one. If there exists an asset from the remaining assets with a higher expected return with a lower risk.'

Illustration of the Effect of Inconsistency

We illustrate in the diagram below the shape of the Efficient Frontier which can occur if the collection of assets are not consistent.

Notes of Illustration: For all portfolios on the Efficient Frontier (i.e. red line) which have a risk which lies between C and D, or equivalently an expected return which lies between A and B; there exists a portfolio which has the same level of risk but has a high level of return. Moreover, for all points (except end point) there exists a portfolio with a lower risk and a higher expected return. Therefore, this section of the Efficient Frontier should be ignored with respect to the construction an optimal portfolio from the assets available (represented by small black squares in diagram).

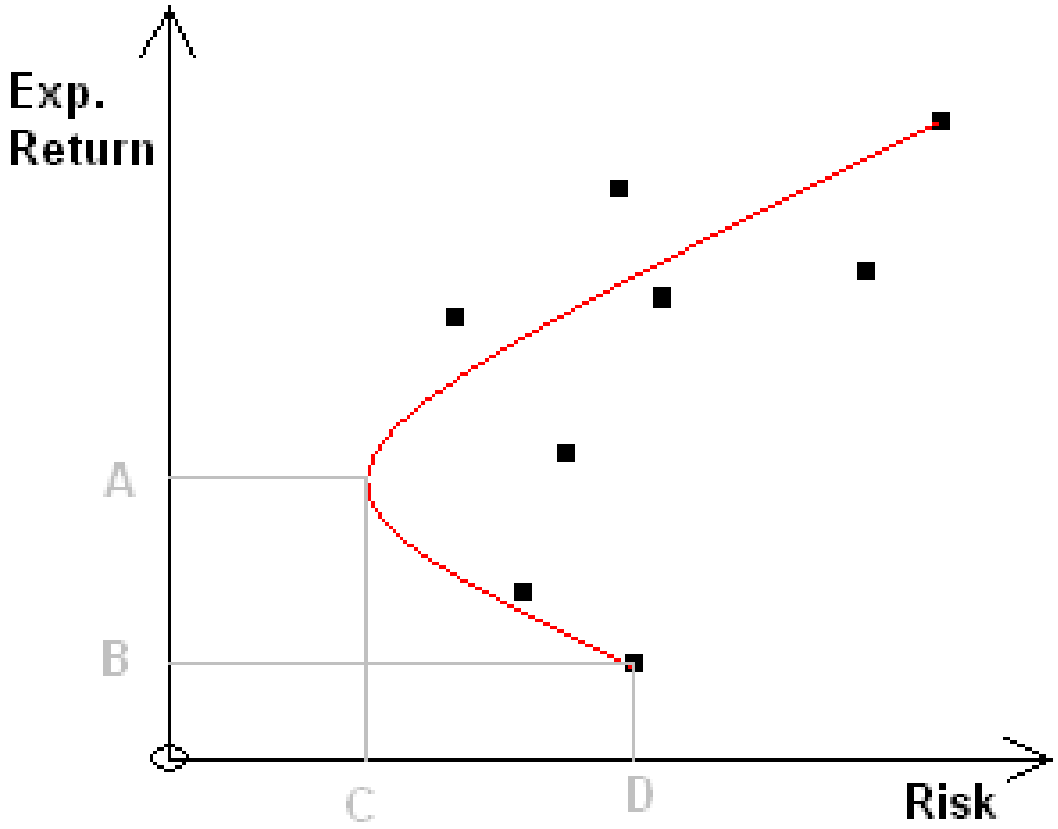


Fig: The loop-back effect at the lower end of the range of the expected return

Further Remarks: Note that in order to obtain the portfolios with the lowest respectively highest values of the expected return it is necessary to either construct a portfolio which consists solely of the asset with the lowest respectively higher expected return (we assume that these assets weights are not constrained). For this reason there is a rationale for considering the Efficient Frontier at both of these points, since they are unique they must by definition be the portfolio with that given (exact) level of return with the lowest risk. However from the prospective of selecting the optimal portfolio corresponding to an expected return at A (in the above diagram), it is not appropriate.

Another motivation for including the portfolio corresponding the `A`, and similar non-optimal portfolios in the Efficient Frontier is that the portfolios at the extremes of the Efficient Frontier lie at the vertex of the arced region (see dark gray shading in diagram below) in which all the possible portfolios which can be constructed from the available assets lie.

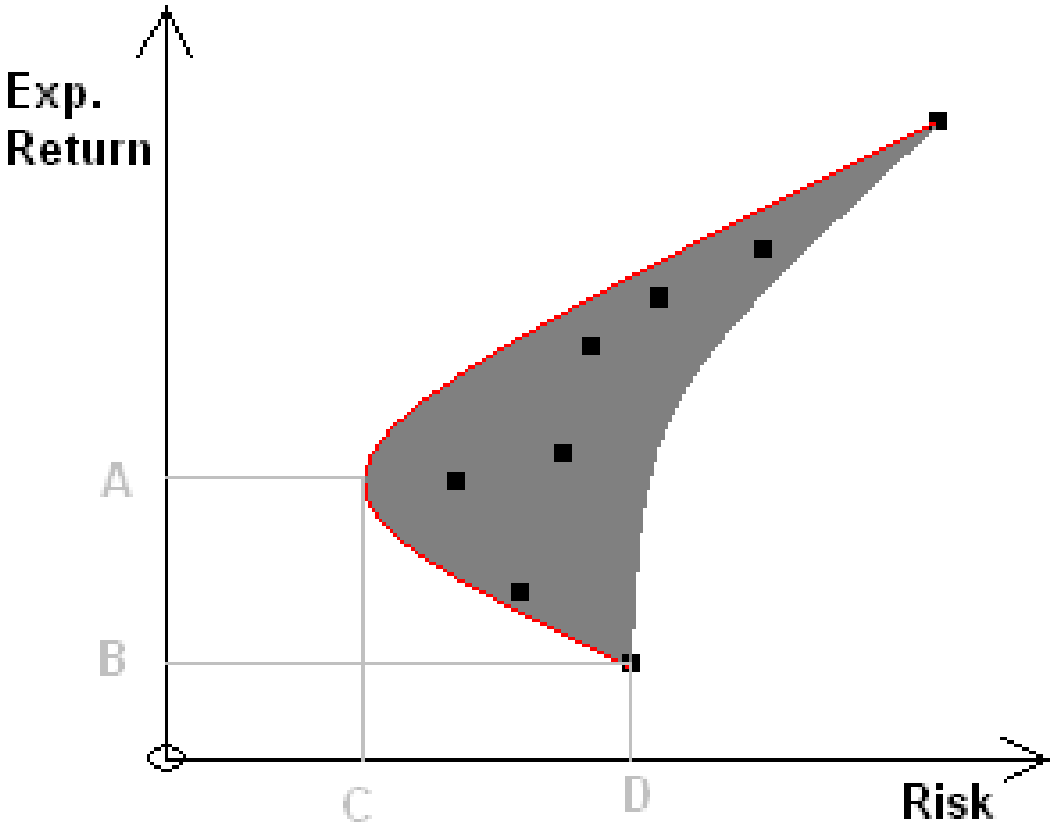


Fig: All possible portfolios lie within the dark gray region.

Approximate Lower Bound of the 'Maximum Range'

Therefore, from the point of view of applying the Efficient Frontier to selecting the optimal portfolio we will need to evaluate a (new) lower bound greater than the bound over which the Efficient Frontier exists over which the Frontier does only contain (justifiable) optimal portfolios. Here we will provide a quick and easy approach which allows a lower bound to be evaluated. Note that this lower bound will often not be optimal.

No Upper Bound Constraint on asset with the Lowest Risk

In the case when there are no constraints placed on the weight of the asset with the lowest risk we are able to evaluate the (approx.) range of the expected return which should be used when constructing the optimal portfolio is given by:

- Upper Bound on the Expected Returns: The expected return of the asset with the highest expected return.
- Lower Bound on the Expected Returns: The expected return of the asset with the lowest RISK.

If you use this range of expected returns then the Efficient Frontier constructed will not display the problem of suggesting the below optimal portfolio. Note that in this instance if

you require a portfolio below this lower bound of the expected return for which the exists an portfolio (i.e. an expected return is above the expected return of the asset with the lowest expected return and below the lower bound set). Then you should just select the portfolio on the Efficient Frontier with a risk equal to the risk of the asset with the lowest risk.

With Constraints

In the case when there are constraints on the weights of the assets (or at least an upper bound constraint on the asset with the lowest risk) then a construction based on similar principles applies. This construction is as follows:

- Upper Bound on the Expected Returns: The expected return of the asset with the highest expected return.
- Lower Bound on the Expected Returns: If you have an upper bound on the asset with the lowest risk then form a portfolio by taking the maximum weight of this asset then the asset with the next highest risk and so on... until the sum of the weights is one. Then the lower bound on the expected return should be the expected return of the portfolio constructed in this means.

These constructions have been implemented and offered as public methods within the `MaxRange` class.

Discussion and Illustration of this Approach

In the below diagram we illustrate this approach. Note that in this case the lower bound corresponding to a risk of `F`, is not optimal since the optimal points has an expected return of `A`, and risk of `C`. Also, note that only the Efficient Frontier to the right of the intersection will be considered (denoted by the arrow).

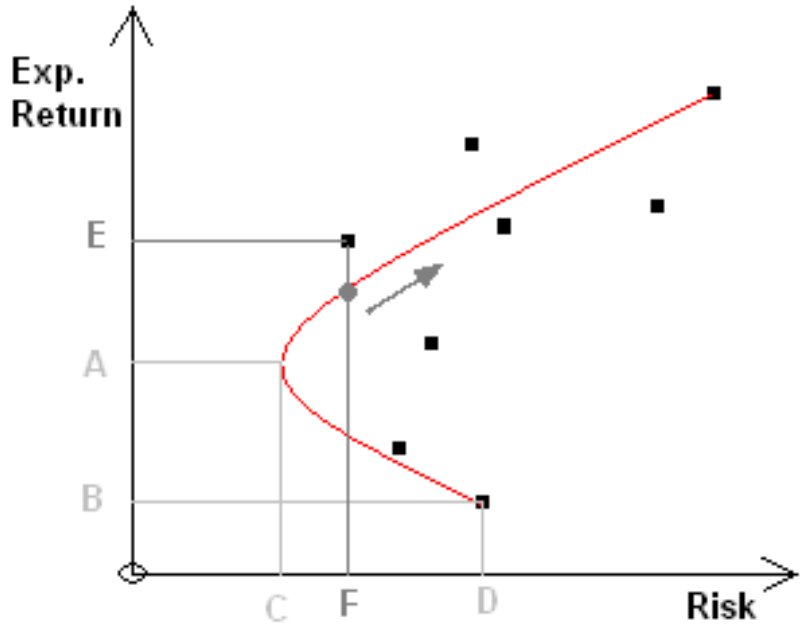


Fig: All possible portfolios lie within the dark gray region.

When we are interested in portfolios with an expected return above the lower bound provided by this approach then this approach is quite sufficient. However if we are interested in portfolios which are below this bound then we will require the more precise (and computationally demanding approach) given below. There are also instances in which the data is so inconsistent that this approach will return a lower bound which is equal to the higher bound. For an un-constrained asset set we illustrate this possibility within the following diagram.

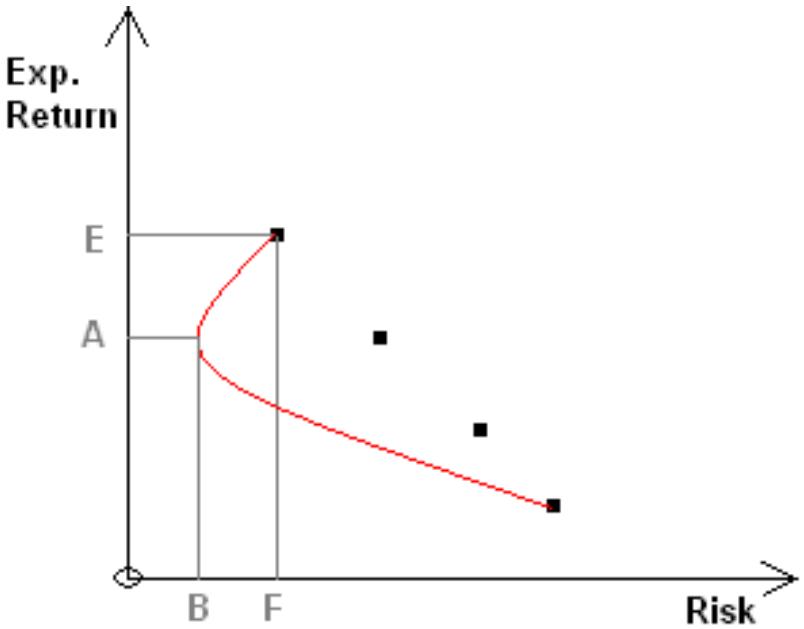


Fig: When the (approx) lower bound equals the upper bound.

Exact Evaluation of the Lower Bound of the 'Maximum Range'

Though the above approach will yield an range over which all portfolios of the Efficient Frontier are optimal portfolios, the lower bound on the range of expected returns will not in general be optimal. In order to provide a precise evaluation of the lower bound of the expected return we will need to use an algorithm which considers the differential of the Efficient Frontier. In particular, we will need to construct the Efficient Frontier of the possibly constrained assets and evaluate unique point (if it exist) where the rate of change of the risk with respec to the expected return is zero. If this point is a minimum then the corresponding value of the expected return at that point is the precise lower bound over which the Efficient Frontier should be considered when used to construct an optimal portfolio.

We use this indirect (i.e. via Frontier) approach because even if the historical source data is not consistent with the assumptions of Portfolio Theory the above mentioned effects may not take effect. We offer procedures which deal with this case within the `MaxRange` class.

3.4.5 Selecting the Optimal Portfolio

The construction of a portfolio with regard to an investors risk - reward profile lies at the core of Markowitz Theory. By obtaining information concerning the value judgments to the various risk - reward combinations which will be available to the investor we are able to select the optimal (or preferred) portfolio for the continuum of portfolios on the Efficient Frontier. If we where not aware of a given investors risk preferences then it would not be possible to select a preferred portfolio on the Efficient Frontier. There are different level of granularity of the investors preferences which are sufficient in order to select an optimal portfolio. At one end the investor may only provide information concerning the maximal level of risk acceptable or the level of expected return desired from which an optimal portfolio can be selected. On the other hand the investors utility function may be given which offers much more fine grained information concerning this investors risk-reward preferences and allows a value judgment to be assign to a continuum of risk-reward combinations.

As mentioned above and within the over view of this section within Markowitz Theory there are essentially three ways in which to select an optimal portfolio from the selection of portfolios known as the Efficient Frontier which are each optimal for there corresponding value of the expected return or risk. The three mechanism for selection the optimal portfolio are as follows:

1. **Give value of the Expected Return** - Since the Efficient Frontier is monotonically increasing and continuous in the expected return a point and hence portfolio can be selected for a given value of the expected return over the range of the expected return. Once the (possibly constrained) Efficient Frontier has been constructed we are able to select an optimal portfolio by using Portfolio.Markowitz.efficientFrontier(double, int); alternatively you may wish to use the approach of Portfolio.Markowitz.findRisk(double, double[], double[])

2. **Given value of the Risk** - Since the Efficient Frontier is monotonically increasing and continuous in the value of the risk a point and hence portfolio can be selected for a given value of the expected return over the range of the total risk. Once the (possibly constrained) Efficient Frontier is known we are able to select an optimal portfolio by using Portfolio.SolveFrontier.findReturn(double, double[], double[])

3. **Provide the Investors Utility Function** - The investors utility function is the locus of points at which the investor gets a particular level of satisfaction or utility from a combination of expected return and risk. This function may be a function with respect to the expected return or risk. By identifying the points at which the utility function and Efficient Frontier coincide we are able to identify a collection of portfolios which are optimal with respect to the investors expressed risk/return preferences. If the utility function is given as a function of the expected return then you can use Portfolio.Markowitz.optimalPortfolio(double, double, double[][]), Portfolio.Markowitz.optimalPortfolioMaxExpected(double, double, double[][]), or alternatively you may wish to use the approach; Portfolio.SolveFrontier.findRisk(double[], double[], double[], double[], double). If the utility function is given as a function of the total risk then the you will need to used Portfolio.SolveFrontier.findReturn(double[], double[], double[], double[], double). For further details concerning the details of our implementation we refer the reader to Portfolio.Markowitz or Portfolio.CapitalMarket

Since the Efficient Frontier is monotonically increasing in the expected return and risk the application and nature of selecting the optimal portfolio from knowledge of the expected return or risk is reasonably straightforward and we refer the reader to the Portfolio.Markowitz and Portfolio.SolveFrontier for further discussion. However, with regard to the use of the investors utility function there are a number of issues such as the means and equivalent of the ways in which the utility function are given and the means by which a solution is found which should be detailed further. In the following discussion we will treat each one of these issues in turn.

Providing the Investors Utility function

A utility function is the locus of points at which the investor gets a particular level of satisfaction or utility from a combination of expected return and risk. Clearly, each investor will have their own utility function depending on their individual trade-off between expected return and risk.

We provide methods by which the optimal portfolio can be selected from the Efficient Frontier when the investors utility function is given. Within our implementation you are able to provide the investors utility function in one of two ways:

- **Set of Interpolation Points** - The utility function is given on a finite set of points around which it can be interpolated in order to construct a continuous utility function. This can be done by using Portfolio.Markowitz.setUtilityFunctionInterp.

- **Given as a polynomial expansion by providing its coefficients** - We are able to construct a polynomial which represents the utility function if the coefficients of the polynomial are given. This can be done by using Portfolio.Markowitz.setUtilityFunctionPoly.

Structure of the 'Polynomial' and 'Interpolated' Utility Function

As mentioned above the investors utility function can be given as a set of interpolation points or in polynomial form. Here we provide further details as to the exact structure of the utility function (in either form) which will need to be provided within our implementation.

1. **Structure of the polynomial which defines the Utility Function**

   The polynomial which defines the investors utility function takes the following form:

   $$p(x) = coef[0] + (coef[1] * x_i) + \ldots + (coef[n-1] * x_{n-1})$$

   where the $coef[i]$, $i = 0, ..., n-1$ are coefficients of the polynomial utility function which are provided as a parameter. Now within this representation the values at the variable x, corresponds to the risk level of a polynomial and the corresponding value of $p(x)$, is the value of the expected return which to investors demands in order to be exposed to the chosen level of risk.

2. **Structure of the Interpolation Utility Function**

   The interpolation utility function is generated by interpolating a tabulated function which is provided as two arrays. The first array corresponds to an ordered sequence of the various total risk levels of the portfolio and is denote by $x[0.., n-1]$ (with $x[0] < x[1] < ... < x[n-1]$). The first term of the second array corresponds to the expected return for the total risk $x[0]$. The second term of the second array corresponds to the expected return for the total risk $x[1]$. The third term is defined in a similar fashion and so on. This provides n coordinate points or equivalently a tabulated function which we can interpolate in order to provide a unique utility function which expresses the investors risk-reward profile.

The relationship between the Interpolation and Polynomial methods of setting the Utility Function

Note that the two means of setting the utility function namely the interpolation and polynomial procedures are closely related. Moreover we are able to roughly map between these two means of representing the investors utility function.

If the interpolation points are known at (x_i, y_i), for $i = 0, 1, ..., n-1$ then we can construct the utility function given as a polynomial of order n, by solving the following n polynomial expressions which will allow us to deduce to values of the coefficients of the polynomial which takes the same values at the interpolation points:

$$p(x_i) = coef[0] + (coef[1] * x_i) + \ldots + (coef[n-1] * x_i^{n-1})$$

where $i = 0, \ldots, n$ and $coef[i]$ are the coefficients of the utility function given in polynomial form. Alternatively, if we are given a polynomial $p(x) = y$, which represents the utility function of the investor then we are able to read off the interpolation points at $x_i : i = 0, \ldots, n-1$, by which the utility function can be defined in accordance with the interpolation approach. That is, the interpolation points (x_i, y_i), for $i = 0, 1, \ldots, n-1$, for some x_i, $i = 0, \ldots, n-1$, are given by:

$$p(x_i) = y_i$$

where $i = 0, \ldots, n-1$.

The above procedure illustrates that there is a close relationship between the polynomial and interpolation ways of defining the utility function. However care should be taken to point out that though they are closely related they are not equivalent. The reason for this is that the interpolation method uses cubic spline interpolation in order to construct the utility function from the interpolation points. The polynomial method in general uses an n degree polynomial in order to define the utility function. Therefore, except in the case of the polynomial method using a cubic polynomial these two approaches can not represent a utility curve which is identical for all points. However, in practice the above procedure will result in a polynomial and interpolation representation which agrees on the interpolation points and is generally qualitatively and quantitatively very close for non-interpolation points.

3.4.6 Discovering the Investors risk - return profile

One final and in fact rather essential practical issue is how best to go about discovering the investors utility function which can naturally only be done by obtaining information from the investor themselves. However, information concerning the investors risk profile in a usable quantitative form is often difficult to obtain because the investor has no natural way in order to quantize his attitude to risk. Moreover, the nature (i.e. granularity) of the information concerning the investors risk - reward profile you are able to obtain will determine the form of the Markowitz Theory you are able to apply. For example, if you are only able to obtain either the investors maximum risk or a lower bound of the level of the expected return desired then you will only be able to obtain the portfolio on the Efficient Frontier which corresponds to the upper bound on the expected return (in the case of knowledge of the maximum risk) or a lower bound on the risk (in the case of knowledge of the lower bound of the expected return).

Here we suggest a few approaches which should assist in your attempt to discover an investors risk profile and translate this understanding into a usable quantitative form. Often the discovery of the investors risk profile will be closely related to the aims, objectives, financial situation and motivation of the investor. By understanding the investment driving forces better you will be able to better tailor an investment portfolio which suits there needs. Below we offer three scenarios and approaches which allow the investors risk profile

to be discovered sufficiently well that a unique optimal portfolio can be chosen from the continuum of portfolios on the efficient frontier. The approaches include:

1. **Investors attitude to differing loses:** What probability of a 5%, 10%, 15%, 20%, 30% loss can the investor live with? From such information you will be able to estimate the investors utility function if you assume that the distribution of asset returns is in accordance with a given distribution (such as the LogNormal distribution). The procedure required in order to estimate the investor utility function is to fit the distribution as closely as possible to the answers given. Since to distribution has been we are able to read off the values of the standard deviation (i.e. risk) for different values of the expected return (i.e. the utility function).

2. **Learning of the investors objectives:** Say an investor wishes to obtain a given sum at a future date, that is, his utility is very much focused at achieving a certain finance goal. Such instances come about with retirement planning where a given sum is desired at retirement and we wish to achieve this sum by taking to lowest risk in order to achieve this return.

   In this instance you just need the investor to inform you of the return required. Once this is known you can select the portfolio from the efficient frontier given for this level of expected return. That is, the portfolio which can be constructed from the available assets which has the desired expected return with the lowest risk.

3. **Understanding the Investors Obligations:** An investors has a sum for which he wish to obtain the maximum return but also at the same time can not afford to lose more than a given amount. Such instances could occur within a large corporation which wishes to invest surplus cash within the market but cannot obtain loses of a certain level on this cash if it is to retain its present credit rating. In such instances the level of which the lose which the investor wishes to avoid at all possible costs should be stated. Then assuming the distribution of returns follows a given distribution (such as the LogNormal distribution) you will be able to state the change of such a loss occurring for a given portfolio on the efficient frontier.

The type of procedure performed above in order to ascertain the investors risk profile is in general referred to as calibration. Often the success of the application of a given model from the theory of quantitative finance will come down the success with which we are able to calibrate the parameters on which it depends.

3.4.7 Examples of selecting the Optimal Portfolio from the Investors Utility Function

Within this section we include a number of examples which illustrate how the optimal portfolio can be selected from the Efficient Frontier with the use of the investors utility function.

Cross Below Efficient Frontier

This is the most common example which you will encounter. The range of the risk of the utility function lies within a fairly narrow range implying that the investor is focused on controlling the risk of the portfolio and hence is not prepared to significantly increase the level of risk for significantly higher expected returns.

Note that if on the utility function the value of the risk is strictly positive when the expected return is zero. Then the utility function will always determine at least one optimal portfolio if there exists a value of the expected return such that the corresponding value of the risk of the Efficient Frontier is greater than the corresponding value of the risk of the investors utility function.

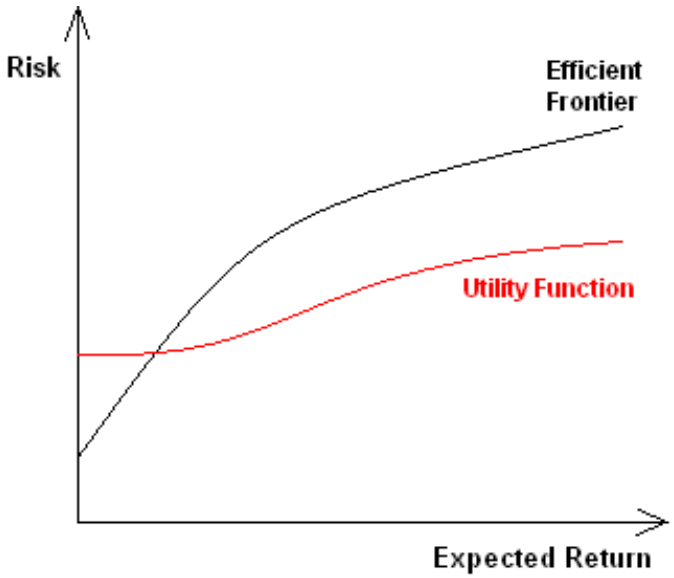


Fig: Typical Utility function shape.

Concave Utility Function

This example is fairly typical with the concave (i.e increasing positive gradient) Utility function usually resulting in an optimal portfolio with a relatively high value of the expected return. The reason being that the for relatively high values of the expected return the investor is prepared to take on proportionally larger amounts of risk in order to achieve a given increase in the expected return. Therefore, this utility functions shape would lead us to believe that the investor places paramount importance in obtaining a high level of expected return.

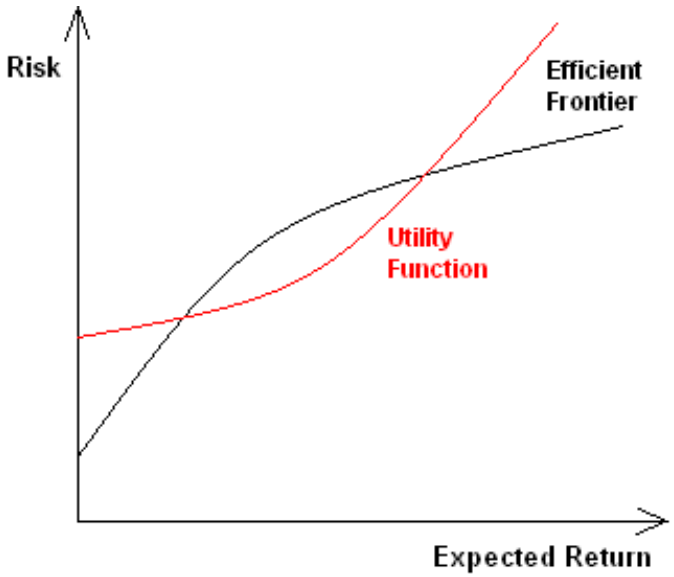


Fig: Concave Utility function resulting in a high expected return.

Cross Above Efficient Frontier

In this case the fact that the utility function for zero risk take of positive value of the expected return implies that the investor is in theory content to hold a cash portfolio and will only invest in risky assets if they match his risk-reward expectations. In order to meet these expectation the utility function must become equal to or greater than the Efficient Frontier for some value of the expected return.

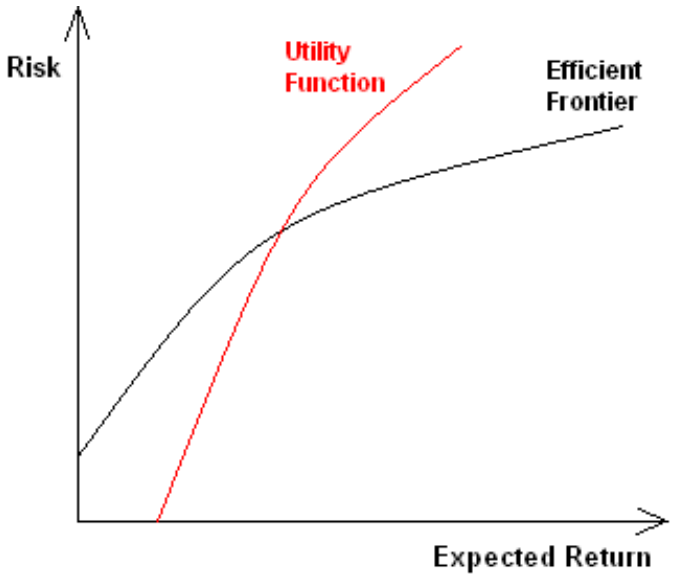


Fig: Possibility of holding cash.

Utility Function as a function of Risk

The following example was included to illustrate how the methods provided within the Portfolio.SolveFrontier class allow the use of utility functions which are functions of risk (rather than the expected return). What this means in practice is that you are able to investigate the investors risk-reward profile by asking the required expected return for a level of risk. If the investors provides this information for a range of values of the risk then you will be able to construct the (risk) utility function. We use the term '(risk) utility function' because by collection the investors profile in this form we are only certain that the result utility function will be a function in risk. That is, given a value of the risk over the range for which the Utility function is given there exists a unique corresponding value of the expected return. However, if we select a value of the expected return there may not exist a corresponding unique value of the risk of the utility function constructed by this means.

In the following diagram we provide a example of a (risk) utility function).

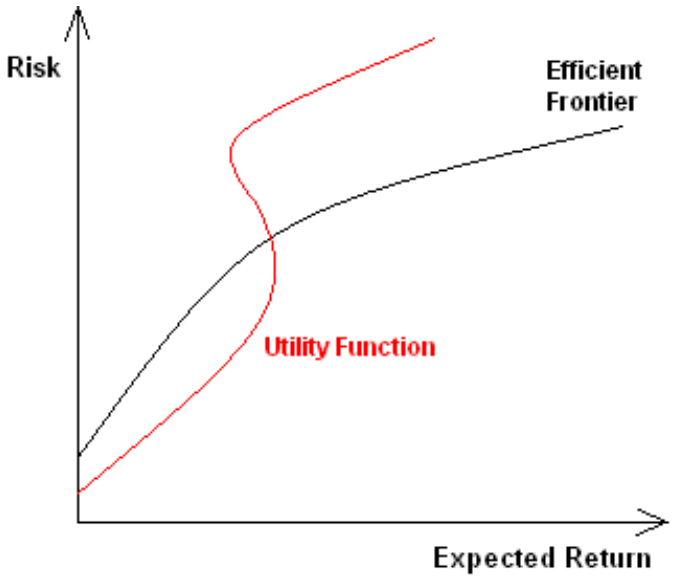


Fig: (risk) Utility Function

3.5 Capital Asset Pricing Model (CAPM)

3.5.1 Overview

The CAPM allows the investors portfolio to be constructed from risky assets and 'cash' holdings or borrowings which are either lent or borrow from the market at some prevailing rate. These additional options of allowing the portfolio to either borrow or lend cash at a prevailing market rate accurately reflects the possibilities available and practices used with the management of portfolios. That is, when a managed portfolio has excess capital it will typical be lent to the market at the prevailing rate (which is typically around LIBOR).

Conversely, if the portfolio wishes to increase its positions (and hence expected return) above the capital available within the fund then it will often borrow money at approximately the prevailing market rate (which is typically around LIBOR).

Remark The CAPM simplifies the situation in that it does not take into effect the credit worthiness of the fund when it wishes to borrow cash from the market. However, the assumptions are not unreasonable if we assume that the fund does not become very leveraged, since the fund could in principle at least arrange Repo agreements which would allow it to borrow additional capital at close to the prevailing market rate.

Comparison of CAPM with Markowitz Model

The Capital Asset Pricing Model (CAPM) is an extension of the Markowitz Model. That is, within the construction of the CAPM we assume all the assumption of the Markowitz Model (see 3.1.1) but in addition the investor has the following two options when constructing a portfolio:

1. **Lend excess capital at the Market Rate:** The investor may lend money at the prevailing market rate. This constitutes the ability to hold within the portfolio a risk free asset which will provide the prevailing return available on cash. In practice, such assets are often referred to as a money market accounts and will yield a return in the region of LIBOR.

2. **Borrow capital at the Market Rate:** The investor may borrow money from the market at the prevailing market rate in order to invest within (risky) assets. This will increase the expected return of the original capital base and also increase its risk. In practice, the rate at which money can be lent from the market by a fund will be in the region of LIBOR (at least if the fund structure the loan as a REPO type agreement).

Remark Note that the rate of which money can be borrowed or lend from or to the market is the same.

The similarity of the assumptions of the two models are reflected in there development. In particular, the Efficient Frontier in the sense of Markowitz Theory also plays in central role within the CAPM. In the following section we will explain why the optimal portfolio which in the Markowitz Theory consisted of a locus (i.e. curve) have been transformed to lie of a line in the case of CAPM.

3.5.2 Nature of the Capital Asset Pricing Model (CAPM)

The introduction of the risk free asset, namely cash along with the risky assets transforms the 'curve' on which the optimal portfolios found with respect to the Markowitz Theory into a straight line, known as the Capital Market Line (CML) on which the optimal portfolios with respect to the CAPM lie. The reason for this is that in all cases in order to obtain the optimal portfolio with respect to the CAPM, that is the portfolio which offers

a given return for the minimal risk. You will need to construct a portfolio which consists of a 'weighting' of the **'Market Portfolio'** (explained below) and either lend excess cash to the market or borrow cash from the market in order to purchase more (baskets) of the Market Portfolio.

The **Market Portfolio** is the portfolio on the Efficient Frontier in the sense of Markowitz Theory (see 3.4.2) which offers the greatest return per unit of risk. Expressing this analytically the **Market Portfolio** is the portfolio on the Efficient Frontier which maximizes:

$$\frac{\text{Expected Return of the Portfolio}}{\text{Total Risk of the Portfolio}}$$

The reason for this is that the investor whatever his risk/reward profile will seek the portfolio which offers the greatest return of a given level of risk, or the minimum risk for a given level of return. Clearly, at the level of the expected return of the Market Portfolio, the Market Portfolio itself is the optimal portfolio. However, if you wish to obtain a portfolio with a higher expected return then you should borrow cash (with zero risk) from the market in order to purchase more (baskets) of the 'Market Portfolio'. Since the Market Portfolio is the cheapest way in terms of risk to increase the return of the portfolio. Similarly, if you wish to obtain a portfolio with a lower expected return than the Market Portfolio. You will need to hold a sufficient weighting of the Market Portfolio in order that a portfolio consisting of a weighting of the Market Portfolio with the remaining capital held as (zero risk) cash holdings will ensure the desired level of the expected return. Again the rationale for this construction is that the cheapest way to increase the expected return of a portfolio invested in cash is to move some of the cash into the Market Portfolio which by definition offers the cheapest means with regard to risk in obtained a higher expected return.

Therefore, when selecting the optimal portfolio with respect to the CAPM one of three possible scenarios will occur:

1. **Borrow money** at the prevailing market rate in order to purchase further blocks of the Market Portfolio. That is, the portfolio which offers the greatest expected return per unit of risk. Since the cash borrowings are risk free and unlimited you can in principle hold an arbitrarily large amount of the Market Portfolio using the same (original) capital base. Allowing the portfolio to be geared to a level where are arbitrarily high value of the expected return can be obtained.

2. **Lend Money** at the prevailing rate if you do not require the level of expected return which is offered by investing all the available capital in the Market Portfolio. That is, the portfolio with the highest expected return per unit of risk. The rationale being that even if you require a lower level of expected return a weighted portfolio between the cash with zero risk (and a positive return) and the most efficient means to purchasing additional expected return (i.e. the Market Portfolio) is going to be most efficient means of obtaining the desired level of the expected return.

3. **No cash holdings or borrowing:** if the expected return (or risk) you require is the same as the expected return (or risk) of the Market Portfolio.

The line within the risk/reward plain which replaces the Efficients Frontier of Markowitz Theory is known as the **Capital Market Line (CML)**. Where each point on the CML corresponds to an optimal portfolio (i.e. minimum risk for a given value of the expected return) with respect to the CAPM.

3.5.3 Applying the CAPM

Since the optimal portfolios with regard to the CAPM all lie on the CML within applications we will nearly always proceed along the following lines:

1. Construction of the Efficient Frontier
2. Evaluation of the Market Portfolio
3. Constructing the CML
4. Selecting a Portfolio from the CML

Construction of the Efficient Frontier

The implementation of the construction of the Efficient Frontier for the CAPM follows along exactly the same lines as the implementation provided for the Markowitz Theory and we refer the reader to the earlier section 3.4.2 for general advice concerning the setting of constraints and other issues effecting the construction of the Efficient Frontier. Though the techniques used within the construction does following along similar lines within the implementation of the CAPM we have only exposed the functionality necessary for the application of CAPM. In particular, you will find the followings two methods within the Portfolio.CapitalMarket class:

1. Portfolio.CapitalMarket.setConstraints - Sets of the constraints (lower and upper bounds) on the assets within the collection of (risky) assets from which the market portfolio and the other portfolios on the Efficient Frontier will be evaluated.
2. Portfolio.CapitalMarket.calculateEfficientFrontier - Evaluates a finite number of points on the Efficient Frontier of the collection of assets from which the market portfolio can be constructed and store them within a private field. It is necessary to evaluate the Efficient Frontier because by definition the Market Portfolio is the portfolio on the Efficient Frontier which offers the maximum expected return per unit of risk.

Remark In order to make a more detailed study of the Efficient Frontier we refer you to the methods contained within the Markowitz class.

By applying these two methods the Efficient Frontier is constructed as follows:

1. Evaluated the Efficient Frontier at a finite number of points. That is, find the portfolio which exhibits the lowest risk for a given expected return from the collection of asset available. Note that the weights of the assets may be subject to constraints.

2. Interpolate about these points using cubic spline (or some other method) in order to construct the Efficient Frontier. Since the Efficient Frontier by nature is 'smooth' using cubic spline interpolation to construct the Efficient Frontier will not introduce significant errors.

The points on the Efficient Frontier are portfolios constructed from the set of assets considered which exhibit the lowest risk for a given expected return. These portfolio are characterized by the following three characteristics:

1. Expected Return - The expected return of the portfolio which is estimated from the historical returns of the assets within the portfolio.

2. Total Risk - The total risk of the portfolio which is estimated from the historical returns of the assets within the portfolio.

3. Asset Weights - the weights of the collection of assets from which the portfolio can be constructed.

It is important to point out that the Efficient Frontier in monotonically increasing function in risk and expected return. This means that if we are given a value of the expected return then there will correspond a unique portfolio on the Efficient Frontier with a given total risk. Conversely, if we are given the total risk of the portfolio then there will exist a unique portfolio on the Efficient Frontier with a corresponding value its expected return.

Evaluation of the Market Portfolio

The Efficient Frontier is the collection of portfolios constructed from the given set of available assets. These portfolios have the lowest risk for a given value of the expected return with the possibility of constraints on the weights of the assets.

Once the Efficient Frontier has been constructed are next aim within the application of the CAPM is to find which portfolio on the Efficient Frontier which is the Market Portfolio. That is, the portfolio on the Efficient Frontier which maximizes:

$$\frac{\text{Expected Return of the Portfolio} - \text{risk free rate}}{\text{Total Risk of the Portfolio}}$$

The Market Portfolio is selected from the Efficient Frontier using the method Portfolio.CapitalMarket.marketPortfolio, which will return the (possibly constrained) weights of the Market Portfolio which has the highest expected return over the risk free rate per unit of risk.

Uniqueness of the Market Portfolio

Here we discuss the 'uniqueness' of the market portfolio subject to some remarks concerning the given collection of assets from which the portfolios of the Efficient Frontier are constructed. These remarks though assisting in a deeper understanding of CAPM (and Markowitz Theory) can be safely skipped for those solely interested in the application of the CAPM (or Markowitz Theory).

The uniqueness of the Market Portfolio depends in the nature of the collection of assets from which the Efficient Frontier is constructed. Clearly the Efficient Frontier is convex and hence if we assume that there are two (or more) distinct Market Portfolios then it follows that one Market Portfolio is a leveraged version of the other in the following sense. The covariance between the two (distinct) market portfolios must be one since otherwise by holding a weighting of both market portfolios and gaining from the effects of diversification we are able to construct a portfolio with a higher expected return per unit of risk. Therefore, the covariance between any distinct Market Portfolios must be one.

In fact, if there are two distinct market portfolios with differing values of the risk and expected return then there exists an continuous range of Market Portfolios (i.e. an infinite number) in risk and expected return within the intervals of the risk and expected return of the two given Market Portfolios. The reason for this is that we can form a weighting of the two Market portfolios which forms another Market Portfolio (i.e. it has the same expected return per unit of risk) which has any value of the expected return or risk within there respective intervals. This possibility becomes clear if we consider the portfolio $R = aP + (a-1)Q$, where a lies in the interval $[0, 1]$, and where P, Q are the two given Market Portfolios. Now as a varies from 0 to 1 the portfolio R's expected return and risk will vary over all values within the respective domains but the expected return per unit of risk will remain fixed. That is, they will all be Market Portfolios.

Constructing the CML

The Capital Market Line (CML) is simple the line within the risk - expected return plain which passes through the Market Portfolio and cash. Below we derive formulae for the construction, expected return and risk of an arbitrary portfolio on the CML. We also refer the reader to the diagram of the CML at 3.5.4 which is probably to quickest and easiest means to obtain a better understanding on the CML and its construction.

Expressing this analytically: If we construct a portfolio from a (positive or negative) weighting of cash (C) where the prevailing market rate on lending and borrowing cash is r, and the Market Portfolio (M) for a given collection of (possibly constrained) assets denoted by M with an expected return of $E(M)$ and risk of $R(M)$, then all portfolios (P) on the CML will take the following form:

$$P = x_{cash}C + x_{market}M$$

where $x_{cash} + x_{market} = 1$; with the real numbers x_{cash}, x_{market} denoting the weighting of cash and the Market Portfolio within the constructed portfolio on the CML P.

The **expected return** $E(P)$, of the constructed portfolio P is given by:

$$E(P) = x_{cash}R + x_{market}E(M)$$

where as mentioned above R is the return on cash and $E(M)$ is the expected return of the Market Portfolio, and where again $x_{cash} + x_{market} = 1$; with the real numbers x_{cash}, x_{market} denoting the weighting of cash and the Market Portfolio within the constructed portfolio on the CML P.

Similarly, the **risk** σ_P, of the portfolio P is given by:

$$\sigma_P = (1 - x_{cash})\sigma_m = x_{market}\sigma_m$$

where σ_m is the risk of the Market Portfolio, and where again $x_{cash} + x_{market} = 1$; with the real numbers x_{cash}, x_{market} denoting the weighting of cash and the Market Portfolio within the constructed portfolio on the CML P.

Remark With the Portfolio.CapitalMarket class we provide methods by which the expected return and risk of the Market Portfolio can be constructed. Moreover, we also offer within this class means by which the weighting of the Market Portfolio of an arbitrary portfolio on the CML can be found. In fact, in turns out (see 3.5.3) that if we know any one of the expected return, risk or Market Portfolio weighting of a portfolio on the CML then we are able to deduce the other two properties.

Selecting a Portfolio from the CML

The Capital Market Line (CML) is a collection of portfolios which can be constructed from the available (risky) assets with the option of borrowing or lending (risk free) cash at the prevailing market rate. Since by default the amount of cash which can be borrowed or lend is not restricted the expected return of the resulting portfolios can be anything equal or greater than the return available from cash. Similarly, the risk from possible portfolios can (by default) be any positive number.

We allow within this Component the portfolios on the CML to be selected from knowledge any one of the following:

1. Expected Return
2. Total Risk
3. Weighting of the Market Portfolio

This in fact allows for the greatest generality and moreover which even property is used in the identification the other two properties can be deduced as described below.

Completeness of the Methods: riskCML, weightCML, returnCML, weight2Risk of the CapitalMarket class

The portfolio on the CML can be selected when one of the following three properties is known:

1. Total Risk of the optimal portfolio on the CML.
2. Expected Return of the optimal portfolio on the CML.
3. Weighting of the market portfolio on the CML.

Then using the above mentioned methods we are able to evaluate the other (two) quantities from the three properties which are listed above. For example, if the expected return of the portfolio is known then the weight of the market portfolio can be evaluated using weightCML and the risk can be evaluated using riskCML. If on the other hand the total risk of the portfolio is known then the corresponding expected return of the portfolio can be evaluated by returnCML, and then using this deduced value we are able to evaluated the weight of the market portfolio using weightCML. For completeness we include the method weight2Risk which evaluates the risk of a portfolio on the CML when the weight of the market portfolio within the CML portfolio is known. From knowledge of the risk of the CML portfolio we able to evaluate the corresponding value of the expected return using the method returnCML.

Therefore, using the three '...CML' methods along with 'weight2Risk', which ever one of: total risk, expected return or weight of market portfolio, is used in order to select the optimal portfolio from the CML we are able to deduce the other two quantitative properties.

3.5.4 Summary of the CAPM

We summarize the CAPM with the following diagram which contains the Efficient Frontier (the black curve) which is first evaluated, then the Market Portfolio (denoted by a red square) is found on the Efficient Frontier. After which when 'cash' (denoted by red square on Expected return axis) we are able to construct the CML (red line) by drawing a line through cash and the Market Portfolio.

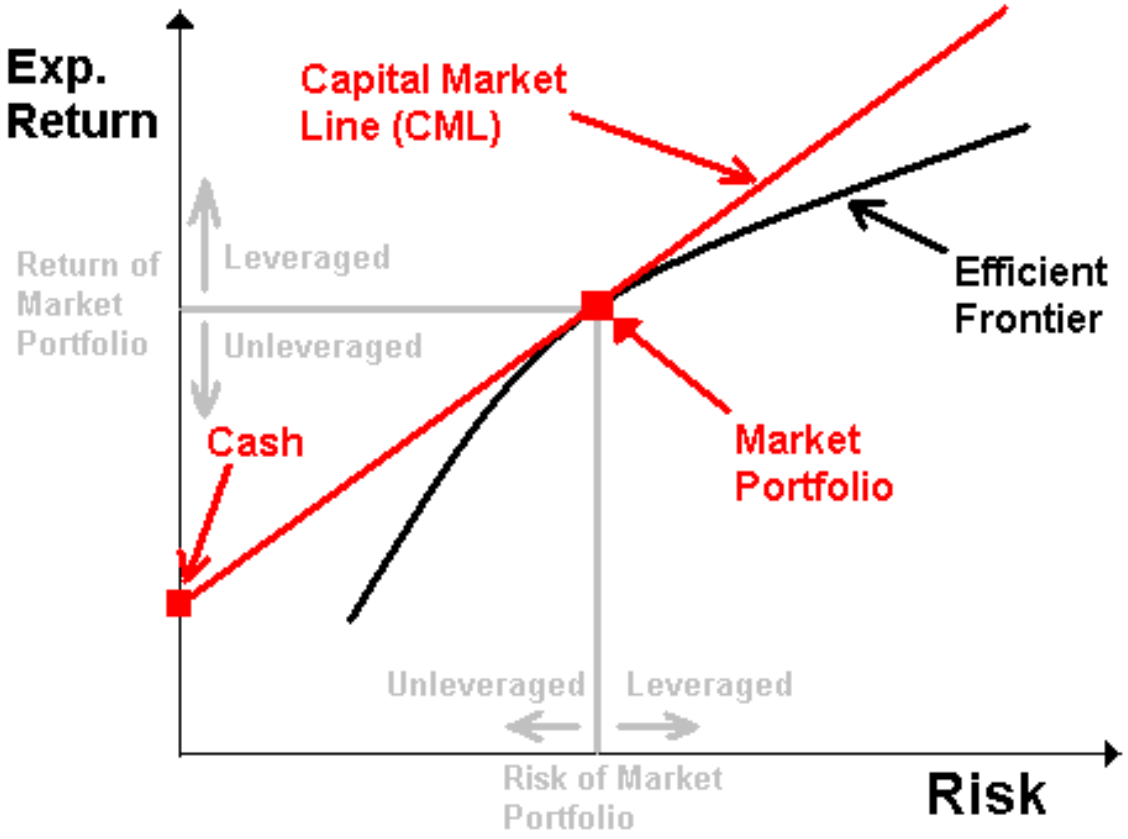


Fig: The CML, Market Portfolio, and the Efficient Frontier

The gray perpendicular lines on the diagram running off from the Market Portfolio indicates the values of the risk and expected return of the Market Portfolio. Any portfolios to the left on the risk axis to the Market Portfolio on the CML are portfolio which borrow cash from the market. Similarly, any portfolio to the right on the risk axis on the CML are portfolio which lend cash to the market. In a similar fashion, any portfolios on the CML with an expected return greater than the Market Portfolio (i.e. above the gray horizontal line) borrow cash from the market, and any with a lower expected return (i.e. below the gray) lend cash to the market.

3.5.5 Putting constraints on the level of borrowing and lending

At present we are able to place constraints on the weights of the (risky) assets from which the portfolios on the Efficient Frontier used within the Markowitz Theory and CAPM can be evaluated. This allows the risky asset held within the portfolios on the Efficient Frontier from which in the case of Markowitz Theory the optimal portfolio will be selected. Hence, within the framework on Markowitz Theory we have a means by which to place constraints on all the assets of the optimal portfolio.

In the case of the CAPM, a portfolio can also contain either cash holdings (i.e. free risk assets yielding the prevailing market rate on cash) or borrowings (i.e. money borrowed from the market at the prevailing market rate). Are aim here is to rationalize and detail

the issues related to why and how constraints can also be placed on the cash holdings or borrowings.

Rational of Borrowing Limit

Within the mandate of a fund, the manager will often be set a maximum amount of gearing which the fund can take. If the manager ever wishes to go above this level then they will need to obtain authorization from the funds board.

From the point of view of risk management and hence credit rating and regulation. If there is maximum level of gearing then there is a worst case scenario otherwise the worst case scenario since the gearing is unlimited is unlimited losses.

Rational for a Cash Limit

The existence of a cash limit forces the fund to invest within the market which it addresses. The primary motivation for introduction such a limit are as follows:

1. **Role of a Fund Manager:** A portfolio manager to paid to invest money by selecting assets within a given sector or market. After all if the money is just deposited within a bank account the fund manager will still collect the management fee without reflecting his mandate. The mandate explicitly said invest in... If the investor wants to be out of the market then they can take money out of the fund.

2. **Reflect fund description/mandate:** An equity fund should not be a money market fund. I.e. Having limit like no more the 40% in cash at any time is reasonable. If this is not possible then capital should be returned to investors for similar reasons as within any other business.[4]

Mechanics of controlling the Cash/Borrowing Level

Within the CAPM the optimal portfolios which are constructed from a combination of the Market Portfolios with the possibility of cash holdings or borrowings lie on the CML. If there is a pre-defined limit on the level of cash holdings or borrowings then these limits will translate into a limit on the weighting the Market Portfolio within the portfolio of the CML. From limits on the weighting we will be able to imply the limits on the risk, or expected returns of the portfolio on the CML which satisfies the cash and borrowing limits since as shown above if one of the following three: weighting, risk, expected return; are known then the other two can be deduced.

[4]For further discussion on the effect of retaining earners and compounding we refer the invested reader to; The Essays of Warren Buffet, Lessons for Investors and Managers, edited by L.Cunningham.

Implemented as an Example

We provide within this package a client example which illustrates how constraints on the level of cash lending or borrowing can be used. The client is entitled:

CapitalMarketClient_CashConstraints

This example illustrate how constraints may be placed onto the levels of the cash held or borrowed to the market by an optimal portfolio constructed in accordance within the Capital Asset Pricing Model. We show how the resulting continuous range of the values of the expected return and total risk can then be deduced which identify which optimal portfolios can be selected when the cash level constraints are observed.

In particular we consider the following problem:

> We know that the market portfolio of a collection of assets has an expected return of 10 percent per year and a risk of 20 percent per year, we also know that cash can be borrow or lend from the market at a prevailing rate of 5 percent a year. According to the CAPM if the portfolio manager can only leverage his portfolio by 20 percent, and must also never hold more than 30 percent of the funds capital in cash then what is the range of total risk and expected return of the (optimal) portfolios which the fund manager can hold?

3.6 Performance Evaluation

As we discussed earlier since there is a relation between the risk and the expected return from an asset (CAPM) an assessment of a portfolios performance based only on the absolute return is misleading and unfair. Therefore, we describe standard methods which are used to given a risk adjusted performance measure.

Within the US mutual fund market there are more than 12,000 funds which apply a variety of investment styles and have a different mix of investment assets. Broadly speaking US mutual funds can be split into the following categories growth funds, balanced funds, income funds, government bonds funds, junk bond funds and international funds. According the CAPM the investors would expect a greater return from a growth fund which invests in stocks than a government bond fund because the growth fund will exhibit more volatility. The amount of "out performance" of the growth fund over the bond fund which the investor would expect is the topic of this section.

We offer various methodologies which are used for the evaluation of portfolio performance in the broad sense. These measures are widely used and offer a fast and efficient way in order to make informed comparisons between portfolio performance taking into account risk, return, asset class mix, yield and market conditions.

Within our components we implement the following methods:

- `totalReturn` - calculates the total return of the portfolio over a given period taking into account any disbursements and dividends payments
- `geometricMeanReturn` - the geometric mean of the return over a number of periods. Say the return is quoted for a 2 year period then by taking the number of periods as 2 the method will return the equivalent return over a 1 year period
- `sharpesRatio` - we implement Sharpe's ratio which takes into account both the return and risk when accessing a portfolios performance
- `treynorsMeasure` - Treynors performance measure which takes into account the systematic risk (or beta) and the average return when assessing the overall performance

3.6.1 Comparing the Sharpe and Treynor Performance Measures

The Sharpe portfolio performance measure is based on the capital market line (CML) and total risk, which makes it more suitable for evaluating portfolios rather than individual assets. On the other hand, Treynor performance measure is based on the capital asset pricing model (CAPM) and are more flexible because by using systematic risk (beta) it can be used to evaluate the performance of both portfolios and individual assets. Both performance measures tend to rank a group of diversified portfolios similarly.

3.7 Further and Supplementary Reading

3.7.1 Supplementary Reading

- **H. Markowitz** Portfolio Selection, Journal of Finance, 1952, p77-91

  This fundamental paper establishes the basis of the Markowitz theory and shows that portfolios dominate individual assets from a risk and return standpoint. Further it shows how to construct optimal portfolios with ideal risk - return characteristics.

3.7.2 Further Reading

- **H. Markowitz** Portfolio Selection, Efficient Diversification of Investments, Basil Blackwell

  Expands on the classic 1952 article listed above detailing the modern treatment and development of the theory.

- **W. Sharp** A Simplified Model for Portfolio Analysis, Management Science, Vol. 9, p27-93

Chapter 4

Programmer's Guide for Microsoft® Office

This chapter describes the steps to take in order to integrate WebCab Portfolio with most Office documents, such as Excel worksheets, and Access documents. The integration is achieved both through VBA (Visual Basic for Application) code and features specific to the Office Application you are using to write your documents. The information in this chapter applies equally to Microsoft Office 2000, XP, and 2003 Applications and documents.

4.1 Developing with VBA from Office

This section describes how to write a VBA client for a business class documented in the API Reference for this product. The steps below are the same across all Microsoft Office Applications (Word, Excel, Access etc.) and involve using the Visual Basic Editor and writing several lines of code:

1. Open the Visual Basic Editor
2. Add a Code Module
3. Declare a Subroutine
4. Add a Reference to This Product
5. Declare a Class Instance Variable
6. Create a Class Instance
7. Call a Class Method
8. Display the Method Result
9. Run the Subroutine

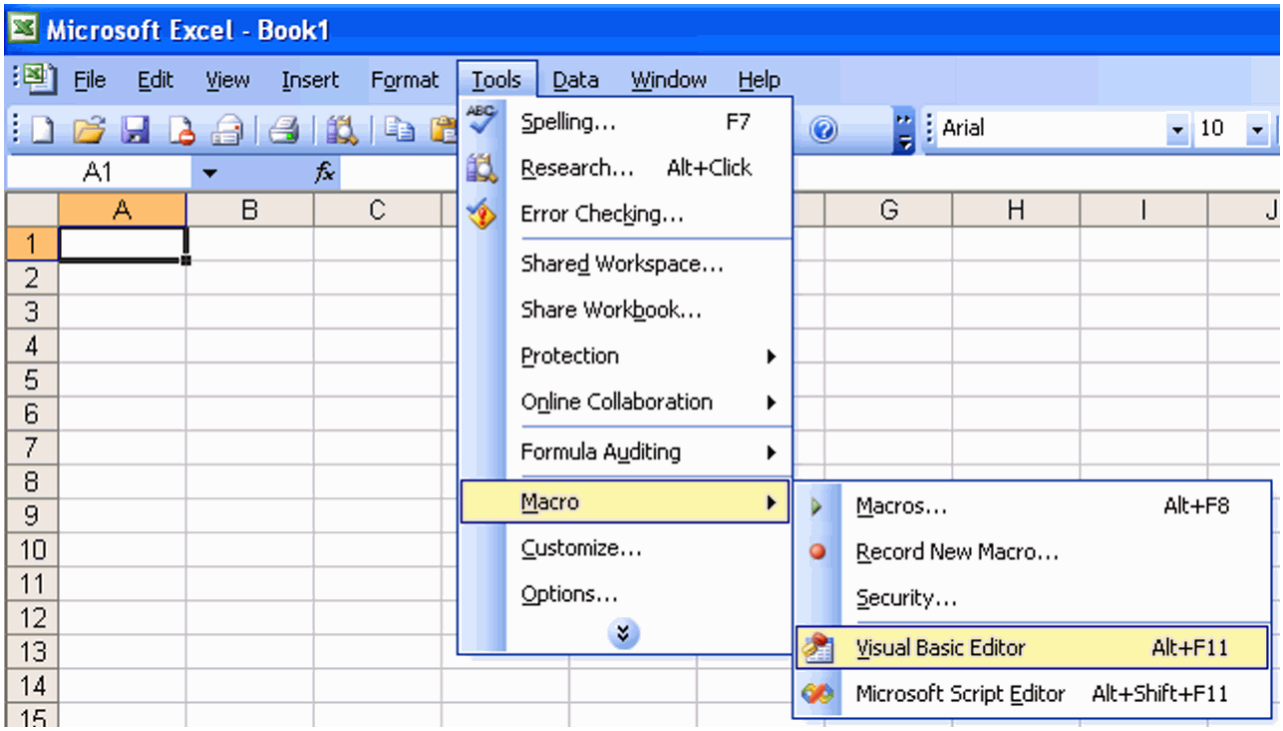


Fig. 4.1: Starting the Visual Basic Editor in Excel

4.1.1 Open the Visual Basic Editor

In order to add VBA code to your Office document, you will first need to open the Visual Basic Editor, by going to the Tools | Macro | Visual Basic Editor menu, as shown in figure 4.1.

4.1.2 Add a Code Module

In the upper-left corner of the Visual Basic Editor window, you can find all Office objects associated with your document. Right click any of the objects and select Insert | Module, as shown in Fig. 4.2 in order to add a new VBA module to your project. A code window will appear, where you will be writing the code that will enable you to use this product's functionality.

4.1.3 Declare a Subroutine

The first step in writing the code, which makes use of the functionality provided by this product is to declare a subroutine, which can then be run directly from your Office Application. To declare a subroutine, use the `Sub` and `End Sub` keywords and the name you wish to assign to this subroutine. For example, in order to declare a subroutine named *run*, you would write the code displayed in Fig. 4.3.

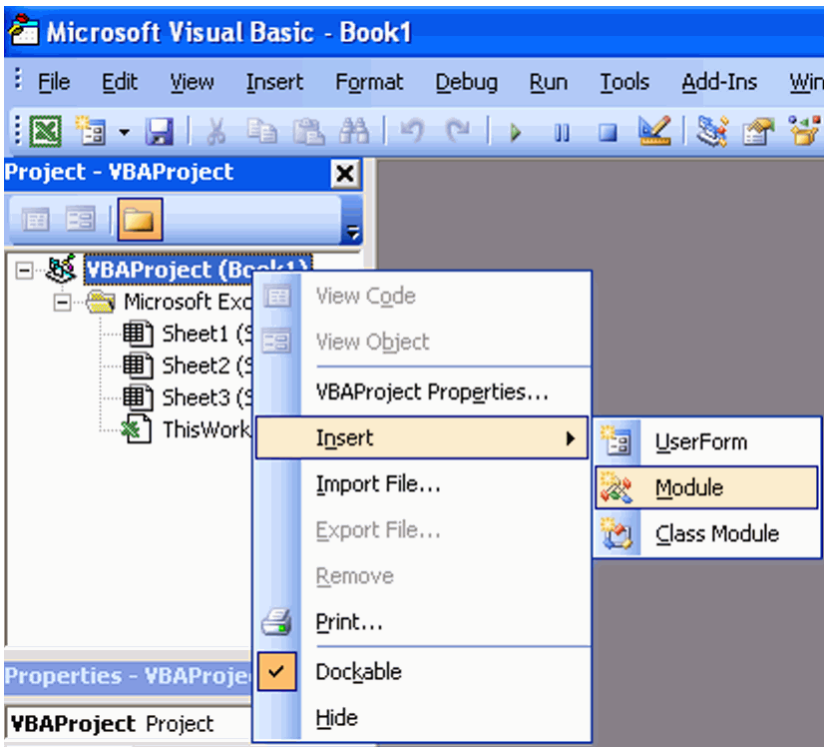


Fig. 4.2: Adding a module to a VBAProject

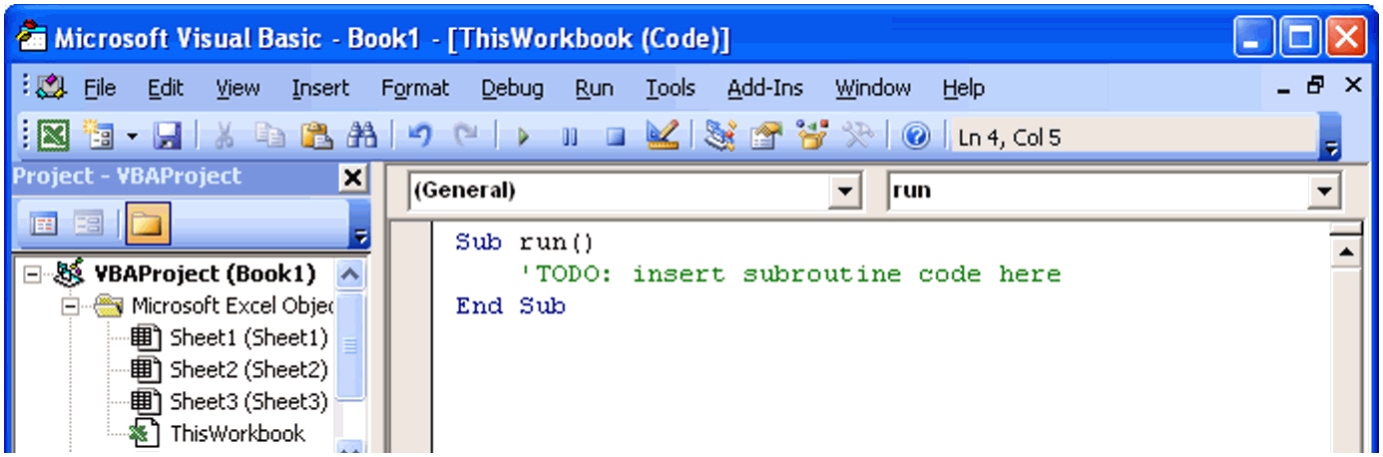


Fig. 4.3: Declaring a subroutine named *run* in VBA

4.1.4 Add a Reference to This Product

Adding a reference to this product is required only once for every Office document that uses functionality provided by this product. This step will enable VBA code auto-completion for all business clases and methods that belong to this product, saving you time typing and browsing the API Reference. Also, it will speed up all calls to this product's methods, increasing the overall performance of your project.

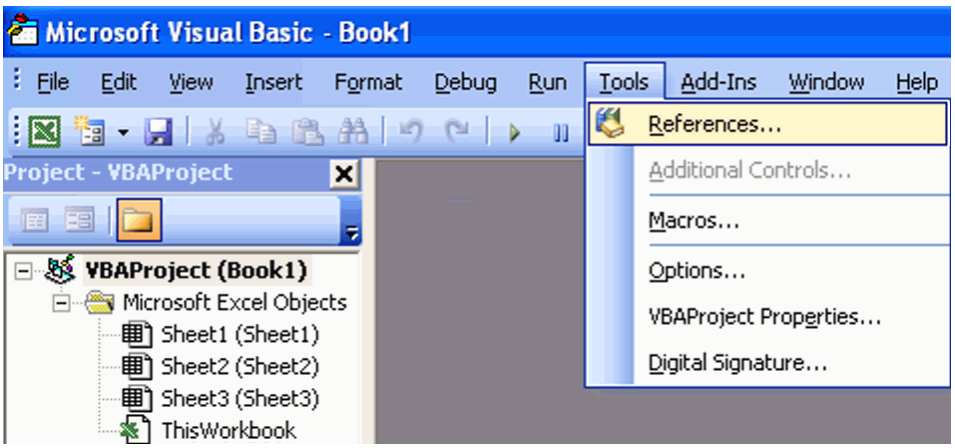


Fig. 4.4: Opening the "References" window in the Visual Basic Editor for Office

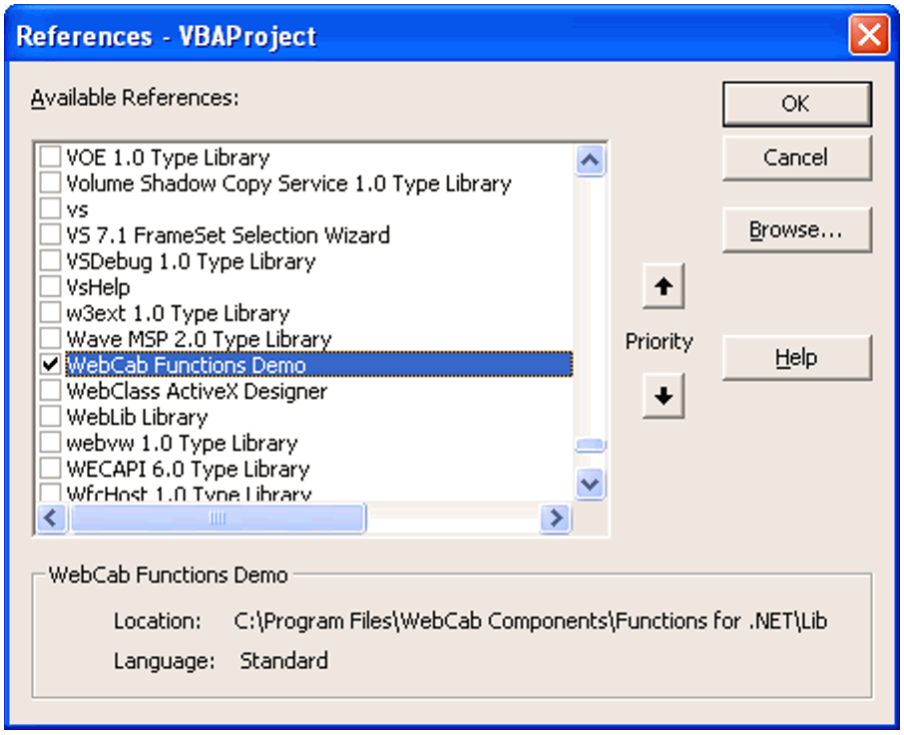


Fig. 4.5: Selecting a reference to a WebCab product, here 'Functions'

To add a reference to this product, go to the Tools | References... menu as shown in Fig. 4.4 in order to open the "References" dialog window. Scroll down and select the entry named *WebCab Portfolio Demo* from the list of available references in the dialog window and then click OK. Figure 4.5 shows how to add a reference to *WebCab Functions Demo*.

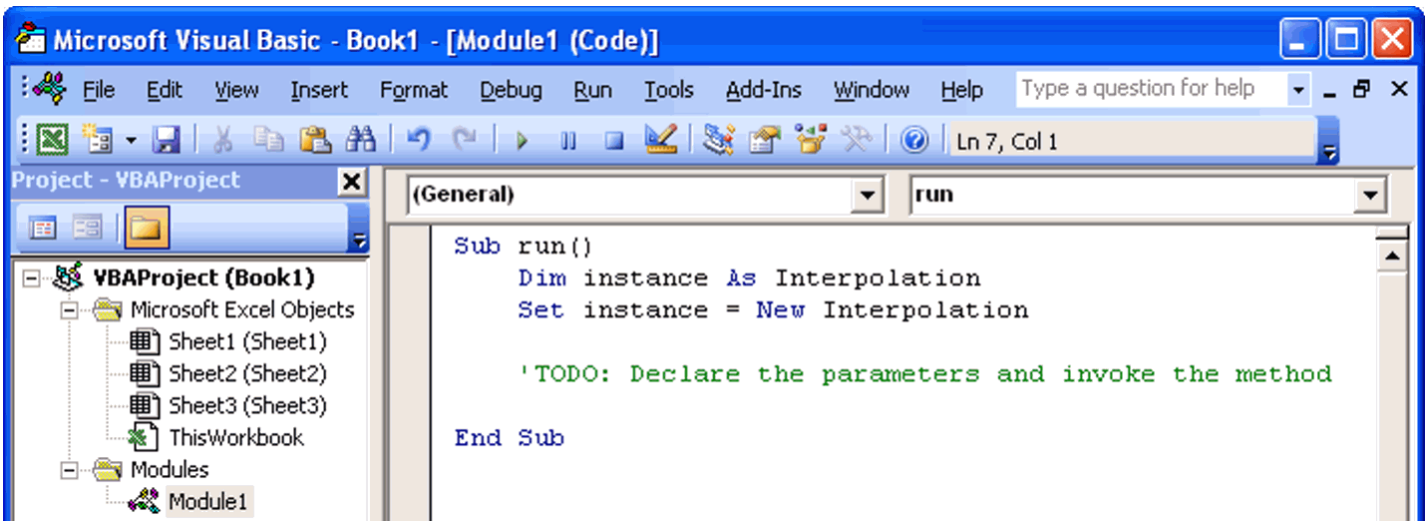


Fig. 4.6: Creating a new instance of a class named "Interpolation"

4.1.5 Declare a Class Instance Variable

The API Reference for this product describes all business classes, their methods and properties, and offers advice on how to use each of them. You can browse the API Reference from the Start Menu at Programs | WebCab Components | Portfolio for .NET | COM | API Reference. In order to call methods of a business class listed in the API Reference, you need to declare a variable to hold a reference to its instance.

Write a `Dim` statement to declare a variable of the same type as the business class that you wish to make calls to. For example, if you wish to call methods belonging to a business class named "Interpolation" and declare a variable of this type named `instance`, you would write the following:

```
Dim instance As Interpolation
```

4.1.6 Create a Class Instance

In order to create an instance of a business class listed in the API Reference, you will have to write code that connects to its corresponding COM server[1].

Use the `Set` keyword in order to assign the class instance to the variable you have declared in the previous step. The `Set` keyword is required in class instance assignments, as omitting this keyword would result in a run-time error. The reference assigned to this instance is created by writing the `New` keyword, followed by the name of the same business class you used to declare the variable.

[1] "COM server" is a generic term used to describe a COM interface, which provides functionality to VB and VBA applications.

For example, assume you wish to create an instance of a business class named Interpolation, which is located in the WebCab Functions for .NET product. Figure 4.6 shows how to create an instance of this class and how to assign it to the `instance` variable declared in the step above.

4.1.7 Call a Class Method

In order to call a method belonging to a business class for which you have created an instance, you can use the name of the method as listed in the API Reference and append it to the name of the variable that holds the class instance, separating it with a period. For example, in order to call a method named *MyMethod*, you could write the following line of code:

```
result = instance.MyMethod (parameter-values)
```

where *parameter-values* are the ordered values of the parameters separated by a comma. The actual type of these parameters and that of the *results* variable depends on the signature (i.e. the return type and parameter types) of the *MyMethod* method.

Remark For a more detailed Visual Basic example of how to invoke a method, see section 4.1.10.

4.1.8 Display the Method Result

To see what the result of the method call was, you can display it inside a window by calling the *MsgBox* function, as shown below:

```
MsgBox "The result of the method call was " & result
```

This line of code will display a small dialog window containing the result of the method call.

4.1.9 Run the Subroutine

After having finished writing the subroutine code as described in the steps below, you can run the subroutine. There are several ways to run a subroutine, two of which are described below:

1. **Type F5 in the Visual Basic Editor**
   You can run your subroutine by placing the editing cursor within its body (i.e. between the `Sub` and `End Sub` keywords) and pressing the F5 key. This will instantly execute the subroutine, bringing up the result window.

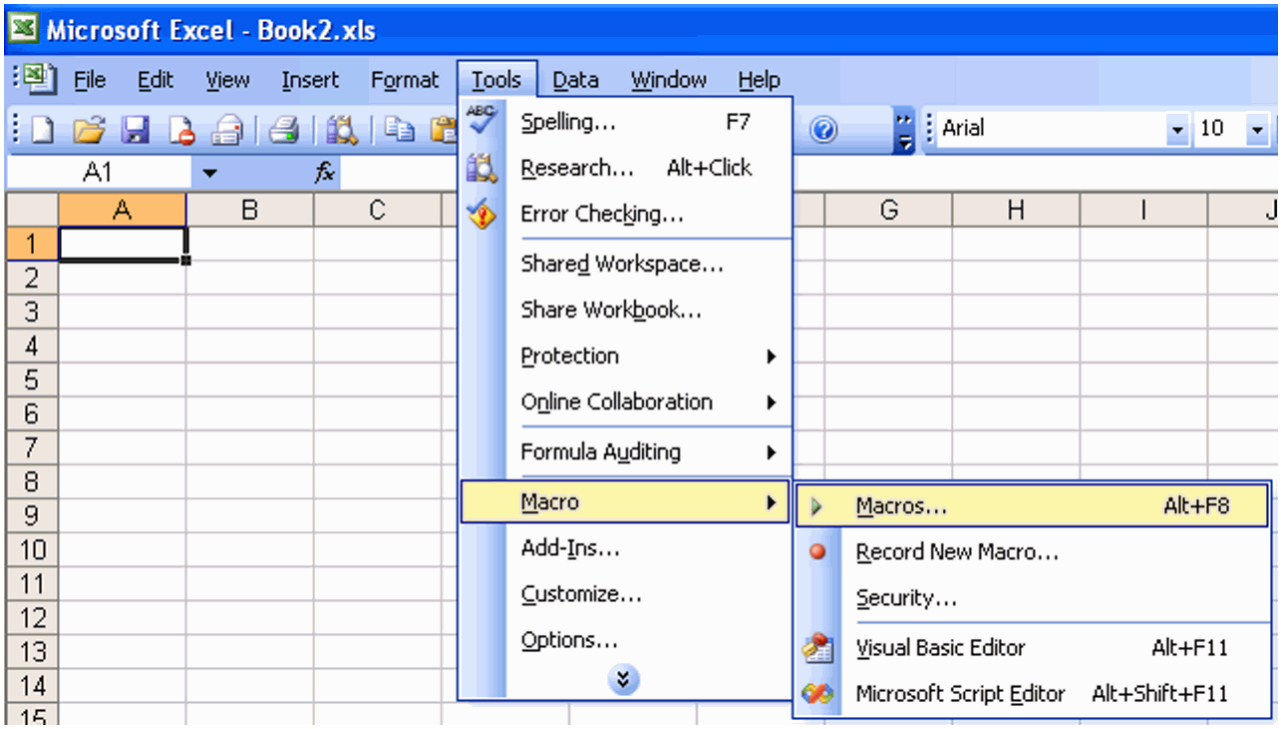


Fig. 4.7: Opening the "Macro" window from the menu in Excel.

2. **Run the Soubroutine as a Macro**
   You can run the subroutine even after having closed the Visual Basic Editor, directly from the Office Application you are using. Go to the Tools | Macro | Macros... menu, as shown in Fig. 4.7 to open the "Macro" dialog window. Select the macro, which has the same name as your subroutine and click Run (see Fig. 4.8).

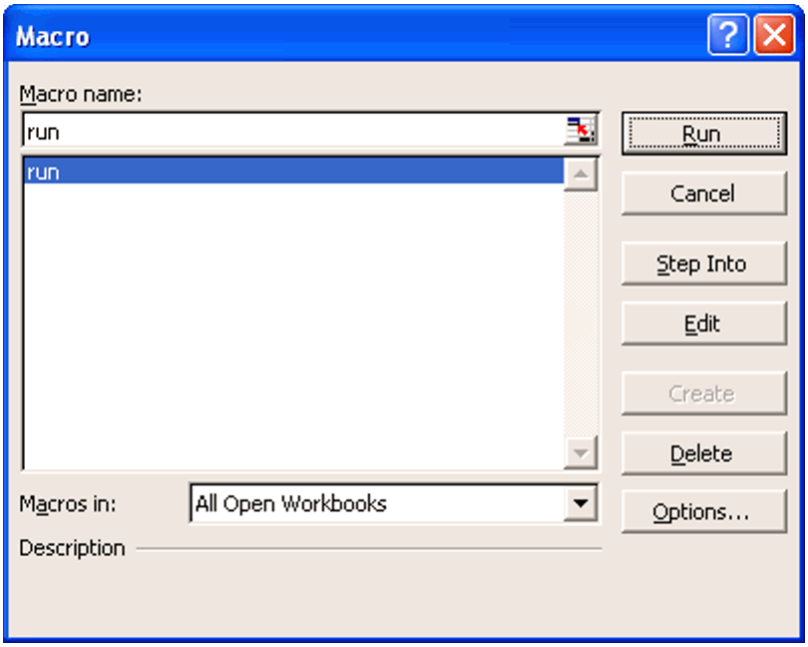


Fig. 4.8: Running a subroutine macro named *run* from Office.

```
' Declaring a subroutine named `run'
Sub run()
    Dim instance As Interpolation
    ' Creating an new `Interpolation' instance
    Set instance = New Interpolation

    ' Declaring the first parameter (a double array)
    Dim x(0 To 4) As Double
    x(0) = 1
    x(1) = 2
    x(2) = 3
    x(3) = 4
    x(4) = 5

    ' Declaring the second parameter (a double array)
    Dim y(0 To 4) As Double
    y(0) = 3
    y(1) = 2.4
    y(2) = 1.7
    y(3) = 1.4
    y(4) = 0.7

    ' Declaring the variable to hold the method result
    Dim result As Double

    ' Calling the `CubicSplinePointwise' method.  All other
    ' parameter values are being written here.  The result
    ' is stored in the `result' variable.
    result = instance.CubicSplinePointwise(x, y, -0.6, -0.7, 1.2)

    ' Printing the result inside a dialog window
    MsgBox "The result of the method call was " & result
End Sub
```

Table 4.1: Generic Office VBA Example

4.1.10 A Generic VBA Example for Office

In this section we provide the complete VBA source code for a subroutine named *run*, which makes a call to a method in a business class and prints the result of the method call inside a window. The same steps mentioned above are being followed in this example.

The code in table 4.1 calls a method named CubicSplinePointwise, belonging to the Interpolation business class in the WebCab Functions product. This method takes five parameters, of which the first two are one-dimensional arrays and the last three are double values. The

method returns a double value.

4.2 Integrating with Microsoft Excel

This section is dedicated exclusively to Microsoft Excel users, who wish to tightly and seamlessly integrate the functionality provided by this product directly into Excel. At the end of this section, you will have learned how to create your own user-defined functions, which you can call directly from your worksheets as formulas or from the Insert | Function... menu. The following steps describe completely how to achieve integration of this product's functionality within Excel:

1. Open the Visual Basic Editor
2. Add a Code Module
3. Declare a Function
4. Add a Reference to This Product
5. Declare a Class Instance Variable
6. Create a Class Instance
7. Call a Class Method
8. Store the Method Result as a Function Return Value
9. Insert the Function in your Worksheet

4.2.1 Open the Visual Basic Editor

First you will need to open the Visual Basic Editor, by going to the Tools | Macro | Visual Basic Editor menu, as shown in figure 4.1. The Visual Basic Editor will allow you to write the necessary VBA code that uses the functionality provide by this product.

4.2.2 Add a Code Module

In the upper-left corner of the Visual Basic Editor window, you can find all Microsoft Excel objects associated with your workbook. Right click any of the objects and select Insert | Module, as shown in Fig. 4.2 in order to add a new VBA module to the existing Visual Basic project. A code window will appear, where you will be writing the code that will enable you to use this product's functionality.

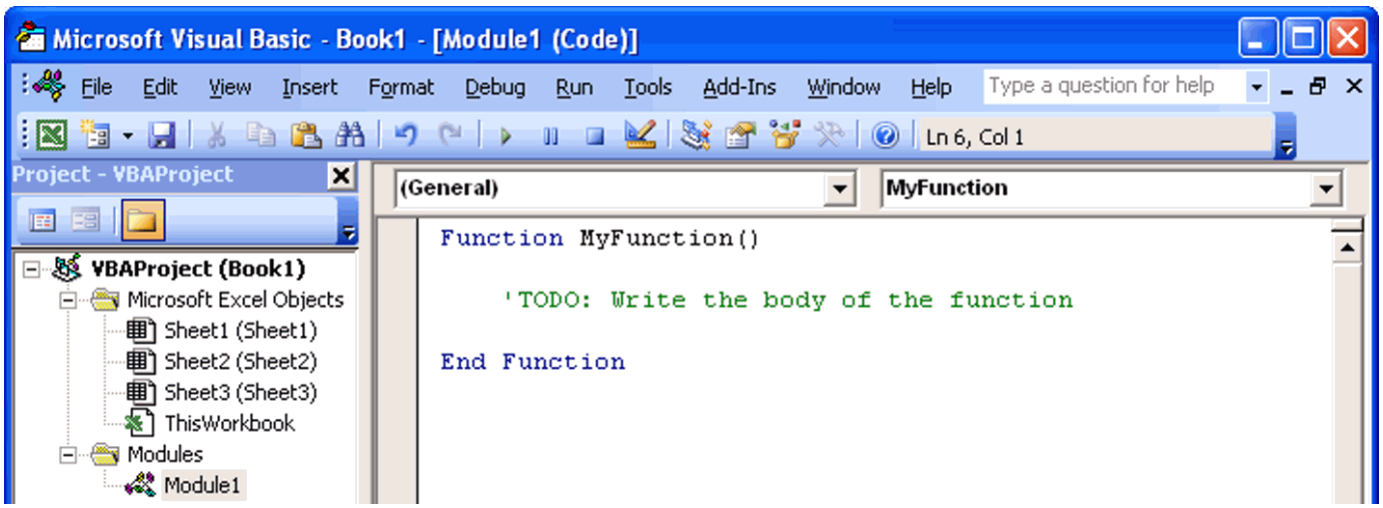


Fig. 4.9: Declaring a function named "MyFunction" in VBA

4.2.3 Declare a Function

In the source code window, declare a function using the `Function` and `End Function` keywords. You can pick any name you wish for this function, as you will be able to call it directly from a worksheet, like a regular Excel formula. You can change the name of the function at any later time, by simply editing the function declaration. Figure 4.9 shows how to declare a function named *MyFunction*.

4.2.4 Add a Reference to This Product

As described in section 4.1.4, in order to add a reference to this product, you will need to go to the Tools | References... menu as shown in Fig. 4.4. This will open the "References" dialog window. Scroll down and select the entry named *WebCab Portfolio Demo* from the list of available references in the dialog window and then click OK. Figure 4.5 shows how to add a reference to *WebCab Functions Demo*.

4.2.5 Declare a Class Instance Variable

The API Reference for this product details all business classes, their methods and properties, and gives advice on how to use each of them. You can browse the API Reference from the Start Menu at Programs | WebCab Components | Portfolio for .NET | COM | API Reference. In order to call methods of a business class listed in the API Reference, you need to create a variable to hold a reference to its instance.

Write a `Dim` statement to declare a variable of the same type as the business class that you wish to make calls to. For example, if you wish to call methods belonging to a business class named "Interpolation" and declare a variable of this type named `instance`, you would write the following:

```
Dim instance As Interpolation
```

4.2.6 Create a Class Instance

Use the `Set` keyword in order to assign the class instance to the variable you have declared in the previous step. The `Set` keyword is required in class instance assignments, as omitting this keyword would result in a run-time error. The reference assigned to this instance is created by writing the `New` keyword, followed by the name of the same business class you used to declare the variable.

For example, the Interpolation business class inside the *WebCab.COM.Math.Interpolation* namespace has the following full name: *WebCab.COM.Math.Interpolation.Interpolation*. In order to create an instance of this class and assign it to the variable we have declared in the previous step, we would write the following VBA code:

```
Set instance = New Interpolation
```

4.2.7 Call a Class Method

In order to call a method belonging to a business class for which you have created an instance, you can use the name of the method as listed in the API Reference and append it to the name of the variable that holds the class instance, separating it with a period. For example, in order to call a method named *MyMethod*, you would write the following line of code:

```
instance.MyMethod (parameter-values)
```

where *parameter-values* are the ordered values of the parameters separated by a comma. The actual type of these parameters and that of the *results* variable depends on the signature (i.e. the return type and parameter types) of the *MyMethod* method.

Remark For a more detailed Visual Basic example of how to invoke a method, see the next section, 4.2.8.

4.2.8 Store the Method Result as a Function Return Value

The result of the method call must be stored as the method's return value, by assigning the method's result to the the the name of the function itself, as shown below:

```
MyFunction = instance.MyMethod (parameter-values)
```

```
' Declaring a function named `MyFunction'
Function MyFunction()
    Dim instance As Interpolation
    ' Creating an instance of the `Interpolation' business class
    Set instance = New Interpolation

    ' Declaring the first parameter (a double array)
    Dim x(0 To 4) As Double
    x(0) = 1
    x(1) = 2
    x(2) = 3
    x(3) = 4
    x(4) = 5

    ' Declaring the second parameter (a double array)
    Dim y(0 To 4) As Double
    y(0) = 3
    y(1) = 2.4
    y(2) = 1.7
    y(3) = 1.4
    y(4) = 0.7

    ' Calling the `CubicSplinePointwise' method.  All other
    ' parameter values are being written here.  The result
    ' is stored as a return value for this function.
    MyFunction = instance.CubicSplinePointwise(x, y, -0.6, -0.7, 1.2)
End Function
```

Table 4.2: The VBA code for a user-defined function in Excel

The complete source code of a function named *MyFunction* is shown in table 4.2. The example calls the same method as the generic VBA example in section 4.1.10.

4.2.9 Insert the Function in your Worksheet

Switch back to the worksheet window and open the "Insert Function" window from the Insert | Function... menu (see Fig. 4.10). From the dialog window, click the drop-down list that lists all the categories, and select the category named "User Defined", as shown in figure 4.11.

The name of your function will appear in the list of user-defined functions. Select it and click the OK button (Fig. 4.12). A window named "Function Arguments" will appear, asking for values for the arguments. In case your function takes no arguments, as in our example above, simply click OK, as shown in figure 4.13.

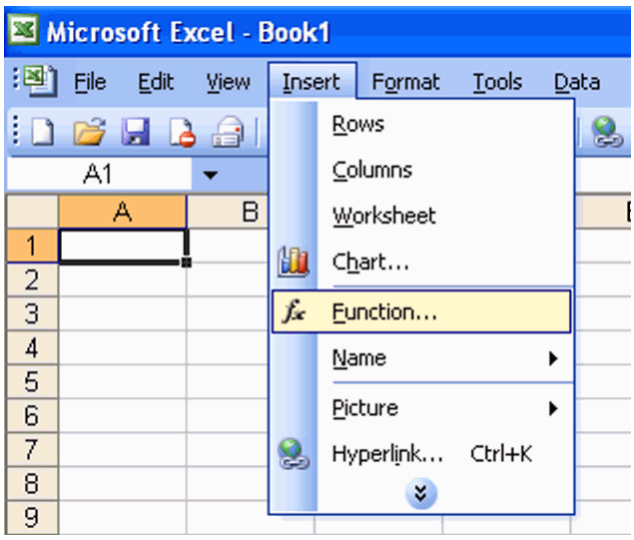


Fig. 4.10: How to open the "Insert Function" window in Excel

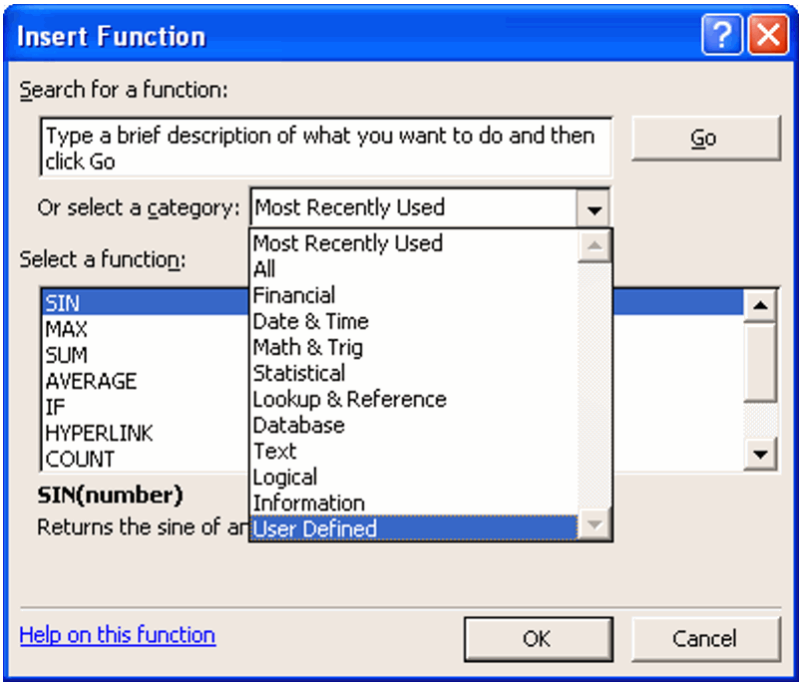


Fig. 4.11: Selecting the "User Defined" category

The current cell value will contain the value of a formula, which calls the "MyFunction" function with no parameters. The result of the formula, listed in the f_x formula bar, will be printed inside the cell, as shown in figure 4.14. You can change the formula directly from the formula bar, or by opening again the "Insert Function" window.

You can also add a reference to the formula directly from the worksheet, by typing the value of the formula directly into the cell, without opening the "Insert Function" window,

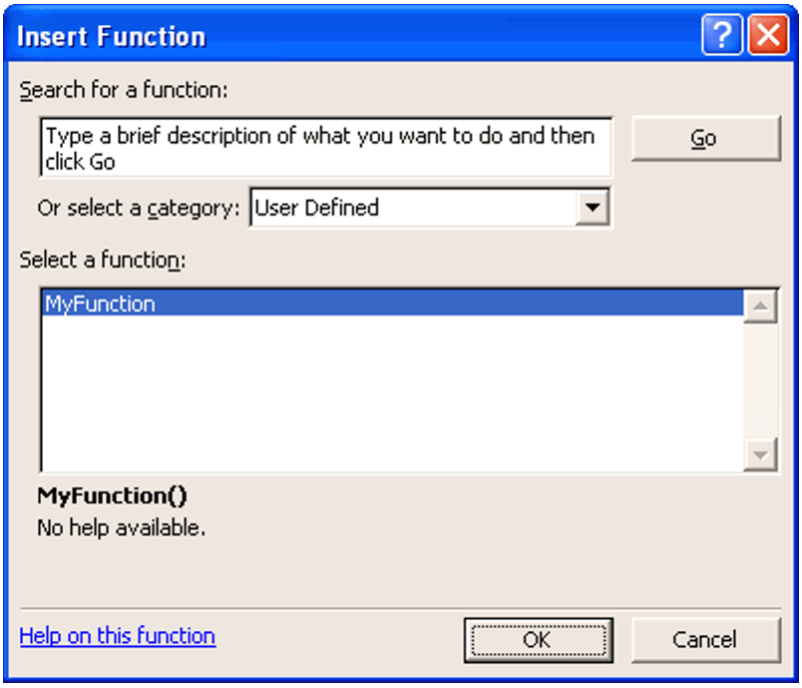


Fig. 4.12: Selecting the "MyFunction" user-defined function

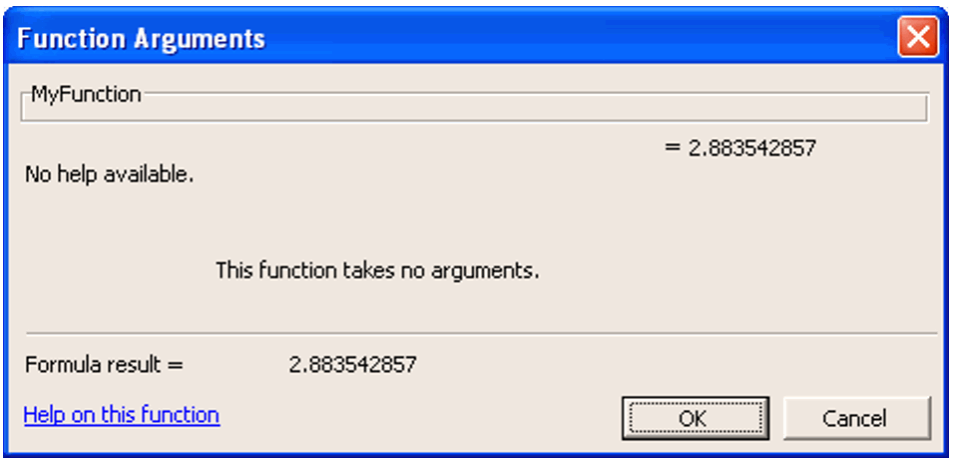


Fig. 4.13: Inserting the "MyFunction" user-defined function into the worksheet

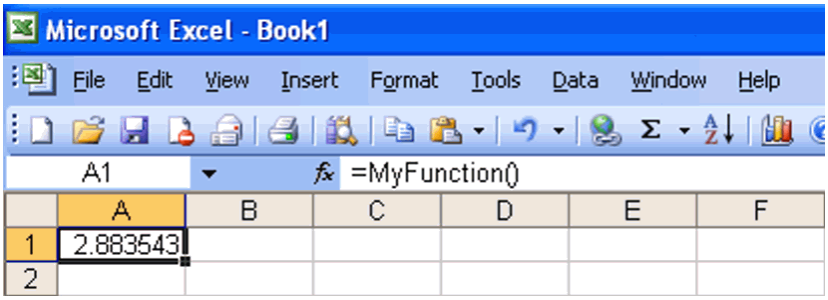


Fig. 4.14: The value returned by the "MyFunction" formula in an Excel worksheet

the same way you would call a standard Excel formula.

Chapter 5

Programmer's Guide for Visual Studio 6

This chapter is dedicated to Visual Studio 6 developers, programming with either or both of the Visual Basic 6 and Visual C++ 6 languages. Using WebCab Portfolio from these languages comes down to connecting to a COM server and making calls to it. The first section is for Visual Basic 6 developers, while the second section is for Visual C++ 6 developers.

5.1 Developing with Visual Basic 6

This section describes the steps required to use this product from VB6. These steps are always the same, irrespective of the type of project you are developing. In an example at the end of this section, we also provide a generic Visual Basic example of how to connect to our components from a "Standard EXE" Project.

Assuming you have already started a new project or opened an existing project, here are the steps required to connect and use a WebCab Portfolio COM server:

1. Add a Reference to This Product
2. Declare a Class Instance Variable
3. Create a Class Instance
4. Call a Class Method

5.1.1 Add a Reference to This Product

Adding a reference to this product is required only once for every VB Project that uses functionality provided by this product. This step will enable VB code auto-completion for all business clases and methods that belong to this product, saving you time typing and browsing the API Reference. Also, it will speed up all calls to this product's methods, increasing the overall performance of your project.

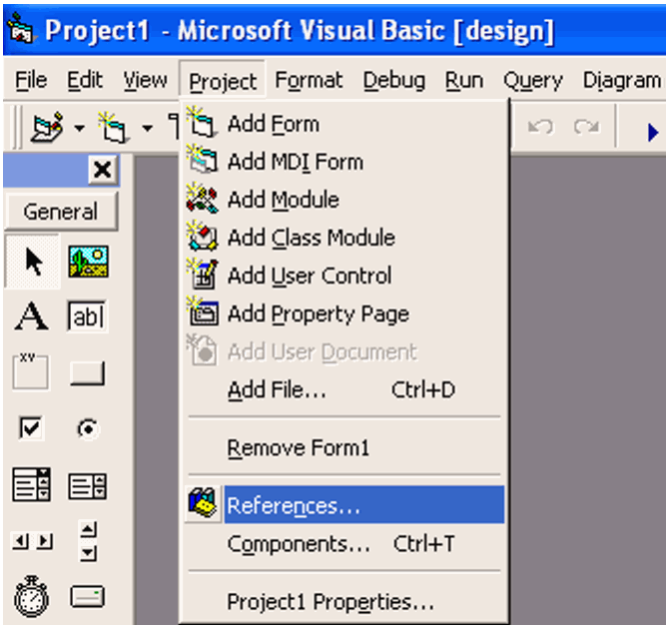


Fig. 5.1: Opening the "References" window in Visual Basic 6

To add a reference to this product, go to the Project | References... menu as shown in Fig. 5.1 in order to open the "References" dialog window. Scroll down and select the entry named *WebCab Portfolio Demo* from the list of available references in the dialog window and then click OK. Figure 5.2 shows how to add a reference to *WebCab Functions Demo*.

5.1.2 Declare a Class Instance Variable

The API Reference for this product details all business classes, their methods and properties, and gives advice on how to use each of them. You can browse the API Reference from the Start Menu at Programs | WebCab Components | Portfolio for .NET | COM | API Reference. In order to call methods of a business class listed in the API Reference, you need to create a variable to hold a reference to its instance.

Write a `Dim` statement to declare a variable of the same type as the business class that you wish to make calls to. For example, if you wish to call methods belonging to a business class named "Interpolation" and declare a variable of this type named `instance`, you would write the following:

```
Dim instance As Interpolation
```

To avoid name clashing with other classes that may be named the same as the business class you are instantiating, you can prefix the name of the business class with the name VB6 assigns to the COM reference to this product. Assuming the name of this reference

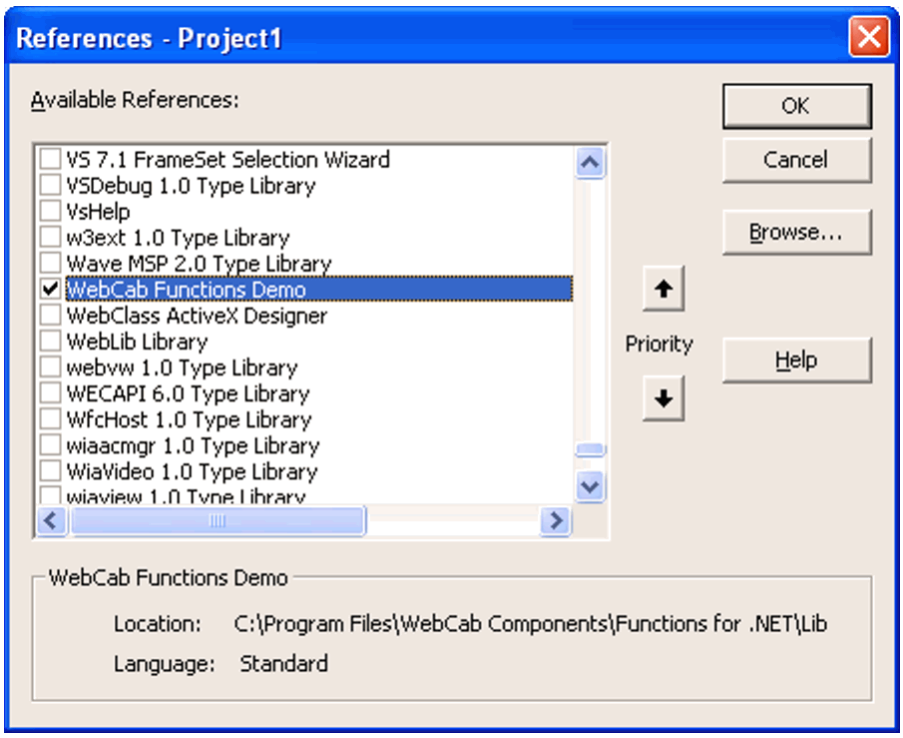


Fig. 5.2: Selecting a reference to a WebCab product, here *WebCab Functions*

is `WebCab_COM_FunctionsDemo`, the complete business class name would be referenced as follows:

```
Dim instance As WebCab_COM_FunctionsDemo.Interpolation
```

5.1.3 Create a Class Instance

In order to create an instance of a business class listed in the API Reference, you will have to write code that connects to its corresponding COM server. Use the `Set` keyword in order to assign the class instance to the variable you have declared in the previous step. The `Set` keyword is required in class instance assignments, as omitting this keyword would result in a run-time error. The reference assigned to this instance is created by writing the `New` keyword, followed by the name of the same business class you used to declare the variable.

For example, assume you wish to create an instance of a business class named Interpolation, which is located in the WebCab Functions for .NET product. The following line of code will create an instance of this class and assign it to the `instance` variable declared in the step above:

```
Set instance = New Interpolation
```

To avoid name clashing with other classes that may be named the same as the business class you are instantiating, you would use the complete name of the business class, as shown below:

```
Set instance = New WebCab_COM_FunctionsDemo.Interpolation
```

5.1.4 Call a Class Method

In order to call a method belonging to a business class for which you have created an instance, you can use the name of the method as listed in the API Reference and append it to the name of the variable that holds the class instance, the same way you would call a method or property in Visual Basic. For example, in order to call a method named *MyMethod*, you could write the following line of code:

```
result = instance.MyMethod (parameter-values)
```

where *parameter-values* are the ordered values of the parameters separated by a comma. The actual type of these parameters and that of the *results* variable depends on the signature (i.e. the return type and parameter types) of the *MyMethod* method.

Remark For a more detailed Visual Basic example of how to invoke a method see the following section, 5.2.

5.2 A specific Visual Basic Example

Within this section we provide an explicit example in Visual Basic 6 which calls the MarkowitzReturn which belong to the EasyOptimal business class of the Portfolio Component and returns the result within a message box. Though this example is written for a particular method from within the Portfolio Component by applying the principles out lined here you should be able to easily adopted this example to call any available method. The MarkowitzReturn method takes five parameters: two double variables,two one-dimensional arrays and one two-dimensional array. The return value is one-dimensional array containing the weights of the assets under consideration. You can use the code from the 5.2 table by copy-pasting inside any of your subroutines or functions. In our example, we placed the code inside the `Form_Load` subroutine, which is been triggered at the time the `Application Form` is first started.

Structure of the Example

This particular example perform the following steps:

- **Start Visual Basic Project** - Within our examples we use a Form project in which we illustrate the method call.
- **Declaring an instance of the class** - In order to use the methods from a given class you must first create an instance of that class.
- **Initializing the instance of the class**
- **Declaring the Parameters** - Create the types which will be used within the method call.
- **Declaring the variable to hold the returned method result**
- **Invoke the Method**
- **Print the result within a message box**

Remark: We assume that the Portfolio component has been registered and is available within the Visual Basic project. For further details on how this is performed please see the previous section.

Obtaining the Source code

We provide an implementation of this example with associated project files within the directory:

```
home/Libraries/Client/Optimization/VBExamples/EasyOptimal
```

where `home` refers to the directory where the product is installed which will generally be: `C:/Program Files/WebCab Components/Portfolio for .NET`

This folder can also be access via the START menu by selecting: `START > Programs > WebCab Components > Portfolio for .NET > .NET > Tools > Portfolio > VBExamples Custom Example`

Within this folder you will find the following files:

1. `EasyOptimal.frm` - The Visual Basic source code file.
2. `EasyOptimal.vbp` - The Visual Basic project file.
3. `EasyOptimal.vbw` - The Visual Basic Project Workplace

In order to run this example you will need to:

1. **Open VB Project file** - Double click on the .vbp file which will open the project within Visual Basic 6. We assume that you have Visual Basic 6 installed and that Visual Basic project files (i.e. .vbs file) are associated with this IDE.

2. **Add a WebCab Portfolio Reference** - Select from the VB6 menu `Project > Reference` which will bring up the 'References' dialog box. With the text area list please select the item 'WebCab Portfolio Demo' and then click 'OK'.

3. **Run the example** - After the Component has been registered you are able to run the examples by pressing F5.

```
Private Sub Form_Load()
    ' Here we demonstrate how the optimal portfolios can be constructed
    ' when the historical returns are known and the portfolio is determined
    ' by given the expected return and risk, in cases where the portfolio
    ' may or may not hold cash.  We also select the optimal portfolio in
    ' accordance with the investors utility function.

    ' Declaring a instance of the class EasyOptimal
    Dim instance colordarkblueAs EasyOptimal
    ' Instantiating the instance
    Set instance = colordarkblueNew EasyOptimal

    ' The value of the expected return for which the optimal portfolio
    ' is sort.
    Dim expectedReturns As Double
    expectedReturn = 101

    ' The lower bounds on the asset weights are.
    Dim lowerBounds(0 To 3) As Double
    lowerBounds(0) = 0.1
    lowerBounds(1) = 0.1
    lowerBounds(2) = 0.1
    lowerBounds(3) = 0.1

    ' The upper bounds on the asset weights are.
    Dim upperBounds(0 To 3) As Double
    upperBounds(0) = 0.5
    upperBounds(1) = 0.5
    upperBounds(2) = 0.5
    upperBounds(3) = 0.5

    ' Historical Values Used
    ' We are able to construct portfolios from four assets with the
    ' following historical returns over the last 4 months:
    ' 1st Asset:  102, 101, 99, 101
    ' 2nd Asset:  100, 103, 99, 101
    ' 3rd Asset:  100, 99, 103, 101
```

```
    ' 4th Asset:  101, 101, 102, 101
    Dim historicalReturns(0 To 3, 0 To 3) As Double

    historicalReturns(0, 0) = 102
    historicalReturns(0, 1) = 101
    historicalReturns(0, 2) = 99
    historicalReturns(0, 3) = 101
    historicalReturns(1, 0) = 100
    historicalReturns(1, 1) = 103
    historicalReturns(1, 2) = 99
    historicalReturns(1, 3) = 101
    historicalReturns(2, 0) = 100
    historicalReturns(2, 1) = 99
    historicalReturns(2, 2) = 103
    historicalReturns(2, 3) = 101
    historicalReturns(3, 0) = 101
    historicalReturns(3, 1) = 101
    historicalReturns(3, 2) = 102
    historicalReturns(3, 3) = 101

    ' The precision used but the algorithm.
    Dim precision As Double
    precision = 0.000000000001

    ' Here we construct the optimal portfolio which can be constructed from
    ' Equity Assets only where the weight of each asset lies within the
    ' range [0.1, 0.5],which has an expected return of 101.
    weights = instance.MarkowitzReturn(expectedReturn, lowerBounds,_
        upperBounds, historicalReturns, precision)

    ' These four lines just print our the result.
    MsgBox "The weights of the first asset in the optimal portfolio is:" _
        & weights(0)
    MsgBox "The weights of the second asset in the optimal portfolio is:" _
        & weights(1)
    MsgBox "The weights of the third asset in the optimal portfolio is:" _
        & weights(2)
    MsgBox "The weights of the fourth asset in the optimal portfolio is:" _
        & weights(3)
End Sub
```

5.2.A specific Visual Basic client example using EasyOptimal class

5.3 Developing with Visual C++ 6

Using this product from Visual C++ 6 involves two major steps: importing the Type Library for this product and passing parameters to the COM methods according to COM standards. All the other steps are rather straightforward and involve basic C++ code writing:

The steps are as follows:

1. Open a New or Existing Project
2. Add All COM Specific '`include`' Declarations
3. Import the Type Library for this Product
4. Call "`CoInitialize`"
5. Connect to a COM Server
6. Declare the Parameter Types and Values
7. Declare the Return Type
8. Call the Method
9. Call "`CoUninitialize`"

5.3.1 Open a New or Existing Project

You can start a new Visual C++ Project by going to the File | New... menu (see Fig. 5.3), which will bring up the "New" dialog window. From this window, choose the "Projects" tab and select the *Win32 Console Application*. Type in the name of your project in the Project name text box (for example, *Project1*) and click OK, as shown in Fig. 5.4.

In the next window, select "A simple application" (see Fig. 5.5), click the Finish button, and click OK in the next window as well. From the left hand side of the screen, switch to "FileView" and select the *.cpp* file with the same name as your project – as seen in Fig. 5.6. Note that the source code snippets in this section apply both to a new project and to one of your existing projects.

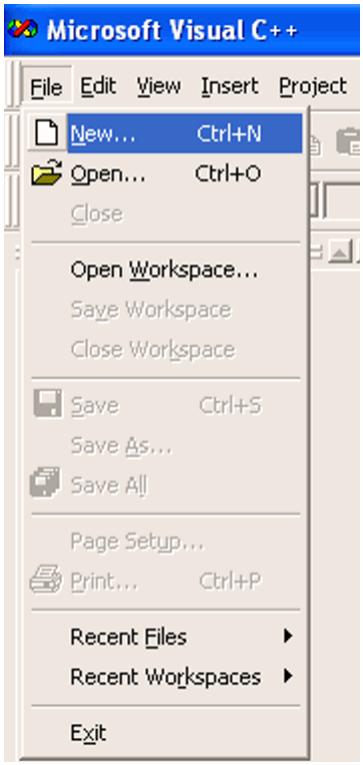


Fig. 5.3: Starting a new project

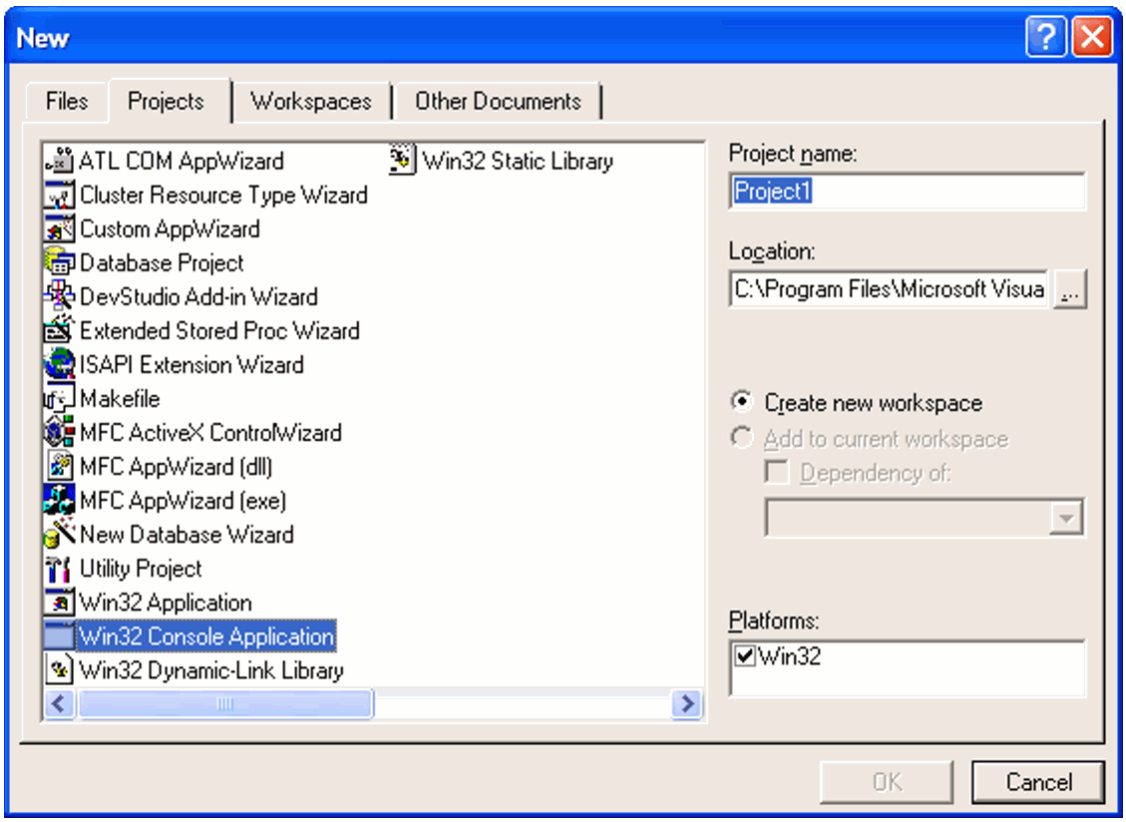


Fig. 5.4: Starting a Console Application named "Project1"

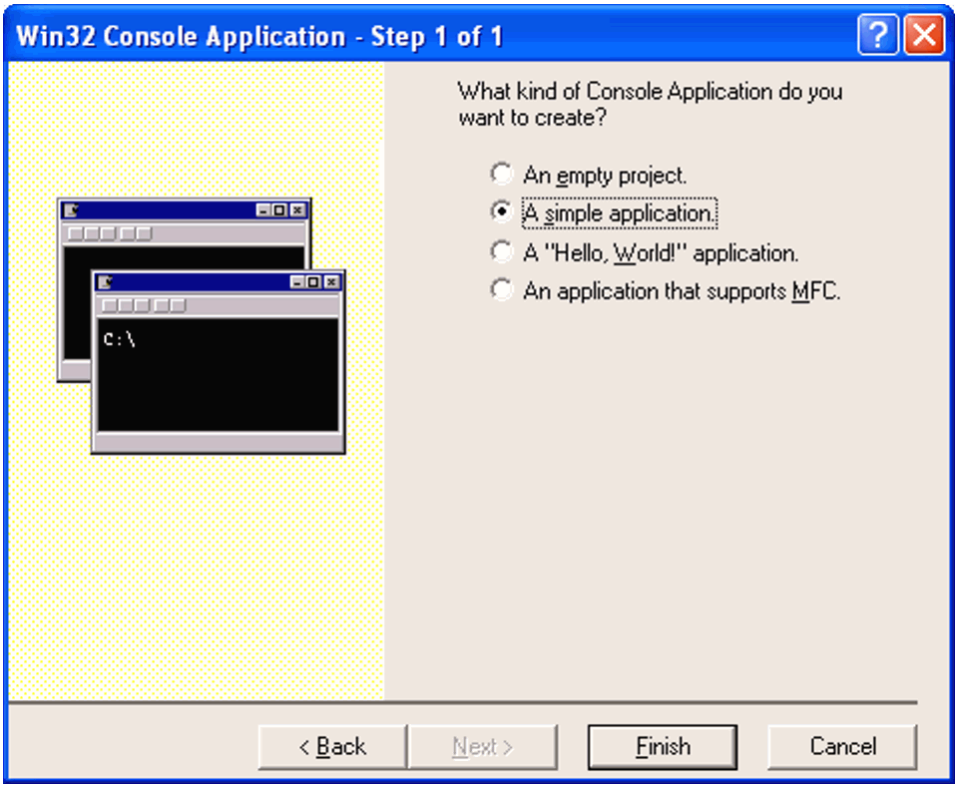


Fig. 5.5: Creating a simple Console Application

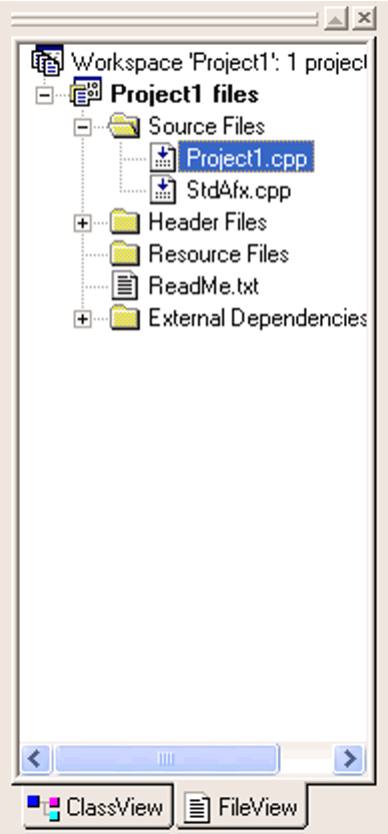


Fig. 5.6: Location of the main C++ source code file, *Project1.cpp*

5.3.2 Add All COM Specific 'include' Declarations

COM specific calls will require that the header file *objbase.h* is included within your project. This header offers functionality to enable C++ to interoperate with COM servers. To include this header file within your project you will need to add at the top of your main source code file the following line:

```
#include "objbase.h"
```

5.3.3 Call "CoInitialize"

In order to allow Visual C++ clients to connect to a COM server, you need to make a call to a function named CoInitialize, which belongs to the *objbase.h* header file, mentioned above. The call to the function must be made as shown below:

```
CoInitialize(NULL);
```

5.3.4 Import the Type Library for this Product

The name of all COM interfaces and their methods are described by a Type Library file with a *.tlb* extension. These methods are automatically exposed to your Visual C++ 6 client by the IDE, if you import this *.tlb* file into your project.

In order to import the Type Library for this product, add an `#import` declaration (right after the `#include` declarations in your main source code file) followed by the complete path to the COM *.tlb* file, which is located in the *COM Libraries* subdirectory of the instalation path for this product. The default installation path for users with Administrator privileges is *C:\Program Files\WebCab Components\Portfolio for .NET*, while the default installation path for a user without Administrator privileges is *C:\Documents and Settings***UserName***\Local Settings\Application Data\WebCab Components\Portfolio for .NET*.

If you add `no_namespace` at the end of the `#import` statement, the COM interfaces will be placed within the default namespace, so you can access them directly without prefixing them with the namespace name.

For example, if you installed with Administrator privileges, you can import the Type Library for this product by writing the following statement:

```
#import "C:\Program Files\WebCab Components\Portfolio for .NET\
\COM Libraries\WebCab.COM.PortfolioDemo.tlb" no_namespace
```

If you omit the `no_namespace` parameter, the COM methods will be exposed as part of a namespace called `WebCab_COM_PortfolioDemo`.

5.3.5 Connect to a COM Server

For every COM server described in the COM API Reference, the previous step defines a type with the same name prefixed by "`COM_`" and followed by "`Ptr`". Declare a variable of this type and invoke its method named *CreateInstance* with the GUID of the COM server itself as a parameter.

For example, in order to connect to the Interpolation COM server, part of the WebCab Functions for .NET product, you would write the following C++ code:

```
COM_InterpolationPtr instance = NULL
instance.CreateInstance(__uuidof(Interpolation));
```

The variable named "instance" will hold the connection to the "Interpolation" COM server in all subsequent calls.

5.3.6 Declare the Parameter Types and Values

Passing non-array parameters to the COM methods is as straightforward as passing parameters to a C++ method. You can declare a parameter to hold the value, pass the value directly, or pass the result of another method call.

When it comes to passing one-dimensional or two-dimensional arrays as parameters to COM methods, you will need to perform some additional COM specific steps. First of all, all COM arrays are typed as `SAFEARRAY`. The `SAFEARRAY` type holds a pointer to a standard C++ array, but has some extra fields that describe the type of the array, its number of dimensions, and most importantly its lower and upper bounds. Handling arrays of this type can be done fairly easy using standard `SAFEARRAY` manipulation functions, such as **SafeArrayCreate**, **SafeArrayRedim**, and **SafeArrayDestroy**.

You will need to pass either one-dimensional or two-dimensional safe arrays to any of the COM methods within this product. Here is how you can accomplish this from Visual C++ 6:

- **Declaring a One-Dimensional Safe Array**
  Use the **SafeArrayCreateVector** function to create a one-dimensional safe array, by passing the element type, the lower bound and the number of elements the array has. The element type is one of the types declared by the `VARENUM` enumeration type. Of the most common types, you will be using the `VT_R8` constant (corresponding to the *double* type in the API Reference), `VT_I4` (corresponding to the integer type), `VT_DATE` (the *DateTime* type), and `VT_BOOL` for *boolean* types.

  After making the call to this function, you will need to copy the values referenced by a C++ one-dimensional array to the memory location referenced by the *pvData* field of the safe array structure. This C++ array must hold the values that you wish to send across to the COM method as a parameter value.

  For example, in order to create a one-dimensional array of 4 double elements, here is how you could define the corresponding safe array and the underlying C++ one-dimensional array:

```
int noElements = 4;
double pvData_myOneDimensionalArray[] = {1, 2, 3, 4};
SAFEARRAY *psa_myOneDimensionalArray = SafeArrayCreateVector(
    VT_R8, 0, noElements);
memcpy(psa_myOneDimensionalArray->pvData,
    pvData_myOneDimensionalArray, noElements * sizeof(double));
```

- **Declaring a Two-Dimensional Safe Array**
  In order to declare a two-dimensional array, you will first declare the correspond-

ing C++ two-dimensional array. However, the C++ array needs to be transposed, such that the number of rows becomes the number of columns and vice-versa. This is due to the transposed memory layout of safe arrays, opposed to that of C++ arrays.

For example, if to declare an n by m safe array, you will need to declare an m by n C++ array, and lay the elements in this reversed order.

Also, you will need to make sure that the number of elements of all sub-arrays is the same, because the two-dimensional safe array is a rectangular array, which assumes fixed number of elements on each dimension.

The safe array structure can be initialized using the **SafeArrayCreate** function, which takes the following parameters: the element type, the number of dimensions (two in our case), and the lower bound and number of elements on each dimension. The lower bound and number of elements is declared using a `SAFEARRAYBOUND` structure, one for every dimension.

Assuming a two-dimensional array of 2 by 4 double elements, its corresponding safe array and C++ two-dimensional array must be 4 by 2 and its elements must be laid out as follows:

```
int noRows = 2;
int noColumns = 4;
double pvData_myTwoDimensionalArray[4][2] = {
    { 1, 5 },
    { 2, 6 },
    { 3, 7 },
    { 4, 8 },
};

SAFEARRAYBOUND myTwoDimensionalArrayBounds[2];
myTwoDimensionalArrayBounds[0].lLbound = 0;
myTwoDimensionalArrayBounds[0].cElements = noRows;
myTwoDimensionalArrayBounds[1].lLbound = 0;
myTwoDimensionalArrayBounds[1].cElements = noColumns;

SAFEARRAY *psa_myTwoDimensionalArray = SafeArrayCreate(
    VT_R8, 2, myTwoDimensionalArrayBounds);
memcpy(psa_myTwoDimensionalArray->pvData,
    pvData_myTwoDimensionalArray, noRows * noColumns *
sizeof(double));
```

The actual values in the safe array are { `1, 2, 3, 4` } on the first row, and { `5, 6, 7, 8` } on the second row.

Before sending a safe array to a COM method, you will also need to perform one last step – wrap it within a `VARIANT` type. The `VARIANT` structure is a generic type that can wrap around any other COM type. Define a variable of type `VARIANT`, and use its fields to specify the type of the safe array and the reference to the safe array structure. To describe the type of the safe array, use the `VT_ARRAY` constant and the constant corresponding to the element type of the array.

For example, a variant corresponding to the two dimensional safe array declared above will look as follows:

```
VARIANT myTwoDimensionalArray;
myTwoDimensionalArray.vt = VT_ARRAY | VT_R8;
myTwoDimensionalArray.parray = psa_myTwoDimensionalArray;
```

A complete list of `VARENUM` constants corresponding to all COM types that might be required by this product can be found at this MSDN link.

5.3.7 Declare the Return Type

COM methods that return arrays require special attention. Instead of declaring a C++ array pointer, you will need to declare a pointer to a `SAFEARRAY` structure, and assign to it the result of the method call. Here is how you would declare a variable to hold a safe array returned by a COM method:

```
SAFEARRAY *arrayResult;
```

COM methods that return non-array types can be used like regular C++ methods, by declaring a variable of the corresponding type and assigning to it the result of the method call.

5.3.8 Call the Method

You can call a COM method by using the variable that holds the connection to the COM server and the name of the method, as listed in the COM API Reference. For example, assuming you wish to call a method named *ComMethod*, which takes two `double` values for parameters and returns a third `double` value, here is how you could perform this call from C++:

```
double result = instance->ComMethod (10, 20);
```

Any other COM method calls can be made the same way, by using a different method

```
SAFEARRAY *twoDimArray;
// A fictive call is made to the COM method
twoDimArray = instance->ComMethod (paramValue1, paramValue2 etc.);

// We read off the lower and upper bounds on each dimension
long l1, u1, l2, u2;
SafeArrayGetLBound(twoDimArray, 1, &l1);
SafeArrayGetUBound(twoDimArray, 1, &u1);
SafeArrayGetLBound(twoDimArray, 2, &l2);
SafeArrayGetUBound(twoDimArray, 2, &u2);

// We read through all elements and print them to the screen
long indices[2]; // the indices for every dimension
for (indices[0] = l1; indices[0] <= u1; indices[0]++) {
    for (indices[1] = l2; indices[1] <= u2; indices[1]++) {
        double element;
        SafeArrayGetElement(twoDimArray, indices, &element);
        printf ("twoDimArray[%d,%d] = %lf", indices[0], indices[1],
            element);
    }
}
```

Table 5.1: Handling a two-dimensional safe array in C++

name, a different set of parameter values, and maybe a different return type.

Special care must be taken when processing the result of COM methods, which return a one or two-dimensional array. Since, as described above in section 5.3.7, the type of the variable to hold the result is a pointer to a `SAFEARRAY` structure, you will need to use safe array specific methods, in order to inspect all element values, and reuse them in C++.

Table 5.1 gives an example of how to handle a two-dimensional array, covering most of the operations you are likely to perform in your C++ applications with safe arrays returned by COM methods.

5.3.9 Call "CoUninitialize"

In order to free generic COM related resources held by a C++ client, you must make a call to the CoUninitialize function. The call to this function can be made right before the end of your application or when your client no longer requires to use COM resources. The line of code which releases all COM resources is:

```
CoUninitialize();
```

5.3.10 A Generic Visual C++ Example

In this section we provide a generic Visual C++ example of a simple Console Application, which connects to one of our business classes, invokes one of its methods and prints the result in a window. This source code example works with a new Win32 Console Application, which can be created as described in section 5.3.1. Since this example's structure in not specific to the method being invoked, you may adapt this example to whatever business class, product, and method you wish to call.

This C++ application calls a method named RelativeToAbsolute, belonging to the AssetParameters business class in the WebCab Portfolio for .NET product. This method takes one two-dimensional `double` array parameter and returns another two-dimensional `double` array.

```
// Project1.cpp, a Simple Win32 Console Application
//

#include "stdafx.h"
#include "objbase.h"
#include "stdio.h"

// Adding a reference to the Portfolio COM Type Library
#import "C:\Program Files\WebCab Components\Portfolio for .NET\
\COM Libraries\WebCab.COM.PortfolioDemo.tlb" no_namespace

int main(int argc, char* argv[])
{
    CoInitialize(NULL); // Enable C++ to COM interoperability

    /*
     * Declare a variable to hold the connection to the
     * `AssetParameters' COM server and establish
     * the connection using its GUID.
     */
    COM_AssetParametersPtr instance = NULL;
    instance.CreateInstance(__uuidof(AssetParameters));

    int noRows = 2;
    int noColumns = 4;
    // Reverse the dimensions for the C++ array, by declaring first
    // the number of columns and then the number of rows.
    double pvData_absoluteValues[4][2] = {
        { 100, 140 },
        { 120, 135 },
        { 125, 135 },
        { 115, 140 },
    };
    SAFEARRAYBOUND bounds[2];
    bounds[0].lLbound = 0;
    bounds[0].cElements = noRows;
    bounds[1].lLbound = 0;
    bounds[1].cElements = noColumns;
    SAFEARRAY *psa_absoluteValues = SafeArrayCreate(VT_R8, 2, bounds);
    memcpy(psa_absoluteValues->pvData,
        pvData_absoluteValues, noRows * noColumns * sizeof(double);
```

Table 5.2: Generic VC++ example (continued on the next page)

```
    VARIANT absoluteValues;
    absoluteValues.vt = VT_ARRAY | VT_R8;
    absoluteValues.parray = psa_absoluteValues;

    // Call the `AbsoluteToRelative' COM method
    SAFEARRAY *relativeValues;
    relativeValues = instance->AbsoluteToRelative(absoluteValues);

    // Print the result inside a MessageBox
    char text[200] = "";
    long indices[2];
    long l1, u1, l2, u2;
    SafeArrayGetLBound(relativeValues, 1, &l1);
    SafeArrayGetUBound(relativeValues, 1, &u1);
    SafeArrayGetLBound(relativeValues, 2, &l2);
    SafeArrayGetUBound(relativeValues, 2, &u2);
    for (indices[0] = l1; indices[0] <= u1; indices[0]++) {
        sprintf (text, "%s{ ", text);
        for (indices[1] = l2; indices[1] <= u2; indices[1]++) {
            double element;
            SafeArrayGetElement(relativeValues, indices, &element);
            sprintf (text, "%s%lf ", text, element);
        }
        sprintf (text, "%s}\n", text);
    }

    MessageBox(NULL, text, "The Relative Values", MB_OK);

    CoUninitialize(); // Free all COM specific resources
    return 0;
}
```

Table 5.3: Generic VC++ example (continued)

Chapter 6

Programmer's Guide for Borland C++ Builder

This chapter contains information concerning the use of this Component product from Borland's C++ Builder product lines; include Borland C++ 2005, Borland C++BuilderX and Borland C++Builder; on the Windows development platform. The principles and code examples provided here can be applied verbatim in order to use this component with the C++ language on the Windows development platform. Though the use of the different IDE products lines will differ slightly, the same Windows client code can be used verbatim within any of the C++ Builder product lines.

6.1 Developing with Borland C++ Builder

In order to connect and make calls to our COM servers from Borland C++ Builder, you will need to write late-binding code.

The steps are as follows:

1. Open a New or Existing Project
2. Add all COM Specific 'Include' Declarations
3. Call "`CoInitialize`"
4. Create a Class Instance
5. Obtain a Method ID
6. Declare the Parameter Values and Types
7. Declare the Return Type
8. Call the Method
9. Call "`CoUninitialize`"

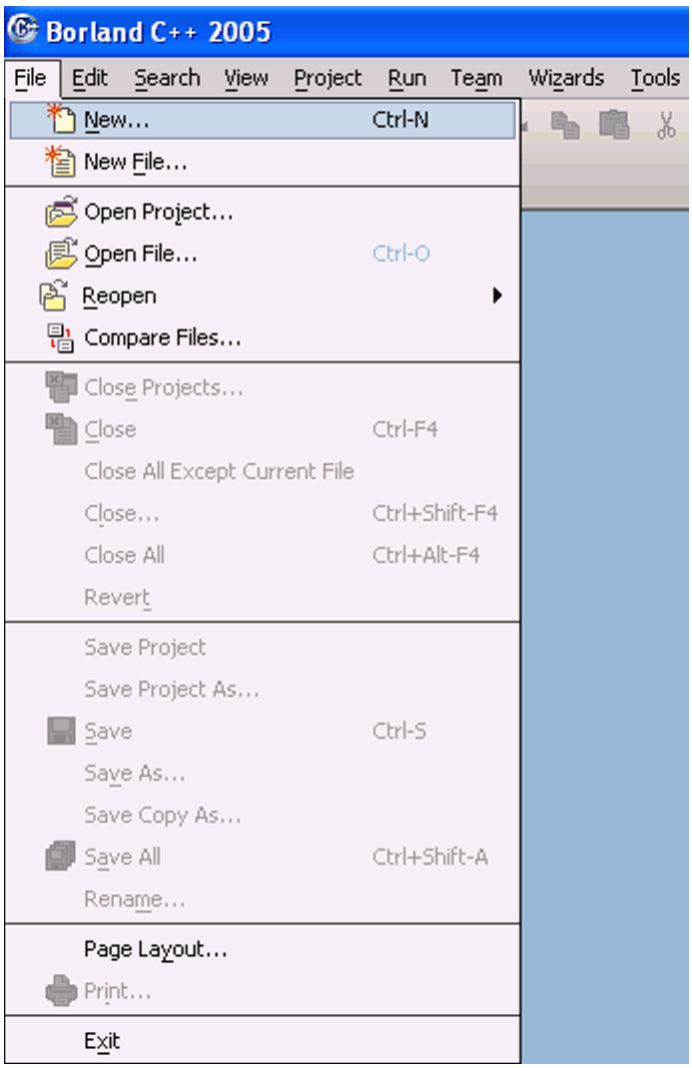


Fig. 6.1: Starting a new project

6.1.1 Open a New or Existing Project

You can start a new Borland C++ Builder Project by going to the File | New... menu (see Fig. 6.1), which will bring up the "Object Gallery" dialog window. From this window, choose the "Project" left-hand item, select the *New Console* project and click OK, as shown in Fig. 6.2.

Type in the name of your project in the next dialog window and click Next (see fig. 6.3). In the next window (figure 6.4) simply click Next, and in the last window, select the checkbox next to the *untitled* entry and click the Finish button, as shown in figure 6.5.

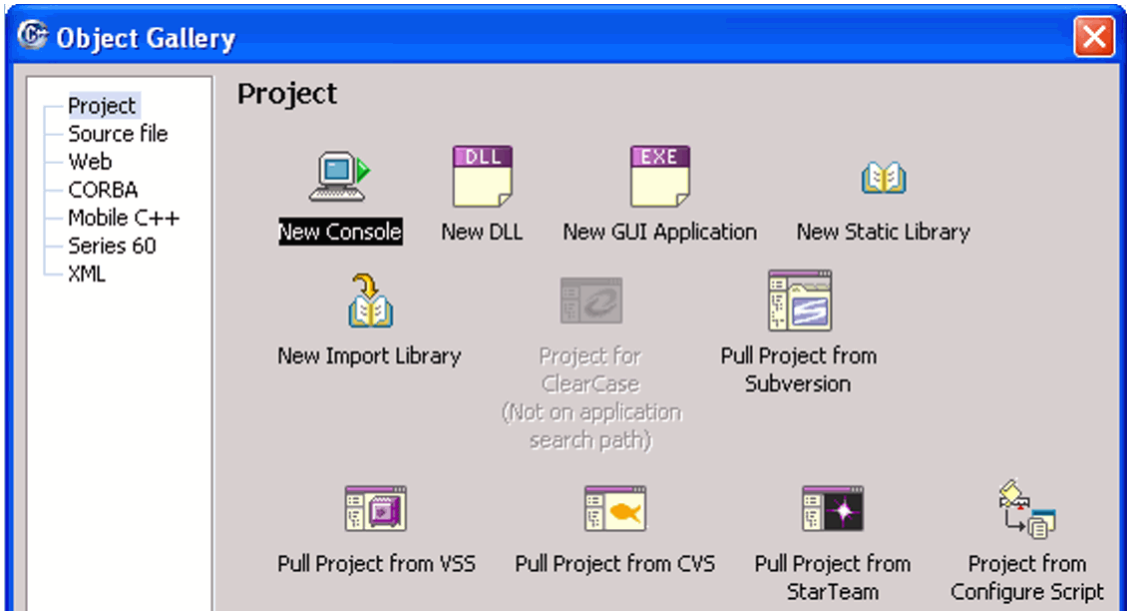


Fig. 6.2: Starting a Console Application

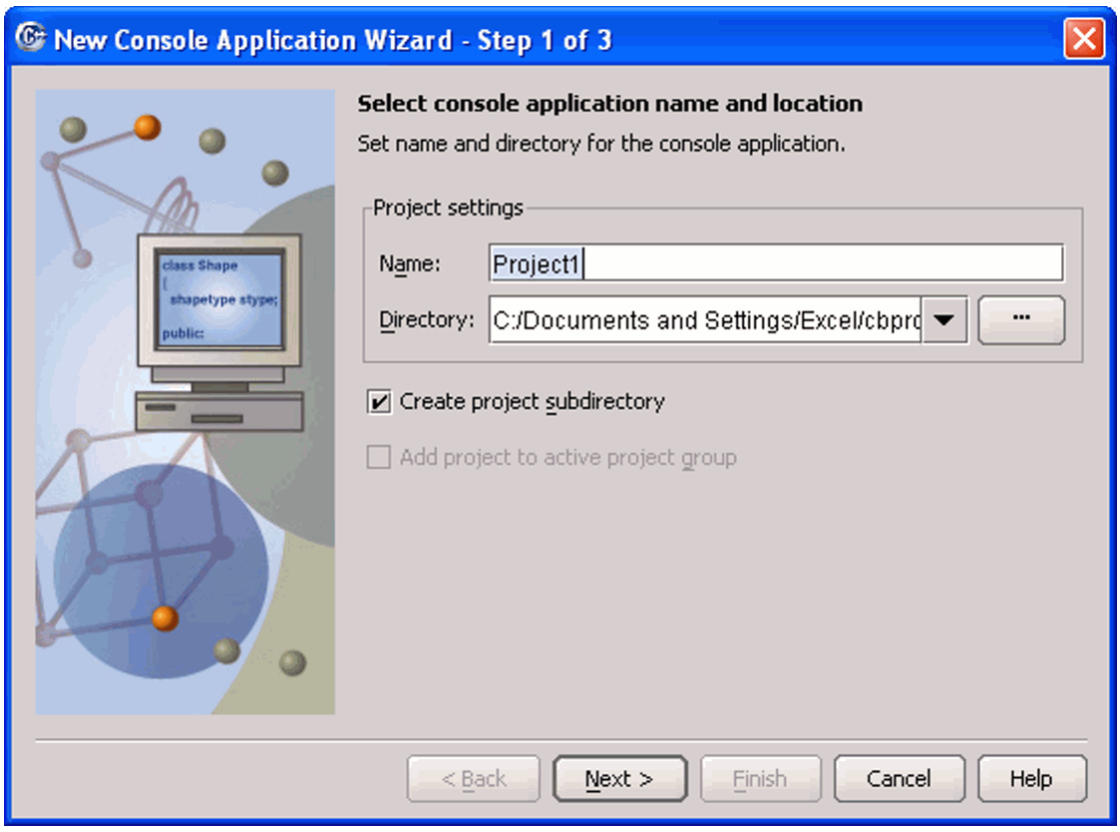


Fig. 6.3: Choosing a name for a new Borland C++ Console Application

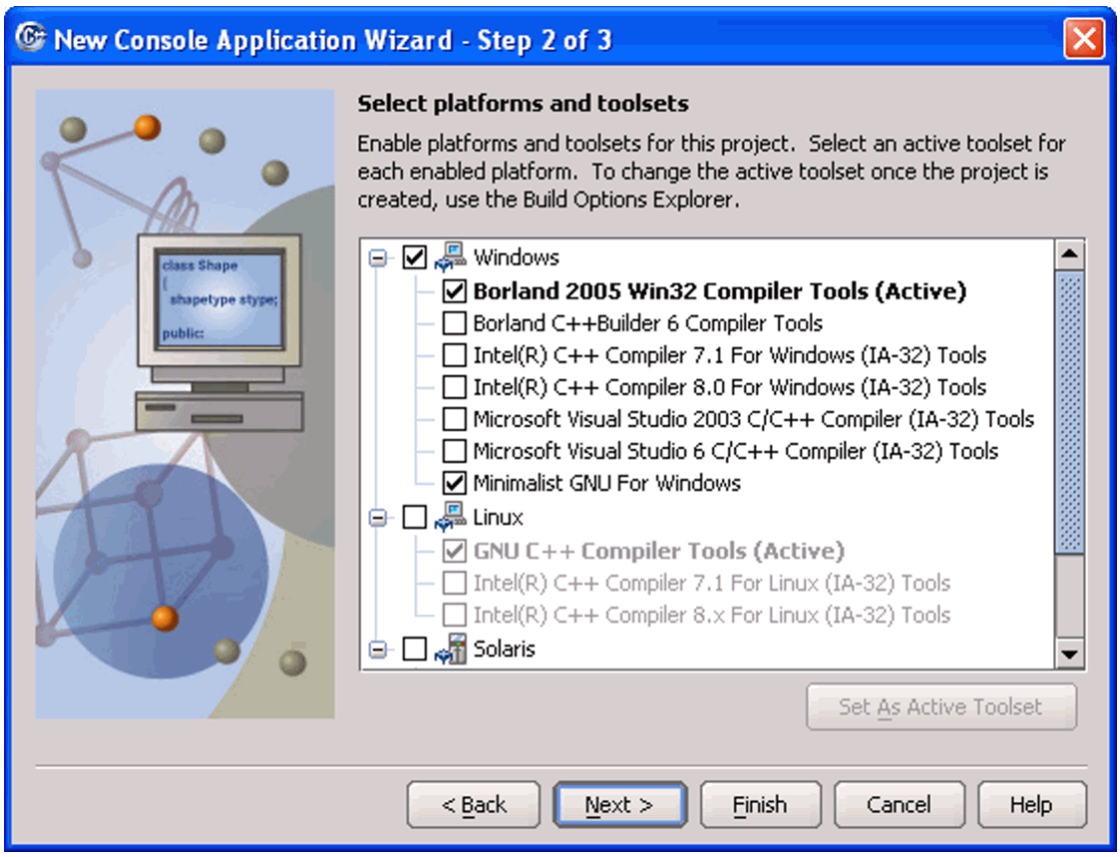


Fig. 6.4: Selecting platforms and tools sets for a new Borland C++ Console Application

6.1.2 Add all COM Specific 'Include' Declarations

COM specific calls will require that the header file *objbase.h* is included within your project. This header offers functionality to enable C++ to interoperate with COM servers. To include this header file within your project you will need to add at the top of your main source code file the following line:

```
#include "objbase.h"
```

6.1.3 Call "CoInitialize"

In order to allow Borland C++ Builder clients to connect to a COM server, you need to make a call to a function named CoInitialize, which belongs to the *objbase.h* header file, mentioned above. The call to the function must be made as shown below:

```
    CoInitialize(NULL);
```

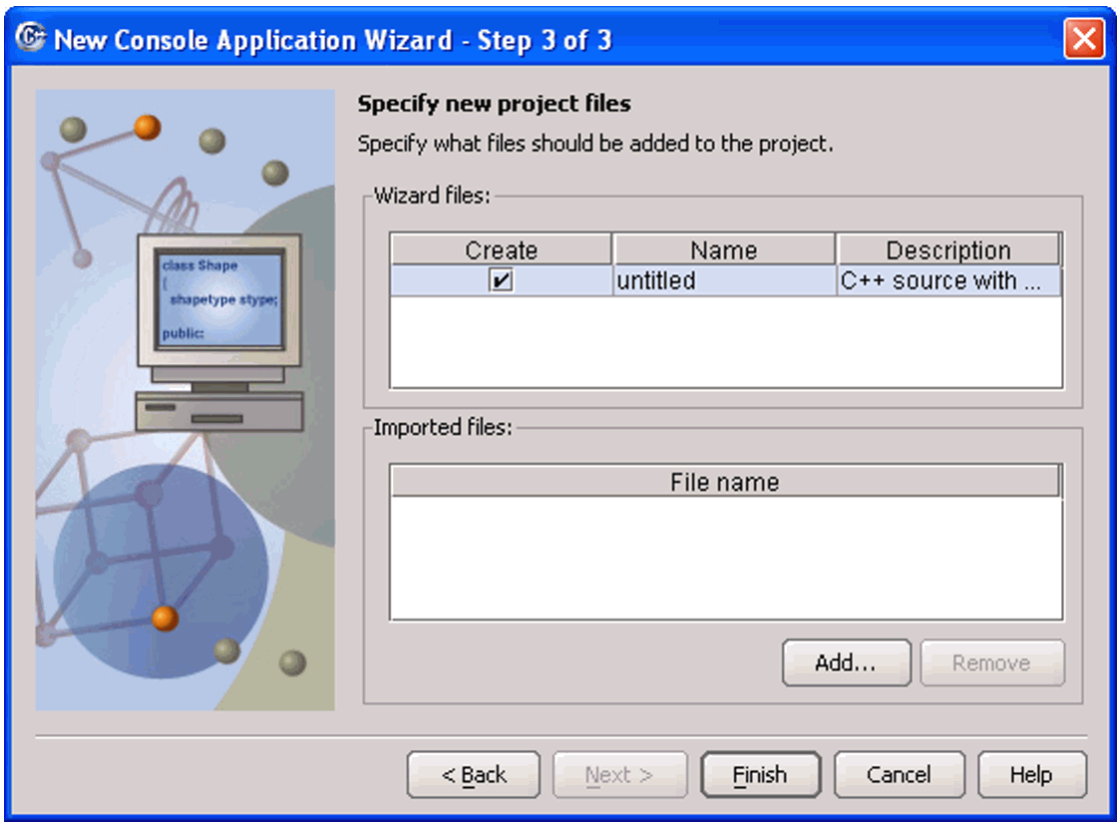


Fig. 6.5: Specifying the project files for a new Borland C++ Console Application

6.1.4 Create a Class Instance

In order to create an instance of a WebCab Portfolio business class, you will need to provide the full name (i.e. the ProgId) of the business class, as declared within the API Reference. The full name of a business class contains the namespace, followed by a period and the name of the class itself.

The C++ code below returns an instance of the Interpolation business class, located inside WebCab Functions for .NET. Its namespace is WebCab.COM.Math.Interpolation, meaning its full name is WebCab.COM.Math.Interpolation.Interpolation. You can easily adapt the code below to create an instance of another business class, by replacing the ProgId below with the full name of the class you wish to use.

```
CLSID clsid;
CLSIDFromProgID(OLESTR( // Get the CLSID of the ProgId
    "WebCab.COM.Math.Interpolation.Interpolation"),
    &clsid);
LPDISPATCH instance = NULL; // the COM `instance'
LPUNKNOWN punk = NULL;

CoCreateInstance (clsid, NULL, CLSCTX_INPROC_SERVER,
    IID_IUnknown, (void**) &punk);

punk->QueryInterface(IID_IDispatch, (void**) &instance);
punk->Release();
```

The variable named "instance" will hold the connection to the "Interpolation" COM server.

6.1.5 Obtain a Method ID

In order to call a method using the instance created above, you will need to obtain a reference to its ID. Use the code below to obtain the ID of the method you wish to call, by replacing the generic method name "MyMethod" with the name of the method you wish to call. The variable "methodID" will hold the reference to the method.

```
DISPID methodID; // The ID of the method

/*
 * Replace `MyMethod ' with the name of the method you wish to call
 */
wchar_t *methodName = L"MyMethod";
instance->GetIDsOfNames(IID_NULL, &methodName, 1, LOCALE_USER_DEFAULT,
    &methodID);
```

6.1.6 Declare the Parameter Values and Types

For every parameter, you will need to declare a variable of type `VARIANT`, whose type will be set to the type of the parameter, and its value will correspond to the value you wish the parameter to take. The code listed in table 6.1 shows how to declare the parameter values for a method that takes two `double` parameters.

For information about how to set other types of parameters, such as integers, boolean values, and arrays; we refer the reader to this 'IDispatch Data Types and Structures' article on Microsoft's MSDN site.

```
int noParameters = 2; // the number of parameters
// 1st parameter
VARIANTARG parameter1;
parameter1.vt = VT_R8; // 8-bit real type
parameter1.dblVal = 100.0; // 100.0
// 2nd parameter
VARIANTARG parameter2;
parameter2.vt = VT_R8; // 8-bit real type
parameter2.dblVal = 95.0; // 95.0

DISPPARAMS parameters; // the list of parameters
memset(&parameters, 0, sizeof(DISPPARAMS));
parameters.cArgs = noParameters;
parameters.rgvarg = new VARIANTARG[noParameters];
memset(parameters.rgvarg, 0, sizeof(VARIANT) * noParameters);

// Add the parameters in reverse order
parameters.rgvarg[1] = parameter1;
parameters.rgvarg[0] = parameter2;
```

Table 6.1: Declaring parameter types and values for a COM method

6.1.7 Declare the Return Type

A `VARIANT` variable needs to be declared for the return value of the method as well. The type of the `VARIANT` will be automatically set by the method invocation, so you will only need to initialize its type to `VT_EMPTY` by calling the `VariantInit` function.

```
VARIANTARG result; // Variable to hold the result
VariantInit(&result);
```

6.1.8 Call the Method

Calling the method comes down to using all the variables declared in the previous steps (the instance, the method ID, the parameters, and the return value) in one call, as shown below:

```
// Call the underlying method
instance->Invoke(methodID, IID_NULL, LOCALE_SYSTEM_DEFAULT,
   DISPATCH_METHOD, &parameters, &result, 0, NULL);
```

The return value of the method is stored in the `result` variable, in the field corresponding to the method's return type. For example, if the method returns a `double`, you will write the following code in order to store the result in a `double` variable named "resultAsDouble":

```
// Get the result as double
double resultAsDouble = result.dblVal;
```

6.1.9 Call "CoUninitialize"

In order to free generic COM related resources held by a C++ client, you must make a call to the CoUninitialize function. The call to this function can be made right before the end of your application or when your client no longer requires to use COM resources. The line of code which releases all COM resources is:

```
CoUninitialize();
```

6.1.10 A Generic Borland C++ Builder Example

In this section (tables 6.2 and 6.3) we provide a generic Borland C++ Builder example of a simple Console Application, which connects to one of our business classes, invokes one of its methods and prints the result in a window. The source code below works with a new Win32 Console Application, which can be created as described in section 6.1.1. Since this example's structure in not specific to the method being invoked, you may adapt this example to whatever business class, product, and method you wish to call.

This C++ application calls a method named Kairi, belonging to the MovingAverage business class in the WebCab Technical Analysis for .NET product. This method takes two `double` parameters and returns a `double` value.

```
// Project1.cpp, a Simple Win32 Console Application
//

#include "objbase.h"
#include "stdio.h"

int main(int argc, char* argv[])
{
    CoInitialize(NULL); // Enable C++ to COM interoperability

    CLSID clsid;
    CLSIDFromProgID(OLESTR( // Get the CLSID of the ProgId
        "WebCab.COM.Finance.Trading.Indicators.MovingAverage"),
        &clsid);
    LPDISPATCH instance = NULL; // the COM `instance'
    LPUNKNOWN punk = NULL;

    CoCreateInstance (clsid, NULL, CLSCTX_INPROC_SERVER,
        IID_IUnknown, (void**) &punk);

    punk->QueryInterface(IID_IDispatch, (void**) &instance);
    punk->Release();

    DISPID methodID; // The ID of the method

    // The method name is `Kairi'
    wchar_t *methodName = L"Kairi";
    instance->GetIDsOfNames(IID_NULL, &methodName, 1, LOCALE_USER_DEFAULT,
        &methodID);

    int noParameters = 2; // the number of parameters
    // 1st parameter
    VARIANTARG parameter1;
    parameter1.vt = VT_R8; // 8-bit real type
    parameter1.dblVal = 100.0; // 100.0
    // 2nd parameter
    VARIANTARG parameter2;
    parameter2.vt = VT_R8; // 8-bit real type
    parameter2.dblVal = 95.0; // 95.0
```

Table 6.2: Generic VC++ example – Part 1

```
    DISPPARAMS parameters; // the list of parameters
    memset(&parameters, 0, sizeof(DISPPARAMS));
    parameters.cArgs = noParameters;
    parameters.rgvarg = new VARIANTARG[noParameters];
    memset(parameters.rgvarg, 0, sizeof(VARIANT) * noParameters);

    // Add the parameters in reverse order
    parameters.rgvarg[1] = parameter1;
    parameters.rgvarg[0] = parameter2;

    VARIANTARG result; // Variable to hold the result
    VariantInit(&result);

    // Call the `Kairi' method
    instance->Invoke(methodID, IID_NULL, LOCALE_SYSTEM_DEFAULT,
       DISPATCH_METHOD, &parameters, &result, 0, NULL);
    delete [] parameters.rgvarg;

    // Get the result as double
    double percentage = result.dblVal;
    // Print the result inside a MessageBox
    char text[200];
    sprintf(text, "The percentage is %lf", percentage);
    MessageBox(NULL, text, "Method result", MB_OK);

    CoUninitialize(); // Free all COM specific resources
    return 0;
}
```

Table 6.3: Generic VC++ example – Part 2

Chapter 7

Programmer's Guide for .NET

This chapter describes development techniques for various business solutions that make use of the functionality provided by our .NET Service. We analyze both standard and enterprise solutions, including stand-alone applications, ASP.NET pages, and XML Web services.

7.1 Developing with .NET Class Libraries

The .NET Edition of this product offers core-level functionality to the .NET developer allowing the implementation of fully personalized components and applications for specific business problems on a variety of deployment environments. Irrespective of the complexity of the project a programmer may be working on, this .NET Service can be integrated in an equally straightforward manner by providing the needed functionality in a compact and precise way.

The Portfolio for .NET v5.0 component comes as a collection of DLL files. Each DLL is a packed collection of classes that provide the functionality offered by this .NET Service as documented inside the API reference directory of this package. Depending on the type of .NET solution you are developing, you will be using these DLL files in a specific way. Once you have chosen a particular implementation of a deployment framework, the structure of your source code will need to adapt accordingly. In the following discussion we describe several basic .NET implementation examples that can be used to make use of our product's functionality.

7.1.1 Stand-alone C# .NET Applications

A stand-alone C# application is a C# class which acts as a client for the deployed .NET components (i.e. DLLs). The source code listed below defines a standard stand-alone application skeleton which communicates with a class within a .NET Component. By creating an instance, calling a method, sending in a set of parameters and then retrieving the returned result of the method call. As soon as the method call has gone through successfully the method continues by displaying its result on the screen.

```
/*
 * The class we are is located inside the WebCab.Libraries.NetService
 * namespace.
 */
using System;
using WebCab.Libraries.NetService;

public class StandAloneExample {

   public static void Main () {

      /*
       * Creates an instance of the NetClass class
       */
      NetClass instance = new NetClass ();

      /*
       * Defines the method parameter.
       */
      double parameter = 25;

      /*
       * Invokes a method with the specified parameter
       * and puts the result in the result variable.
       */
      double result = instance.ComputeResult (parameter);

      /*
       * Prints out the retrieved result.
       */
      Console.WriteLine ("Method 'ComputeResult' in class NetClass
           returned " + result + ".");
   }

}
```

If the method `ComputeResult` of the `NetClass` class returned `100`, this stand-alone application would print the following:

```
Method 'ComputeResult' in class NetClass returned 100.
```

7.2 Developing with XML Web Services

This component has been XML Web service enabled and the relevant deployment files can be deployed which means that you may deploy the Application as a web service and make the methods contained therein available as XML Web service methods.

7.2.1 Deploying the XML Web Services

Using Windows XP with .NET Framework and IIS installed

In order to deploy this .NET Service as an XML Web service you will need to go through the following steps:

- Copy DLL Files - copy the DLL files provide into the bin directory of the IIS servers directory. On the standard install this is located at:

  ```
  C:\Inetpub\wwwroot\bin
  ```

- Copy asmx Files - now copy the asmx files included into the folder:

  C:\Inetpub\wwwroot\

  Now the web service can be accessed through the URL:

  ```
  http://localhost/WebServices/filename.asmx
  ```

Remarks

1. If you prefer to change the URL in which the XML Web service can be accessed then just create and then copy the asmx files into another subfolder of the 'wwwroot' directory, in which case the corresponding URL of the web service will corresponding change.
2. In order to copy the installation files to your IIS location you will need to have appropriate read and write access to the folder under question.

7.2.2 Writing XML Web Service Clients

The building of an XML Web service client will require the following steps:

- Create a proxy class for the XML Web service.

- Reference the proxy class in the client code.
- Create an instance of the proxy class in the client code.
- Call the method on the proxy class corresponding to the XML Web service method you want to communicate with.

For most clients, these steps differ only on how the proxy class is referenced and how the XML Web services are deployed.

7.2.3 Writing Console XML Web Service Clients

An XML Web service client may take a number of forms including a console application, ASP.NET page, HTML page, Windows Form, Java Applet/Frame and any other XML Web service enabled client technology.

Here in order to illustrate the core step involved in making a client we describe the steps required to build a console application. Creating other types of clients will involve similar steps. We assume that you have the .NET Framework installed (on the client and server) and that the IIS server has (been installed and) started on the server.

To create a console XML Web service client application you must:

- Deploy the DLL provided into a bin which you need to create somewhere in a subdirectory of the IIS web server you plan to use. The default location for this web server on your local machine is 'C:\Inetpub\wwwroot'. The DDL is the required XML Web service .NET assembly file.
- Deploy the web service descriptor 'Classname.asmx' provided, into a IIS web-server subdirectory.[1] Note that we will assume below that the URL of the deployed Classname.asmx file is:

```
http://yourserver/Webservices/Classname.asmx
```

- Create a XML Web service proxy by using the WDLS command line tool. Enter a DOS prompt in the directory *where you intend to* (develop and) run your console application from and type at the command line:

```
Wsdl.exe http://yourserver/Webservices/Classname.asmx
```

[1] The purpose of the asmx file is to tell the server that the XML Web service exists so that it may start searching for the relevant classes within the DLL files in the bin directory. Hence, the asmx file is very simple and consists of the one line:

```
<%@ WebService language="C# " class="Classname" %>
```

and is saved as a TXT file with an extension .asmx.

That is 'Wsdl' followed by the URL of the asmx file for the XML Web service. This utility will generate a C# file which will be written to the present directory (which should be the clients directory)[2]. The WSDL file is an interface (in OOP terminology) or proxy for the DLL C# implementation of the XML Web service.

- Write a console application in the standard way and instantiate the WebService methods (as if the DLL is hosting locally) using the WSDL proxy. See the below template:

```
using System;


public class ClassnameClient {
   /// <remarks>
   /// The Main method ensures that the console will run.
   /// </remarks>
   static void Main() {
      /*
       * We instantiate the "object" represented by
       * the proxy class.
       */
      Classname classnameObject = new Classname();
      /*
       * We invoke the 'Compose' method of the
       * Classname XML Web service
       */
      Console.WriteLine("XML Web service method Compose
         applied to 7.0 and 3.9 is " +
         classnameObject.Compose(7, 3.9));
   }
}
```

[2]WSDL is a standard by which all XML Web services are described. See http://www.w3.org/TR/wsdl

7.2.4 Importing Web services into Visual Studio .NET projects

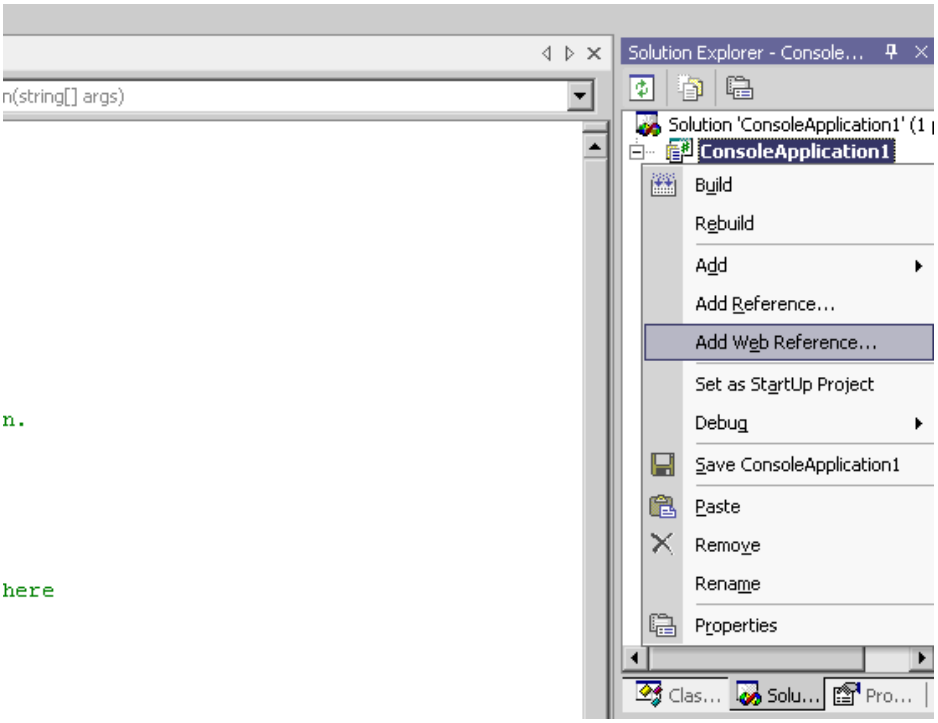


Fig: Adding a Web reference.

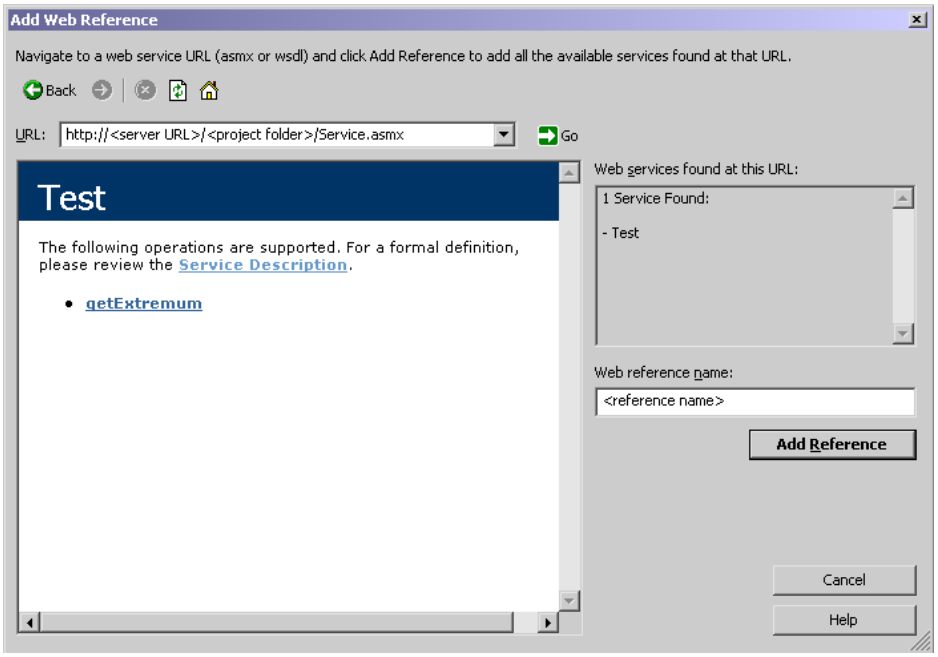


Fig: Adding a Web reference.

7.3 Connecting to a Database with our .NET Libraries

7.3.1 Overview

Most of the financial and mathematical functionality provided by this .NET Service is intended for use within a multi-tier environment where the back-end role is taken by an SQL Database server. By packaging this functionality within .NET components such as XML Web services and .NET Class Libraries we separate the business logic from the database logic, allowing you to customize the way you access the database and the way the retrieved data is sent to the business methods.

In order to assist you in developing DBMS based .NET solutions, we provide the ADO Mediator as a standardized way of using our .NET components with an SQL Database server. The ADO Mediator is also a .NET component, which sits on the same level as the other financial and mathematical .NET components and is a seamlessly integrated component of this .NET Service. While reducing the amount of DBMS code you have to type, it allows you to concentrate on the SQL queries and procedure calls to the Database server and map the stored data directly to the business methods of this .NET Service.

The ADO Mediator's functionality has been enabled for use with the following modules contained within the WebCab Portfolio for .NET v5.0:

- The **"Portfolio"** Module

7.3.2 The ADO Mediator

There is one `ADOMediator` class for every module in this .NET Service. You will use the corresponding `ADOMediator` class for performing Database operations with classes of a certain module. The ADO Mediator class is located inside the '`ADO`' subnamespace of its corresponding module. For example if a module contains the:

```
WebCab.Libraries.Finance.Trading
```

namespace, its `ADOMediator` class will be found under:

```
WebCab.Libraries.Finance.Trading.ADO.ADOMediator.
```

Configuring the ADO Mediator

There are three steps to perform in order to start using an ADO Mediator class:

1. **Specify the ADO.NET Driver**
   The constructor accepts an input connection and an output connection. There is no restriction that you should provide both an input and an output connection. You may choose to read data and display it inside your Application, compute data and write it into the database, or read and write to and from the database, and specify only the input or output database connection.

Connecting to your database requires an ADO.NET Driver. This driver is a .NET class which is part of a DLL file provided by your Database vendor. You specify the driver you wish to use by sending in to the `ADOMediator` constructor its run-time type. That is, if your driver class name is `System.Data.SqlClient.SqlConnection`[3], its run-time type under C# is specified by typing inside your source code:

```
typeof (System.Data.SqlClient.SqlConnection)
```

2. **Provide the input/output `ConnectionString` property**
   The `ConnectionString` property is a piece of text which describes the Database server machine address, the credentials, and additional information required in order to obtain access to certain parts of your Database server. This piece of text is always documented by the provider of the ADO.NET Driver and should be used as such when submitting it to an `ADOMediator` constructor. An example for the previous ADO.NET SQL Server driver would be:

   ```
   "Initial Catalog=Northwind;Data Source=localhost;Integrated Security=true;"
   ```

   You may easily notice that the string above describes the machine name and the initial catalog to connect to. The `Integrated Security=true` bit enables you to connect to the SQL Server 2000 by using the current Windows login username and password.

3. **Assign a business .NET component**
   The previous two steps are part of creating an instance of an `ADOMediator` class and taking care of the Database connections. This third step involves specifying the business class belonging to this module which deals with the financial and mathematical functionality. Choose a class of the corresponding module and assign an instance of this class to the `UnderlyingInstance` property of the previously created `ADOMediator` instance.

   For example, if the name of the class you choose is `TradingClass`, you could write the following C# code[4]:

   ```
   TradingClass aTradingClassInstance = new TradingClass ();
   adoInstance.UnderlyingInstance = aTradingClassInstance;
   ```

Using the ADO Mediator

Once these three steps have been performed, you may use the created `ADOMediator` instance to perform Database related calculations using the assigned .NET business class. There

[3] The built-in ADO.NET driver class name for connecting to the MS SQL Server
[4] We assume `adoInstance` is the name of the previously created `ADOMediator` instance.

are several methods which you may use with your ADO Mediator instance. Some methods allow you to read from the database and perform calculations for every retrieved row, and then store the result in a .NET variable. Other methods allow you to specify the input values and write the result in the database. There are also methods which allow you to read the input parameters from the database and write the method results back into the database in one go.

The basic methods of the ADO mediator are:

- `Select` Method - Allows you to retrieve the results of a method of your choice belonging to the underlying business class instance, applied to every row of an SQL `SELECT` query.
- `Update` Method - Directly updates values inside your database, as returned by a method belonging to the underlying business class instance.
- `SelectAndUpdate` Method - Allows you to combine the two methods mentioned above by selecting the data, computing the results and writing it back to the database, according to your own criteria.

A complete API reference for these methods is found inside the `Documentation` folder of this pack. The Windows Installer (MSI) for this .NET Service has also created a short cut to this documentation from the Start Menu.

How the ADO Mediator Works

The way the ADO Mediator processes requests from the client and transfers them to the .NET components and the Database is shown in the diagram below.

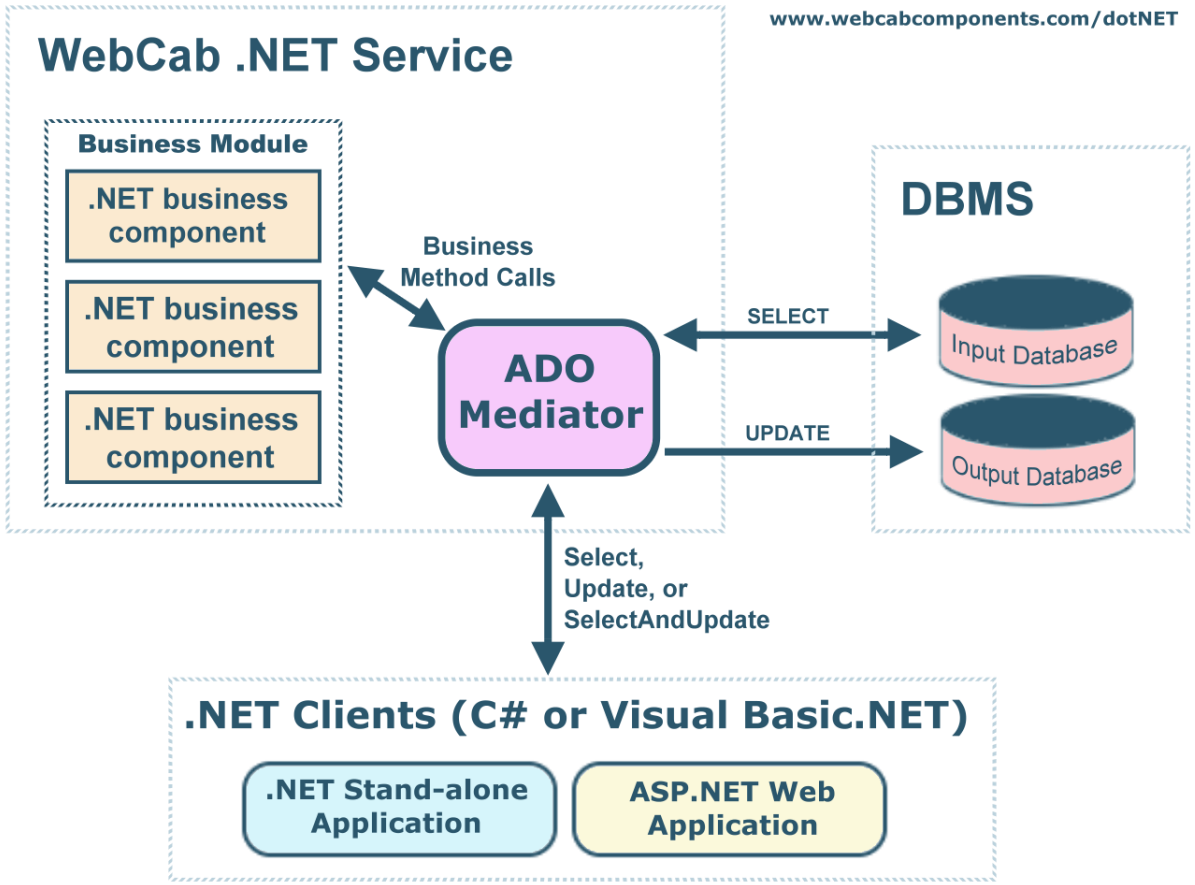


Fig: This diagram describes how the ADO Mediator passes onto the Database and the .NET components its client requests.

C# Source Code Example

The following C# source code makes use of the ADO Mediator's functionality in order to read and update the database according to a known algorithm. We use the standard SQL Server ADO.NET Driver to connect to a fictitious table named `TRADES`, and invoke the `TradingMethod` method of the `TradingClass` business class.

```
using System;
using System.Data.SqlClient;
using WebCab.Libraries.Finance.Trading;
using WebCab.Libraries.Finance.Trading.ADO;
...
  try {
    ...
    /*
     * We specify the same driver and connection string for both the input and output
     * connections to the SQL Server 2000 Database server.
     */
    ADOMediator adoInstance = new ADOMediator (
        typeof (SqlConnection), // Input
    "Initial Catalog=Northwind;Data Source=localhost;Integrated Security=true;",
        typeof (SqlConnection), // Output
    "Initial Catalog=Northwind;Data Source=localhost;Integrated Security=true;");

    /*
     * We assign an instance of the TradingClass business class to the newly created
     * instance of this ADO Mediator.
     */
    adoInstance.UnderlyingInstance = new TradingClass ();

    /*
     * We invoke the SelectAndUpdate method which will populate the table according to
     * the result of the method and the position of the input values in the database.
     */
    adoInstance.SelectAndUpdate ("TradingMethod",
        "SELECT C_ID, SHARES, VALUE FROM TRADES", // Select
        "UPDATE CUSTOMER SET MONEY=@MONEY WHERE C_ID=@C_ID", // Update
        "@MONEY", // Maps the 'TradingMethod' method result to @MONEY
        {new Object[] {"@C_ID", "C_ID"}}, // Maps C_ID to @C_ID
        false); // True when updating with a stored procedure
  }
  catch (ADOMediatorException e) {
    Console.WriteLine (e);
  }
```

Using ADO Mediator with SQL Server 2000

The ADO Mediator instance above requires the same information as in the case of writing your own ADO.NET source code. The difference lies in the fact that you do not have to manually control the driver specific operations such as opening and closing a connection, and only concentrate on your SQL queries and the functionality of our business classes. You start by following the three steps mentioned above[5] and continue by making calls to any `ADOMediator` methods.

[5]Specifying the input and/or output ADO.NET drivers and their connection strings and assigning an instance of the fictitious `TradingClass` business class.

As you may notice, the `SelectAndUpdate` method above follows a logical pattern of describing how the data should be read from the database, computed, and written back, with special care to the use of named parameters. You do not require previous experience in dealing with named parameters, as they are basically a way to substitute real values with names inside SQL queries. You may have spotted the `@MONEY` and `@C_ID` named parameters in our `UPDATE` statement above. These named parameters will be replaced with real values, taken either from the `TradingMethod`'s result, or directly from a column returned by the `SELECT` query.

In our case, we replace `@MONEY` with the result of applying the `TradingMethod` method[6] to the values of the `SHARES` and `VALUE` columns returned by the `SELECT` query. The second named parameter (`@C_ID`) is replaced directly by the value of the `C_ID` column of the `SELECT` query. For more information about named parameters you may wish to consult your ADO.NET driver's manual. Most special cases are related to the use of named parameters and have been covered by the current version of the ADO Mediator.

7.4 Portfolio Methods Overview

This .NET Service offers portfolio analysis according to Markowitz and Capital Market Theory. There are two main categories of methods which are provided within this .NET Service:

- statistical parameter calculation
- portfolio analysis

The values returned from the first category are used when calling portfolio analysis methods. You can calculate either individual asset parameters (e.g. the expected return from an asset), or portfolio parameters (e.g. the expected return of the portfolio). Usually you will be interested more in the analysis functions. Almost all of these functions require the covariance matrix and/or the expected returns vector. You can obtain them by calling `CovMatrix`, respectively `ExpArray`. Also, when calculating statistical parameters, you can give the probabilities as parameters or you can use an estimation based on historic rates of return (see 3.1.1 for details).

The most time consuming part of the analysis is the determination of the efficient frontier (see 3.4). This requires a series of minimizations, which are done using Rosen's gradient projection algorithm. Because it is inefficient to do this every time you want to know the portfolio on the efficient frontier corresponding to a given expected return, the result is determined by interpolating through a series of points. The interpolation points must be calculated before a call to `EfficientFrontier(double, int)` is made. The method `CalculateEfficientFrontier` does this. If you are interested instead in only one point from the efficient frontier, you can use `EfficientFrontier(double, double[][],`

[6] Which in our example belongs to the fictitious `TradingClass` business class.

`double[], int, double)`. In order to determine the optimum portfolio (according to Markowitz model) a utility function (indifference curve) must be provided (see 3.5). It can be given either as a set of interpolation points, or as a polynomial (by specifying coefficients). Having a utility function and an efficient frontier, you can determine the set of optimal portfolios (i.e. the set of intersection points between the two curves) using `OptimalPortfolio`. If you are interested only in one value you can use `OptimalPortfolioMaxExpected` which determines the point in the set with the maximum expected return. The maximum number of optimal portfolios found is 100. If this number is exceeded, an exception will be thrown.

We provide a method for calculating the equity portfolio (see 3.6). This also requires the efficient frontier to be calculated.

Remarks You should be aware that there are two sources of errors: the first is the optimization procedure, the second is the interpolation. Usually the errors generated by the interpolation are much bigger than that resulting from the optimization. You can reduce the interpolation errors by increasing the number of interpolation points. It is much harder to control the optimization errors. If you are especially interested in an accurate result, you can adjust the 'portal' parameter to lower values, otherwise this should always be set between $1E-3$ and $1E-6$. The 'projtol' parameter is not the tolerance of the result, so there are no guarantees that the error will be less than a specific number. Also the increase in duration of the algorithm does not depend linearly on the value of 'projtol', so a small decrease of it can cause a great increase in time consumption. Generally, the accuracy of the result depends on the input data. Usually, the more assets are in the portfolio, the bigger optimization errors are.

7.5 Exceptions

If one of the methods which require the efficient frontier to be calculated is called before `CalculateEfficientFrontier`, the exception `EfficientFrontierNotCalculatedException` will be thrown.

If one of the methods which require the utility function is called without initializing it, the exception `UtilityFunctionNotInitializedException` will be thrown.

If the number of optimum portfolios found is greater than 100, the exception `TooManyPortfoliosExceptic` is thrown.

The expected return of a portfolio cannot be greater than the maximum expected return of the assets. If you want an expected return higher than this, the problem has no solutions. So, when calling `EfficientFrontier` with such an expected return, you will get a `NoSolutionException` . Also if the range given to `CalculateEfficientFrontier` contains any points for which there is no solution, the exception will be raised.

Chapter 8

Examples

Within this chapter we illustrate how the Portfolio Component can be applied to solve real life problems. We provide with this Component examples of two types:

1. **QA Examples**
2. **Custom Examples**

8.1 Question and Answer (QA) Client Examples

8.1.1 Overview

Within this folder we offer client examples written using a number of .NET compliant languages for the .NET Business Components provided within this package. In particular, for each business method contained within each business Component we provide an example implemented within C# , VB.NET and C++.NET; which illustrates how functionality contained within this component can be applied to solve real world problems. In addition, to these examples we also include custom applications which solve some of the generic problems which this Component is designed to address.

8.1.2 Structure of QA Examples Directory

By drilling down the QA Client directory structurr you will find the following six directory levels:

1. Client Level
2. Language Level
3. Business Module Level
4. Business Class Level
5. Example Level
6. Custom Example Level

which have the structure as shown in the following diagrams.

Top Three levels of the QA Directory Structure

```
(Client Level)   (Language Level)          (Business Module Level)

QAClients +
          |
          + C# ------------------------+
          |                             |
          + VB.NET  --+...              + 'Business Module 1' ----------
          |                             |
          + C++.NET --+...              + 'Business Module 2' --+...
          |                             |
          + QALibrary.dll               + ...
          |
          + QALibraryModule.netmodule
          |
          + Readme.txt
```

Bottom Three levels of the QA Directory Structure

```
    (Business Class Level)        (Example Level)     (Custom Example Level)

--+
  |
  + 'Business Class 1' --------+
  |                            |
  + 'Business Class 2' --+...  + 'BusinessClass'.exe.conf
  |                            |
  + ...                        + 'BusinessClass'.cs
                               |
                               + compile.bat
                               |
                               + run.bat
                               |
                               + run_gui.bat
                               |
                               + 'CustomClient' -------+ 'CustomClient'.cs
                               |                       |
                               + ...                   + compile.bat
                                                       |
                                                       + run.bat
```

8.1.3 Quick Start Guide

Browse down to the particular client within the business module for the given .NET compatible language which you are interested in. Run the compile.bat, script in order to compile the example and then run the run.bat, script in order to run the example.

8.1.4 Explanation of the QA Directory Structure and its files

1. **Client Level**

   The directory containing the Questions and Answer Examples is named 'QAClients'. This directory can be located by using the Readme.html file, which is presented at the end of the installation process, by selecting:

   Run Examples > Run .NET Class Library Examples

   The Readme.html file can also be opened from the shortcut:

   Start > Programs > WebCab Components > 'Name of Components Package' > Readme.html

2. **Language Level**

   Within the 'QAClients' folder is the Language level of the QA directory structure. Within this folder you will find 3 sub-directories: C# , VB.NET and C++.NET; which contain examples written in each of the .NET languages C# , VB.NET and C++.NET respectively. Within this folder you will also find the following three files:

   - **QALibrary and QALibraryModule.netmodule** - contain utility functionality necessary for running the clients. If you wish to run these examples from within Visual Studio .NET (for example) then you will need to register this DLL within your VS.NET project.
   - **Readme.txt** - this Readme file.

   As mentioned above within the overview we provide an implementation of each 'Question and Answer' example as three eqivalent implementations, namely: a C# client, a VB.NET client and a C++.NET client. Selecting one of the three folders C# , VB.NET or C++.NET; in order to view the client implementated within the corresponding language.

3. **Business Module Level**

   After selecting one of the sub-folders C# , VB.NET, C++.NET at the 'Language Level' (see (2) above) you will enter a folder which contains a sub-folder for each of the modules contained within this component package. The modules are convenient collections of business classes containing the business functionality offered by this component. Each module has a sub-folder which contains the 'Question and Answer' examples of each method of the classes which it contains.

4. **Business Class Level**

   At the business class level under the module level we are presented with one sub-folder for each of the classes within the business module. Each of these folders contains the files of the examples for each of the methods contained with the respective class.

5. **Example Level**

   At the Example level you will be presented with the files which allow you to compile and then run the examples of the application of the Business methods of the respective class. The files within this directory which constitute the Client example are as follows:

   - **Source Code File(s)** - Depending on the .NET compatible language selected at the 'Language Level' this folder will contain a source code file(s) with the extension .cs, .vb or .cpp corresponding respectively to the C# , VB.NET or C++.NET client.
   - **Configuration File** - Apart from the main source code file the folder will also contain a configuration file with the extension .conf, this file contains the information concerning the location of the DLLs which the example will require. You may add more references as needed but you should not remove the references already contained within this file.
   - **Compile Batch File** - Assuming that you have access to the relevant C# , VB.NET or C++.NET compiler the compile batch file (compile.bat) once run will compile the client example from the folders source code file. Below we include some remarks on how to obtaining suitable .NET compilers in order to be able to compile and then run these C# , VB.NET and C++.NET examples.
   - **Run Batch File** - The run batch file (run.bat) will run the examples executable file (exe) which is generated from the compilation of the source code file. If the example over runs your console screenful then press space in order to page down.

6. **Custom Example Level**

   At the Custom Example level you will find a source code file containing the implementation of the Application which addresses a typical question for which the Component is designed. The source code has a linear structure with full inline commentary. By just running the compile and then run scripts contained within this directory you will be able to solve to given problems view the results obtained. The files contained within each Custom Client Example directory are as follows:

   - **Source Code File(s)** - Depending on the .NET compatible language selected at the 'Language Level' this folder will contain a source code file(s) with the extension .cs, .vb or .cpp corresponding respectively to the C# , VB.NET or C++.NET client.

- **Compile Batch File** - Assuming that you have access to the relevant C# , VB.NET or C++.NET compiler the compile batch file (compile.bat) once run will compile the client example from the folders source code file. Below we include some remarks on how to obtaining suitable .NET compilers in order to be able to compile and then run these C, VB.NET and C++.NET examples.
- **Run Batch File** - The run batch file (run.bat) will run the examples executable file (exe) which is generated from the compilation of the source code file. If the example over runs your console screenful then press space in order to page down.

8.1.5 Remarks on .NET compilers

1. **Required compilers and how to test for them**

   In order to compile the C# , VB.NET or C++.NET examples you will require for each of these .NET languages a suitable compiler. The easiest way in which to test whether you have access to the three main .NET language compilers at your command prompt is to type the follow commands at the command line:

   csc - Invokes the C# compiler
   vbc - Invokes the VB.NET compiler
   cl - Invokes the C++.NET compiler

2. **How to obtain the required compilers**

   Compilers for the C# and VB.NET .NET compatible languages are provided within the .NET SDK v1.0 (or higher). At the time of writing within the standard Microsoft .NET SDK the C++ compiler is enabled only as a C compiler.

   In order to obtain the C# and VB.NET command line compilers you simply need to download and install the .NET SDK available from the Micrsoft site. In order to obtain a fully functional C++.NET compiler you can either download and install Microsoft's feeely available and fully functional Visual C++ Toolkit 2003, available from: http://msdn.microsoft.com/visualc/vctoolkit2003/; or use the C++ compiler which is provided with Microsoft's Visual Studio .NET IDE.

8.2 Custom QA Examples

Portfolio theory addresses the very practical aim of trying to get the highest expected return from a portfolio for a given level of risk. Within this section we detail the custom examples which we provide with the portfolio components standard QA Example, which illustrate the application of the Portfolio Component.

8.2.1 Markowitz Custom Clients

Within this section we describe the custom examples which we provide for the Morkowitz class. The source code for these examples can be found within sub-directories of the directory:

$$Clients > QAClients > \text{“}Language/Technology\text{”} > Portfolio > Markowitz$$

where "$Language/Technology$" refers the program language and/or technology (for example Java, C, Delphi, EJB, COM, Web services, CORBA etc) by which the example are written in.

1) AbsoluteRelative

This example demonstrates that whether the absolute or relative values are used for the source data of the historical returns the Portfolio Component will yield the “same” results in terms of selecting the optimal portfolio.

In particular, we will show the independence of the Weights of the Portfolio found on the Efficient Frontier for a given expected return. Moreover, we will show that whether we use absolute or relative values for the historical returns the weights of the optimal portfolio found are identical.

One reason for this independence is that the units in which the historical returns are given does not effect the weights of the portfolio of the Efficient Frontier, include the weights of the optimal portfolio is that asset weights themselves are nit-less' (in the sense of Measurement Theory). Hence intuitively speaking during the computation of the asset weights the units are being canceled out with the end result (i.e. the weights) being recording without an attached unit.

2) Efficient Subdirectory

This example demonstrates how we are able to evaluate a single Portfolio on the Efficient Frontier. That is, we demonstrate how we are able to find any portfolio of the Efficient Frontier with only one call to the underlying Rosen's optimization algorithm.

In instances when we only require to evaluate a few portfolios on the Efficient Frontier then this direct approach is not only more efficient but also more accurate.

Level of the Precision

Another aspect of this example is that we show how the portfolio found depends on the precision used. In particular, we find that a precision of 1E-12 (or higher) should be used in order to assure that the weights of the portfolio are corrects to 2 decimal places.

Historical Values Used

Within this example we use the following relative historical values of five assets historical returns over the last five periods:

- 1st Asset: 0.03, 0.04, 0.02, 0.05, 0.03
- 2nd Asset: 0.03, 0.040, 0.1, 0.03, 0.035
- 3rd Asset: 0.03, 0.032, 0.033, 0.035, 0.036
- 4th Asset: 0.02, 0.021, 0.021, 0.021, 0.0205
- 5th Asset: 0.02, 0.01, 0.02, 0.015, 0.028

Where the portfolios on the Efficient Frontier can be constructed with respect to the following asset weight constraints:

- 1st Asset Weight Constraints: Lower bound of = 0.05, Upper bound = 0.2
- 2nd Asset Weight Constraints: Lower bound = 0.05, Upper bound = 1
- 3rd Asset Weight Constraints: Lower bound = 0.05, Upper bound = 1
- 4th Asset Weight Constraints: Lower bound = 0.05, Upper bound = 0.5
- 5th Asset Weight Constraints: Lower bound = 0.05, Upper bound = 1

3) Optimal Subdirectory

With this example we select the Optimal Portfolio in accordance with Markowtiz Theory using an Investors (risk/reward) Utility Function. Here the returns are given in absolute terms.

Suppose you have the historic rates of return for 5 assets given in the table:

Month	Asset1	Asset2	Asset3	Asset4	Asset5
January	100	75	120	50	150
February	105	76	120	50	150
March	107	76.5	130	50	145
April	110	77	130	51	135
May	90	70	125	45	120
June	75	71	105	46	110
July	80	73	105	46	110
August	80	75	110	47	170
September	90	77	120	48	200
October	97	79	120	49	200
November	105	78	125	50	205
December	115	77	135	51	210

Problem: What are the asset weights of the optimal portfolio which has an expected return of 120?

That is, what are the weights of the 5 assets such that the risk of the resulting portfolio is minimized and the return is equal to 120. We have chosen to use a tolerance of $1E-3$.

Solution: The reqwuired weights are:

Asset	weight
Asset1	0
Asset2	0.01067
Asset3	0.98764
Asset4	0
Asset5	0.00169

For further details we refer the reader to the source code and in-line documentation of this example.

4) Optimal2 Subdirectory

Within this example we will consider the construction the efficient frontier for the portfolio which can be constructed from three managed fund with returns of 12 periods of:

- Fund 1: 1.14, 0.47, 0.84, 0.93, 1.1, 0.82, 0.63, 0.72, 0.8, 1.06, 0.79, 0.78
- Fund 2: -6.06, -4.15, 1.37, 4.29, 1.37, -1.21, 4.11, 2.19, -3.71, 1.18, -1.25, 1.29
- Fund 3: -1.7, -4.9, 3.64, 4.14, 1.14, 1.66, 4.18, 4.01, -0.11, 4.24, -2.26, 4.33

(The above returns are given in percentage form (i.e. 1 = 1 percent = 0.01))

Within this example we will construct the efficient frontier and then ead off' the optimal portfolio for a given level of return. We will then construct the optimal portfolio for three different investors with relatively a low, medium and high risk tolerance.

Each utility function for the three investors is given on five points represented a pairs risk and reward points on the utility curve. The three investors correspond to:

- Low Risk Tolerance Investor
- Medium Risk Investor
- Higher Risk Investor

The above returns and risk correspond to monthly expected return and corresponding risk with the three investors are prepared to accept.

5) Optimal4Risk4Return Subdirectory

Within this client we apply the Markowitz Theory to construct the portfolio which has:

- The lowest risk for a given expected return (i.e. expected performance level)
- The highest expected return (i.e. performance) for a given level of risk

for six investment funds where each of the funds historical monthly returns is known over the past 3 years (approx.).

Remark In order to address the first problem it will be sufficient to apply methods from the Markowitz class, in order to solve for risk we will need to use in addition functionality from the SolveFrontier class.

6) Optimal4Risk4Return50Funds Subdirectory

Within this client we apply the Markowitz Theory to construct the portfolio which has:

- The lowest risk for a given expected return (i.e. expected performance level)
- The highest expected return (i.e. performance) for a given level of risk

for 50 investment funds where each of the funds historical monthly returns is known over the past 3 years (approx.).

The historical values of the funds is given within a array of dimension 2 where the first elements consists of 50 members which state the returns in Jan 2001, the second element the return in Feb 2001, and so on.

Remark In order to address the first problem it will be sufficient to apply methods from the Markowitz class, in order to solve for risk we will need to use in addition functionality from the SolveFrontier class.

WARNING: This example will take several minutes to run.

7) Optimal4Risk4Return200Funds Subdirectory

Within this client we apply the Markowitz Theory to construct the portfolio which has:

- The lowest risk for a given expected return (i.e. expected performance level)
- The highest expected return (i.e. performance) for a given level of risk

for 200 investment funds where each of the funds historical monthly returns is known over the past 3 years (approx.).

The historical values of the funds is given within a array of dimension 2 where the first

elements consists of 200 members which state the returns in Jan 2001, the second element the return in Feb 2001, and so on.

Remark In order to address the first problem it will be sufficient to apply methods from the Markowitz class, in order to solve for risk we will need to use in addition functionality from the SolveFrontier class.

WARNING: This examples will take more than an hour to complete on a desktop machine.

8) EfficientAlternative Subdirectory

Within this example we evaluate using the same historical values as within the example contained within the folder 'Efficient' and evaluate the portfolio on the Efficient Frontier. The difference between this example and the example contained within the 'Efficient' folder in that here we do not use the direction approach of evaluating the portfolio, but use the approach via the evaluate of the Efficient Frontier at a number of interpolation points.

The rationale for including this examples is to allow the comparison between these two approaches to the evaluation of the optimal portfolio for a given values of the expected return.

Historical Values Used

Within this example we use the following relative historical values of five assets historical returns over the last five periods:

- 1st Asset: 0.03, 0.04, 0.02, 0.05, 0.03
- 2nd Asset: 0.03, 0.040, 0.1, 0.03, 0.035
- 3rd Asset: 0.03, 0.032, 0.033, 0.035, 0.036
- 4th Asset: 0.02, 0.021, 0.021, 0.021, 0.0205
- 5th Asset: 0.02, 0.01, 0.02, 0.015, 0.028

Where the portfolios on the Efficient Frontier can be constructed with respect to the following asset weight constraints:

- 1st Asset Weight Constraints: Lower bound of = 0.05, Upper bound = 0.2
- 2nd Asset Weight Constraints: Lower bound = 0.05, Upper bound = 1
- 3rd Asset Weight Constraints: Lower bound = 0.05, Upper bound = 1
- 4th Asset Weight Constraints: Lower bound = 0.05, Upper bound = 0.5
- 5th Asset Weight Constraints: Lower bound = 0.05, Upper bound = 1

8.2.2 Capital Market Custom Clients

Within this section we describe the custom examples which we provide for the Capital Market class. The source code for these examples can be found within sub-directories of the directory:

$$Clients > QAClients > "Language/Technology" > Portfolio > CapitalMarket$$

where "*Language/Technology*" refers the program language and/or technology (for example Java, C, Delphi, EJB, COM, Web services, CORBA etc) by which the example are written in.

1) CashConstraints Subdirectory

Within this example we illustrate how constraints may be placed onto the levels of the cash held or borrowed to the market by an optimal portfolio constructed in accordance within the Capital Asset Pricing Model. We show how the resulting continuous range of the values of the expected return and total risk can then be deduced which identify which optimal portfolios can be selected when the cash level constraints are observed.

2) Equity Subdirectory

This example illustrates the construction in accordance with the Capital Asset Pricing Model (CAPM) of the Optimal Portfolio on the CML where constraints are placed on the asset weights from which the portfolios can be constructed.

This example illustrates the application of the Capital Asset Pricing Model (CAPM) functionality implemented within the class CapitalMarket.

In particular, we seek the solution to the following question:

What is the weighting of the Market Portfolio and risk of the optimal portfolios with respect to the CAPM which can be constructed from 5 assets with historical returns of:

Time interval	Asset1	Asset2	Asset3	Asset4	Asset5
1^{st} interval	500	500	300	200	250
2^{nd} interval	450	450	320	210	270
3^{rd} interval	500	400	330	215	300
4^{th} interval	550	300	350	210	280
5^{th} interval	600	350	360	205	290

where the weights of each of the assets of the portfolio constructed must lie between 0.1 and 0.5, that is, each asset can have a minimum weights in 10 percent and a maximum weighting of 50 percent with the constructed portfolio.

Question: What if the Market Portfolio's risk and expected return? In accordance, with

respect to the CAPM what is the optimal portfolios (i.e. lowest risk with a given expected return) which has an expected absolute return of 300 and 210?

Overview of Approach

Once the source data is given we we provide to find the constrained optimal portfolio be following the below steps:

- Instantiate the Capital Market (main class) and Asset Parameter classes which provide the functionality required.
- Evaluate the covariance and expected returns of the collection of assets from which the portfolios can be constructed. This is done using two auxiliary methods from the Asset Parameters class.
- Set the constraints on the asset weights from which the portfolios are constructed.
- We then evaluate the Efficient Frontier on the entire range on which it exists and set its interpolation points to a private field.
- Next we construct the Market Portfolio.
- We then evaluate the risk and expected return of the Market Portfolio which will be used within the follow calculations.
- In order to find the optimal portfolios with respect to the CAPM we evaluate the portfolios of the CML with an expected return of 210 and 300. That is, we evaluate the weighting of the market portfolio and the risk of each of the portfolios.

Historical Source Data

The source data is we use here are the historical values of the 5 assets returns from which the portfolios on the Efficient Frontier can be constructed measured over the last 5 periods.

- 1st Asset Returns: 500, 450, 500, 550, 600
- 2nd Asset Returns: 500, 450, 400, 300, 350
- 3rd Asset Returns: 300, 320, 330, 350, 360
- 4th Asset Returns: 200, 210, 215, 210, 205
- 5th Asset Returns: 250, 270, 300, 280, 290

Business Classes used

Within this example we use the following business classes:

- Capital Market - The CapitalMarket class contain the core functionality within our implementation of the CAPM.

- Asset Parameters - The Asset Parameters class offers a number of utility methods which assist in the evaluation and estimation of the parameters of the methods within the Capital Market class.

Solution Found:

After having calculated the efficient frontier, we find the optimum market portfolio which provide the highest expected return per unit of risk. The market portfolios asset weights are as follows:

	weight
Asset1	0.08605
Asset2	0.08125
Asset3	0.0454
Asset4	0.7873
Asset5	0

Where the market portfolio has a risk of 2.42057 and has an expected return of 256.07759.

Using the market portfolio we are able to construct through either borrow cash from the market to buy more units of the market portfolio or though lending to the market; a portfolio that has any return greater than the return from cash.

Action Require to obtain a return of 300**:** The method *weightCML* in order to obtain a return of 300 returns 1.71912. This number is greater than 1, which means that a sum greater than total investor's wealth must be invested in the equity portfolio in order to have a return of 300. That is, 0.71912 of the initial sum will be borrowed at the market's rate and invested within the optimal portfolio. In this instance the risk of the portfolio increases to 4.16127.

Action Require to obtain a return of 210**:** If the investor only requires an expected return of 210, then only 0.24559 of the investors equity needs to be invested within the optimal portfolio. With the remained being lend to the market at the prevailing market rate of 195. In this instance the investors risk of the portfolio has been reduced to 0.59447.

3) SelectOptimal Subdirectory

This example illustrates the application of the Capital Asset Pricing Model (CAPM) functionality implemented within the class CapitalMarket.

In particular, we seek the solution to the following question:

What is the weighting of the Market Portfolio and risk of the optimal portfolios with respect to the CAPM which can be constructed from the assets:

- Fund 1: Historical Returns = 0.03, 0.04, 0.02, 0.05, 0.03 per year
- Fund 2: Historical Returns = 0.04, 0.045, 0.03, 0.03, 0.035 per year

- Fund 3: Historical Returns = 0.02, 0.021, 0.021, 0.021, 0.0205 per year
- Fund 4: Historical Returns = 0.02, 0.021, 0.021, 0.021, 0.0205 per year
- Fund 5: Historical Returns = 0.025, 0.027, 0.03, 0.028, 0.029 per year

where the prevailing market rate at which cash can be lent or borrowed is 1.95 percent per year?

8.2.3 Asset Parameters Custom Clients

Within this section we describe the custom examples which we provide for the Asset Parameter class. The source code for these examples can be found within sub-directories of the directory:

$$Clients > QAClients > "Language/Technology" > Portfolio > AssetParameters$$

where "$Language/Technology$" refers the program language and/or technology (for example Java, C, Delphi, EJB, COM, Web services, CORBA etc) by which the example are written in.

1) ForwardLooking Sub-directory

Suppose you have the following data regarding 3 securities.

State	1^{st} Asset	2^{nd} Asset	3^{rd} Asset	Probability
1^{st} state	100	120	200	0.1
2^{nd} state	120	130	150	0.7
3^{rd} state	130	140	140	0.2

Each cell in this table contains two values: the return of the asset in one of three states, and the probability of that state occurring.

In our example we evaluate the following:

- The expected variance of each asset:
  - for the 1^{st} asset $var = 60$
  - for the 2^{nd} asset $var = 29$
  - for the 3^{rd} asset $var = 261$
- The expected return for each asset:
  - for the 1^{st} asset exp = 120
  - for the 2^{nd} asset exp = 131
  - for the 3^{rd} asset exp = 153

- The covariance between the 1^{st} and 2^{nd} Asset is: 40

Remark Since the returns of the assets are given as absolute valuee of the expected returns evaluate will also be given in absolute values rather than in relative (i.e. percentage) terms.

Portfolio constructed with Equal Weighting between the three assets available

Suppose that you have a portfolio containing equal shares from each of the three Assets available. That is, the weighting of each of 1st, 2nd and 3rd asset is 1/3. For this constructed portfolio we evaluate the following properties:

- The expected return which is evaluated to be: 134.66667
- The portfolio's risk which is evaluated to be: 2.21108
- The variance of returns which is evaluated to be: 4.88889

8.2.4 Two Asset Portfolio Custom Clients

Within this section we describe the custom examples which we provide for the Two Asset Portfolio class. The source code for these examples can be found within sub-directories of the directory:

$$Clients > QAClients > "Language/Technology" > Portfolio > TwoAssetPortfolio$$

where "$Language/Technology$" refers the program language and/or technology (for example Java, C, Delphi, EJB, COM, Web services, CORBA etc) by which the example are written in.

1) OptimalDiversification Sub-directory

Suppose you have two assets, for which you know three possible returns may take place and you can assign corresponding probabilities for each of these returns taking place in accordance with the following table:

State	Asset1	Asset2	Probability
1^{st} state	300	500	0.5
2^{nd} state	310	400	0.2
3^{rd} state	330	550	0.3

Problem:

Say the first asset represents a portfolio which is held and the second asset represents an asset which the portfolio manager wishes to add to the portfolio such the optimal effects (i.e. minimize the risk) of diversification from the purchase is felt. The question to address here now many of shares of Asset 2 should the portfolio manager buy in order to

acheive this?

Method Used:

Our implementation which can be found within the OptimalDiversification folder when run finds the following answers. Within this implementation we first evaluate the standard deviation of each of the two assets before evaluating the vovariance from which we are able to evaluate the optimal weight using:

:TwoAssetPortfolio:weight2MinimizeRisk:

For further detail we refer the reader to the code of this example or order to see exactly how these solutions we arrived at.

8.2.5 Optimal portfolio

If you have the statistical parameters for a set of securities (the covariance matrix and the vector of expected returns), you can calculate the efficient frontier. Then, given a utility function, you can find what is the set of optimal portfolios in the sense of Markowitz model.

We use the same historic rates of return as in the previous example. The utility function is given as a set of interpolation points:

	1	2	3	4	5	6	7
x	0	40	60	80	100	120	140
y	0	6.0	6.4	7.0	14.5	22.0	29.2

There are 2 optimal portfolios:

	Portfolio 1
Asset1	0
Asset2	0
Asset3	0.269
Asset4	0
Asset5	07309

Modifying the Utility Function: You can change the utility function at any moment by calling one of the initialization functions. Then, you can calculate the new set of optimal portfolios. Suppose that you have a polynomial utility function:

$$ut(x) = 0.00117 * x^2 - 0.0443 * x$$

Again, we find 2 optimal portfolios:

	Portfolio 1	Portfolio 2
Asset1	0	0
Asset2	0.6327	0.0022
Asset3	0.0004	0.7769
Asset4	0.3667	0
Asset5	0	0.2207

8.3 Database Example with JDBC Mediator

The Database Example is located inside the *Client/Portfolio/DatabaseExample* directory. This examples is provided in order to detail how this component may be used with DBMS's. Before being able to run this examples you will need to complete the following steps:

1. **Installing our Tables (MySQL Only)**
   In order to create the database structure from which you are able to load the output results, you will need to run the 'create.sql' scripts in the 'Data' subdirectory. Open the MySQL prompt from the 'Data' subdirectory and:

   - Select (or create and then select) a database you can spare for a table named HISTORICAL_RETURNS. For example, if you have decided using the 'test' database, you would optinally type the SQL command to create it (unless it already exists):

     ```
     CREATE DATABASE test;
     ```

     and then, you would type the following to select it:

     ```
     USE DATABASE
     ```

   - Run the 'create.sql' SQL script located in the 'Data' subdirectory of the current directory by typing at the same MySQL prompt the following:

     ```
     source create.sql
     ```

     If this fails, make sure you have started the MySQL prompt from the 'Data' subdirectory (this is where the database files for this example are stored).

2. **Configuring the Database Connection**
   Edit the Java source code file by filling out JDBC information about your database, such as:

   (a) Driver Name (e.g. `com.jdbc.mysql.Driver`)

   (b) JDBC Url (e.g. `jdbc:mysql://localhost/test`)

   (c) Username and Password

If you are unsure whether you have a JDBC Driver for your Database, skip this step and try running the example with the predefined values. If that fails, you will need to probably download the latest driver.

3. **Running the example**

   Run the 'compile' script (compile.sh for Linux, compile.bat for Windows) to compile the Java source code, and then the 'run' script to see the results.

Uninstalling the Source Data

To delete all the tables used in this example, open the MySQL prompt from the 'Data' subdirectory, select the database where you created the tables, and run the following:

```
source delete.sql
```

8.4 How to use Markowitz Theory?

Within this section we provide several step-by-step guides on how to perform many of the most required tasks when apply Markowitz Theory. The methods references within this section will correspond to methods of the same name found within the Markowitz, AssetParameters or PerformanceEvaluation class's.

8.4.1 How to find the portfolio from a collection of assets that exhibit the lowest risk for a given expected future return?

The problem of finding the optimal portfolio with the lowest risk with a given expected return from a collection of assets of which the historical performance is known, involves applying two methods, namely:

- calculateEfficientFrontier - we construct the efficient frontier that consists of a continuously varying family of portfolios
- efficientFrontier - this method selects a portfolio determined by its expected return from the family of portfolios on the efficient frontier

Constructing such a portfolio is the main application of classical Markowitz Theory. That is, if you take a collection of assets historical performance from which you imply expected performance. By evaluating the covariance matrix of the assets (see section entitled, Ḧow to evaluate the covariance matrix?") and the efficient frontier (see section entitled, Ḧow to construct the Efficient Frontier?"), you may evaluate the optimal risk by applying the method:

```
efficientFrontier(double expectedReturn, int numberOfAssets)
```

from the Markowitz component.

Remark When we refer to the optimal risk we mean the minimal risk for a given expected return which is the same thing as saying the portfolio which is expected to exhibit the smallest standard deviation of its value for a given level of expected return.

8.4.2 How to construct the Efficient Frontier?

The Efficient Frontier is constructed by applying the following Markowitz class method:

```
public void calculateEfficientFrontier(double minimumExpectedReturn,
        double maximumExpectedReturn,
        double[][] covarianceMatrix,
        double[] expectedReturns,
        int numberInterpolationPoints,
        double precision)
```

The developer will need to provide the correct input parameters starting with the following two:

```
double minimumExpectedReturn
double maximumExpectedReturn
```

The minimum and maximum expected returns can be taken to be the minimum and maximum historical returns from the best performing and worst performing assets within the given collection which we will use to construct the optimal portfolio. Since these two extremes certainly suffice (i.e. allow all potential optimal portfolio's to be considered), and in the general case these extreme values may even themselves represent optimal portfolios, which in fact only be the case if the covariance matrix is the Identity matrix.

```
double[][] covarianceMatrix
```

We apply the method 'covarianceMatrix' within the AssetParameters class to the database table that contains the assets historical values from which the portfolio will be constructed. The historical data we are provided with should be stored within a database table in order to apply this method directly.

```
double[] expectedReturns
```

Please see the section entitled "How to estimate the expected return for the historical returns?"

```
int numberInterpolationPoints, double precision
```

The input value's for these parameters is just a matter of balancing speed and accuracy.

Remark The Markowitz class uses highly optimized algorithms to construct the Efficient Frontier that leads to the construction of the optimal risk-return portfolio. This allows the optimal Portfolio to be constructed in one or two minutes for 100+ assets using a commodity desktop PC. Saving the end user time and offering the convenience of being able to use a desktop PC equipment rather than specialist server hardware.

8.4.3 How to estimate the expected return from the historical returns?

Say your source data is the 'historical' monthly performance of a given fund. If we assume that the historical returns of this fund will be closely related to the future expected returns. Then we will be able to estimate the future expected returns using the method:

`expectedReturn(double[])`

from the AssetParameters class.

Further Discussion

Here as in all estimates of the fluctuation of the price (or risk) there will always be two types of risk related to the application of the model. Namely the market risk itself and the risk that the market risk (or structure) will change. The success of any model will depend on its ability to adapt to changes in the market dynamics.

Towards this end we offer a number of approaches to estimating the expected returned which one could use for a variety of differing market dynamics. These procedures are contained within the AssetParameter class and include:

- Arithmetic Historical average - Take the historical return over a given period and take the expected future return to be the arithmetic average of the historical values. Say we take the last 6 monthly historical returns, then the probability of the market states which deliver each of these 6 historically recorded returns over the last 6 months is $\frac{1}{6}$.
- ARCH Type Model - This is an averaging procedure which attaches more weight to more recently measured values. For example, we could take:

  $$\text{expectedReturn} = \sum(\exp(-i)\text{historicalReturn}[i])$$

  Say we take the last 6 monthly value again in order to estimate the expected future return. The probability of the market states that delivered each of the return found in the last six months is:

  $$\frac{\exp(-i)}{\sum_i \exp(i)}$$

for $i = 0, \ldots, 6$.

Our implementation of the ARCH model assumes that the underlying asset has a drift in accordance to the characteristic line. Under this assumption the ARCH model provides an effective model for assets, which depend linearly on stocks or stock indexes.

Remark Predictive models of ARCH/GARCH type or models involving technical or fundamental analysis are not prescriptive and hence will depend heavily on the choices of the parameters the user makes. Saying this, these models enable the user to ensure that whatever the variables used the resulting model will exhibit certain qualitative properties which are embodied within the model itself.

8.4.4 How can I measure the performance characteristics of a portfolio?

One of the core principles of 'Markowitz Theory' (and CAPM), is that investors will only accept higher risk for a higher expected return. Therefore, since we are working within this framework it is only natural to consider the risk adjusted performance of a portfolio and related methods thereof. Within this product we have including methods by which the risk adjusted and non-risk adjusted return can be evaluated.

Risk Adjusted Performance

In order to evaluate the risk adjusted return we use methods from the PerformanceEvaluation class. In particular, we use one of the following two methods:

- Sharpes Ratio - This performance measurement parameter is the most widely used. The higher the value the better the portfolios risk return profile is said to be. To evaluate Sharpes Ratio use the method `SharpesRatio`.
- Treynors Ratio - This measure takes into account the systematic risk (or beta) and the average return when assessing the overall risk adjusted return of the portfolio. To evaluate Treynor's performance measure use the method `treynorsMeasure`.

Non-risk adjusted Performance

The non-risk adjusted performance of a portfolio can be evaluated by using the methods Total Return and Portfolio Return within the performance evaluation class. The Portfolio Return method evaluates the arithmetic return of the portfolio over a period of time where the return of each asset over the period and its initial portfolio weight is known. The total return method allows you to evaluate the return form the portfolio when cash disbursements or dividends are taken into account.

Even though these measures are not risk adjusted the user can still use then in injunction

with a measure of the risk of the portfolio in order to get a better feel for a portfolios overall characteristics. The risk of a portfolio, can be evaluated by the method:

```
portfolioRisk(double[] weight, double[][] covarianceMatrix)
```

which is contained within the portfolio class.

Chapter 9

Guide to WebCab Components

9.1 The Company

WebCab is a privately owned British company that has built business solutions since its inception in 1999. We continue to refine and develop our Mathematical and Financial Framework which we have implemented on the Office, J2SE, J2EE, Delphi, COM and .NET platforms.

9.2 Presentation of Products

Our aim is to provide good quality, useful information to help you decide which component best suits your development needs. WebCab is committed to honesty and realism when presenting our products. In order to achieve this a detailed, clear and factual style for presentation is adopted within our documentation and marketing material.

9.3 Supported Clients, IDEs, Containers and DBMSs

By supporting all major development, server and client side technologies we preserve the developers flexibility in making tool and architecture decisions. In particular, each product contains detailed examples and advice concerning the integration and use of our modules within existing development tool and infrastructure platforms. In short, our documentation provides the information that the developer needs to get their applications up and running as quickly and as easily as possible.

We detail exactly how the developer can use the .NET Component, COM Component and XML Web service contained within this product with the following technologies:

- Client containers - Internet Explorer, Mozilla, Microsoft Office
- IDEs - Microsoft's Visual Studio .NET (incl. Visual C.NET, Visual Basic .NET, Visual C++.NET), Microsoft's Visual Studio 6 (incl. Visual C++, Visual Basic),

Borland's C++ Builder (incl. C++ 2005, C++BuildX), .NET Framework SDK, Office's Visual Basic Editor

- Client Side Technologies - Application Clients, ASP.NET, C#, VB.NET, C++.NET
- XML Web services - Implemented in C#, VB.NET
- DBMS - Oracle, IBM DB2, SQL Server, Sybase, MySQL
- Platforms - Windows 2003/XP/2000/NT/9x

For these technologies we include all installation scripts and template examples which will help the developer quickly and easily assemble their multi-tier enterprise application.

9.4 Transparent Functionality

All technical and business intelligence incorporated within our products is described within the associated documentation. This allows the developer to see the nature of the methodology implemented within our proprietary algorithms.

9.5 Company Culture and Activity

WebCab Components is focused solely on the production of high quality software modules within our Financial and Mathematical Framework. The WebCab team contains a wide range of expertise and experience from the academic, investment banking and software development worlds.

Within the company there exists significant internal competitive pressures with constant peer review and evaluation, resulting in higher quality products, which adhere to high professional standards and offer extended functionality.

9.6 Product Life cycle

We continuously add value to our products by evolving them according to new .NET Framework specifications, customer feedback and market demands. We give particular emphasis to incorporating our clients suggested modifications and additions into our product development cycle.

9.7 Support, Warranty and Upgrades

WebCab warrants that each component will perform substantially in accordance with the accompanying written material. We provide with all our products without charge sixty (60) days product support services including fixing bugs, compatibility issues and other

technical support issues.

All maintenance updates (including service packs) will be distributed free of additional license cost.

Dr Ben Fairfax

Founder and CEO

WebCab Components - From Developer, To Developer