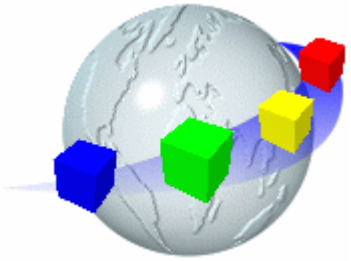
WebCab Probability and Statistics for .NET v3.3

WebCab
Components



Preface

This documentation accompanies the WebCab Statistics and Probability .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 suite. This suite includes a wide range a topics from discrete statistics, discrete probability, standard probability distributions and inferential statistics (in particular, confidence intervals and hypothesis testing). This suite consists of the following four modules:

1. Statistics - Incorporates evaluation procedures of standard quantitative measures of centrality (mean) and dispersion of discrete numerical data sets. These include weighted averages, geometric mean, Inter-Quartile range, mean and standard deviation, sample variance and the coefficient of variation.

2. Discrete Probability - Offering procedures and techniques for studying experiments containing a finite number of events where each event can have one of a finite number of outcomes.

3. Correlation and Regression - This module contains procedures for evaluating the correlation of the data set consisting of pairs of values. We also allow linear regression analysis to be carried out on such sets.

4. Standard Probability Distributions - This module contains methods for calculating the probability, the density, the mean and the variance for several special probability distributions, such as the Normal, Pareto, LogNormal and Poisson distributions.

5. Confidence intervals and Hypothesis testing - This module offers methods for one and two tailed confidence intervals or performing hypothesis testing for the mean, differences between means, proportions and differences between proportions for small and large samples.

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 suite by detailing deployment techniques on the most widely used application servers. From the third through to the seventh chapter we detail the mathematical functionality which is embedded within this suite. This represents the theoretical background of this component's implemented features. Chapter eight until twelve contains the programmer's guide containing a road map for developers to take advantage of every feature and capability on a number a development platforms.

In chapter twelve we provide details of the examples provided with this components which illustrate how features detailed within the programmer's guide can be applied in practice. Finally, we introduce WebCab Components, its philosophy and approach to serving the .NET development community with robust and powerful .NET Services.

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

This suite consists of five packages: Statistics, Discrete Probability, Standard Probability Distributions, Hypothesis Testing, and Correlation Regression which offer the following functionality.

Statistics Module

The Statistics module incorporates topic from data presentation (incl. standard, relative and cumulative frequency tables), Basic Statistics (incl. measure of centrality, dispersion and relative location) and Grouped Data (incl. Sample Mean, Variance and Standard Deviation).

Discrete Probability Module

The Discrete Probability module encapsulates the probabilistic study of finite set of events (i.e. discrete probability) and experiments with a finite number of outcomes (i.e. discrete random variables). Including: probability measures, union/intersection law, conditionals/complementary probability; cumulative distribution functions, mean/variance/expected return of Random Variable.

Correlation and Regression Module

Allows the user to investigate relationships between two variables. These finding can be used to predict one variable from the given values of other variables. We cover linear (Spearman's, t-test, z-transform) and rank (Spearman's, Kendall's) correlation, linear regression and conditional means.

Standard Probability Distributions Module

This module assists in the development of applications that incorporate the Binomial, Poisson, Normal, Lognormal, Pareto, Uniform, Hypergeometric, Weibull and Exponential probability distributions. The probability density function, cumulative distribution function and inverse, mean, variance, Skewness and Kurtosis are implemented where appropriate and/or their approximations for each distribution. We also offer methods which randomly generate numbers from a given distribution.

Confidence Intervals and Hypothesis Testing Module

Within this component we present two aspects of inferential statistics known as confidence intervals and hypothesis testing. Confidence intervals determine the level of confidence in pointwise statistics (e.g. mean, variance) of the sample in relation to the statistics for the entire population. With hypothesis testing the user can judge which of several hypotheses sampled evidence best supports.

1.1.2 Details

This suite offers the following functionality:

Statistics Module

- **Data Presentation**
  - **Frequency Tables** - Evaluate the Frequency table with respect to open left or open right boundary convention.
  - **Cumulative Frequency Tables** - Evaluate the Cumulate Frequency from above or below with respect to the open left or open right boundary convention.
  - **Relative Frequency Tables** - Evaluate the Normalized (or Relative) Frequency Table from above or below with respect to the open left or open right boundary convention.
- **Measures of Centrality**
  - **Arithmetic Mean** - a measure of centrality for quantitative data.
  - **Median** - the middle value when the observations have be ordered by magnitude
  - **Mode** - the most frequently occurring observation.
  - **Weighted Average** - the arithmetic average of a weighted set
  - **Geometric Mean** - the nth root of the product of all n numerical observations.
- **Measures of Dispersion**
  - **Range** - the difference between the largest and smallest observations.

  - **Inter-Quartile Range (IRQ)** - a measure of dispersion which is not affected by extreme values.
  - **Mean Deviation** - Evaluates the average difference between the mean of the observations.
  - **Sample Variance** - The variance from the mean of a Sample of observations.
  - **Sample Standard Deviation** - Square root of the Sample Variance which has the same units as the observations.
  - **Coefficient of Variation** - Relative value of the Standard deviation with regard to the mean.

- **Measures of Relative Location**
  - **Percentile** - The i-th percentile of a data set is the value such that at least i percent of the data set items are less than or equal to this value.
  - **z-Score** - Evaluates the number of standard deviations of a given element of the data set is from the mean.
  - **Chebyshev's Theorem** - Calculate the percentage of items that must be within a specified number of standard deviations from the mean.

- **Grouped Data**
  - **Sample Mean** - Calculate the mean of a sample of grouped data.
  - **Sample Variance** - Evaluate the variance of a sample of grouped data.
  - **Samples Standard Deviation** - Evaluate the standard deviation of a sample of grouped data.

Discrete Probability Module

- **Random Variables**
  - **Set/Get Random Variable** - Set (and get) a Random Variable to an internal tables of random variables.
  - **Associated Probability Distribution** - Evaluation of the Probability Distribution associated to the Random Variable considered.
  - **Cumulative distribution function** - of a Random Variable set or passed via parameters.
  - **Variance** - the variance of a random variable from the internal table or passed via parameters.
  - **Expected value** - the expected value of the random variable from the interval table or passed via parameters.

- **Discrete Probability**

  - **Set Probability Measure** - Set (and get) the probability measure on a discrete set.
  - **Calculate Probability** - Calculates the probability of a number of independent events.
  - **Union** - Calculates the probability of one event occurring from two collections of events.
  - **Intersection** - Calculates the probability of the intersection of two sets of events occurring.
  - **Conditional Probability** - Evaluates the probability of an event assuming that another event takes place.
  - **Complementary Probability** - Evaluate the probability of a set of events not taking place.

Correlation and Regression Module

- **Statistic quantities**
  - **Mean** - calculates the arithmetic mean.
  - **Sample variance** - calculates the sample variance.
- **Correlation coefficients**
  - **Pearson's product moment correlation coefficient** - the most widely used linear correlation coefficient for a data set.
  - **t-test, z-transform** - provides an analytic framework to establishing a confidence level for Pearson's coefficient.
  - **Spearman's and Kendall's rank correlation coefficients** - measure the association between two variables of an ordered data set.
- **Regression line** - using the method of least squares to determine the line of best fit.
- **Confidence interval for the conditional mean** - determines the confidence interval for the true regression line.

Standard Probability Distributions Module

- **Discrete Random Variables**
  - **Binomial distribution** - used to model an experiment which has two outcomes 'successes' and 'failures' of elements from a finite set.
  - **Poisson distribution** - used to model instances such as the number of cars arriving at a petrol station over 1 hour.

  - **Poisson Approximation of the Binomial distribution** - Approximation of the Binomial distribution used when the number of trial is large and the probability of is small.
  - **Hypergeometic Probability Distribution** - closely related to the Binomial probability distribution.
  - **Normal Approximation of the Binomial Distribution** - approximation of the Binomial Probability Distribution by the Normal Probability Distribution.

- **Continuous Random Variables**
  - **Normal distribution** - Used in a broad range of applications include finance (asset price evolution,...), scientific measurement,...
  - **Log Normal distribution** - Used for example when modeling investment returns and the distribution of insurance claim sizes.
  - **Pareto Distribution** - Useful for cautiously modeling the distribution of large insurance claims.
  - **Uniform Distribution** - Used to model situations where the probability is proportional to the length of the interval.
  - **Exponential Distribution** - Can be used to describe situations such as the time between arrivals at a petrol station.
  - **Weibull Distribution** - Used within the study of the reliability of precision engineering parts.

- **Numerical Methods**
  - **Extended Trapezoidal Rule** - this method is implemented in order to evaluate the non-analytic probability density functions of the Normal and Lognormal distributions.

Hypothesis Testing Module

- **Normal Confidence Interval** - used when large samples with > 30 elements are considered.
  - **Two-sided confidence interval** for the mean, proportions, difference between means and difference between proportions.
  - **One-sided confidence interval** for the mean, proportions and difference between means.
  - **Estimating the sample size** for a given confidence of the mean.
  - **Estimating the sample size** for a given confidence of the proportions.
- **Student Confidence Interval** - used when small samples with ≤ 30 elements are considered.

  - **Two-sided confidence interval** for the mean and the difference between means.
  - **One-sided confidence interval** for the mean.

- **Normal Hypothesis Testing** - used when large samples with > 30 elements are considered.
  - **Two-sided hypothesis testing** for the mean, proportions, difference between means and difference between proportions.
  - **One-sided hypothesis testing** for the mean, proportions, difference between means and difference between proportions.

- **Student Hypothesis Testing** - used when small samples with ≤ 30 elements are considered.
  - **Two-sided hypothesis testing** for the mean, proportions and the difference between means.
  - **One-sided confidence interval** for the mean, proportions and difference between means.

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 Probability and Statistics for .NET v3.3 .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.

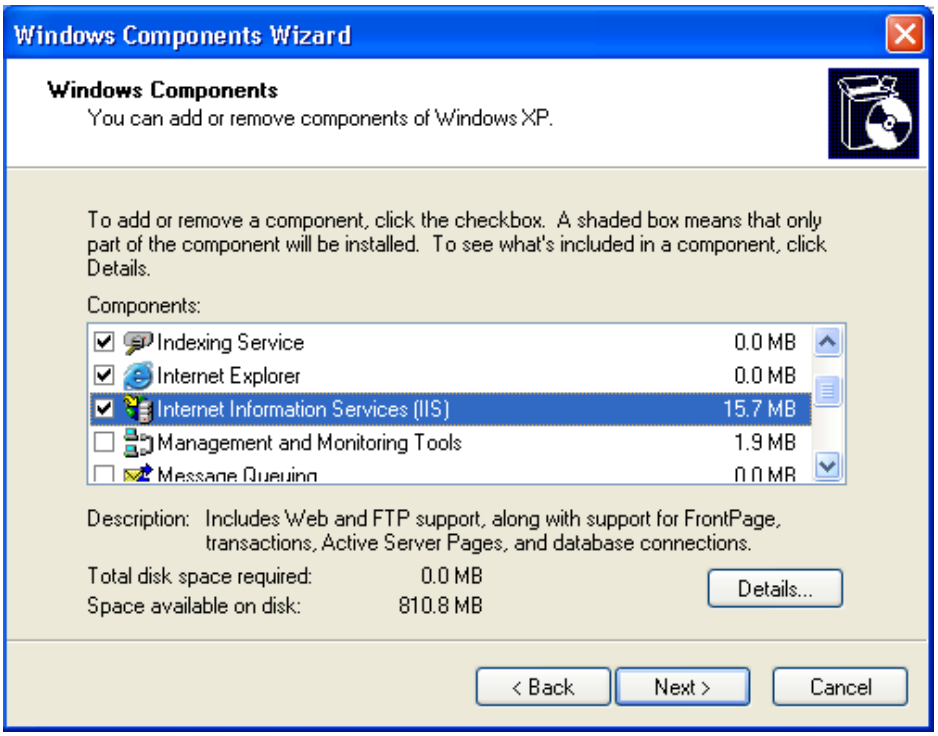
Remark By requesting the page http://localhost, the local IIS Web server will be automatically restarted if it is not already running.

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.



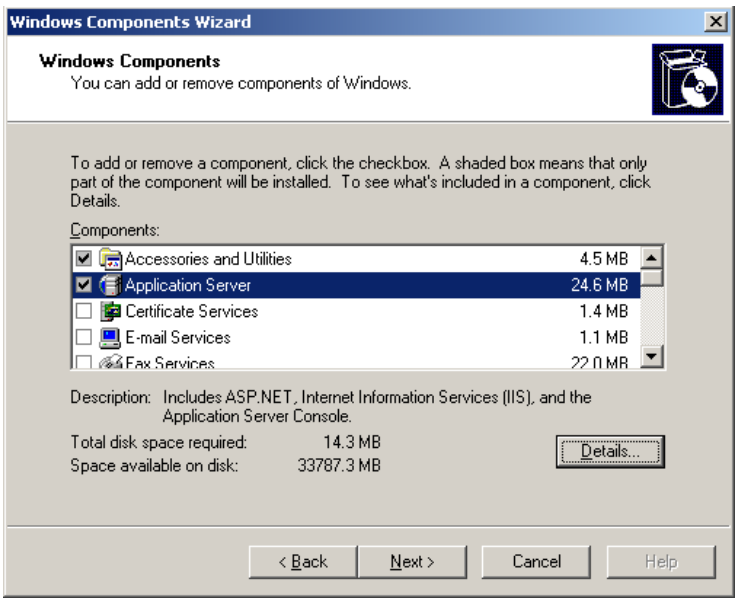
2.2.2 What Do I Need if I am Using Windows 2003?

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.

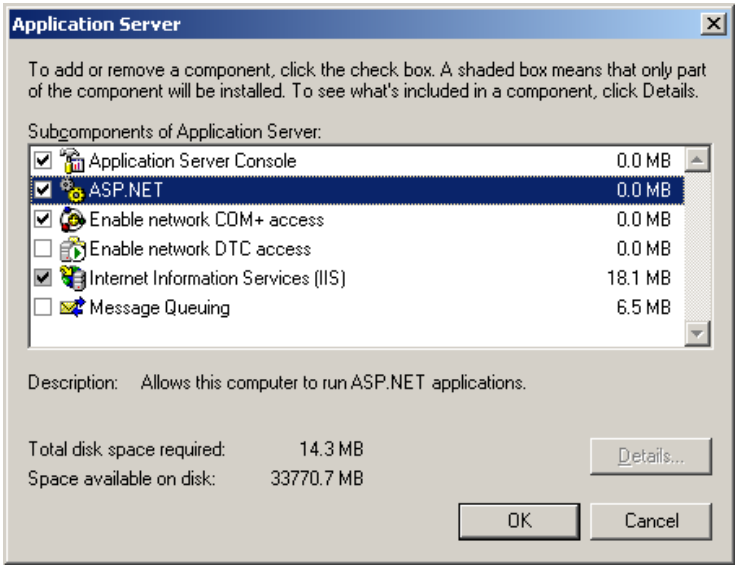
Installing the IIS Web server onto Windows 2003

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.StatisticsDemo.dll, contained within the present directory you must input the command:

```
Gacutil.exe /i WebCab.Libraries.StatisticsDemo.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 Probability and Statistics. 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 Probability and Statistics. 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

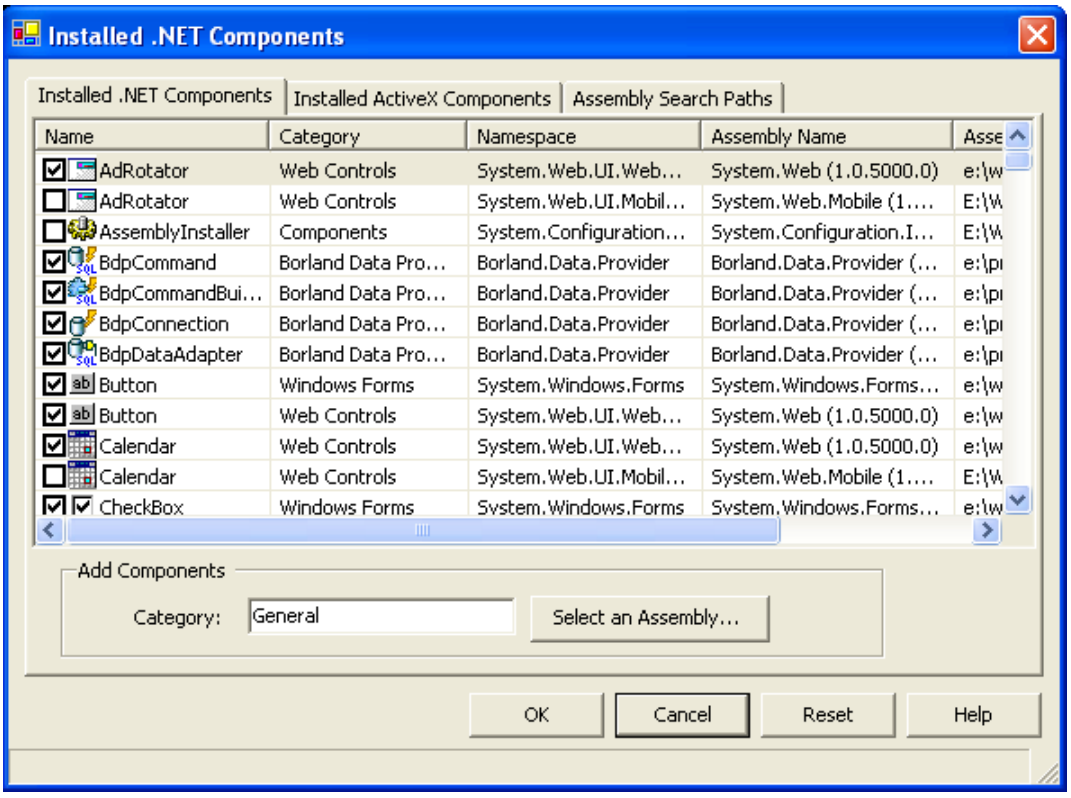
2.4.1 Using the DLL within a Visual Studio .NET project

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.

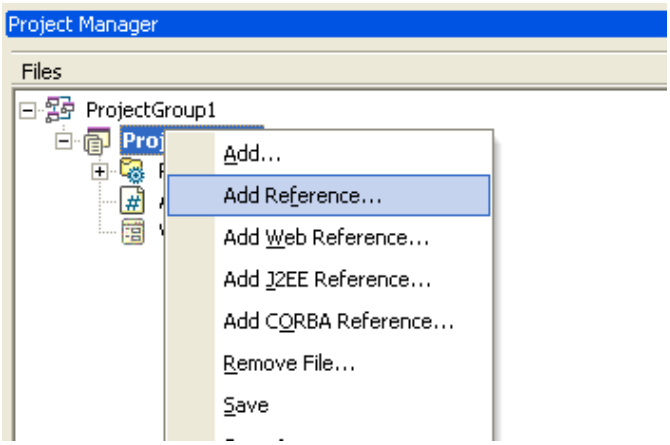
2.4.2 Using the DLL within a Borland's C# Builder project

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\Probability and Statistics`.

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 Probability and Statistics for .NET v3.3 .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 'Statistics 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/Statistics ASP.NET Examples/index.html
   ```

   In case you chose to copy the 'Statistics 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 Probability and Statistics .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 Probability and Statistics 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 'StatisticsDemo' 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/StatisticsDemo/<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:

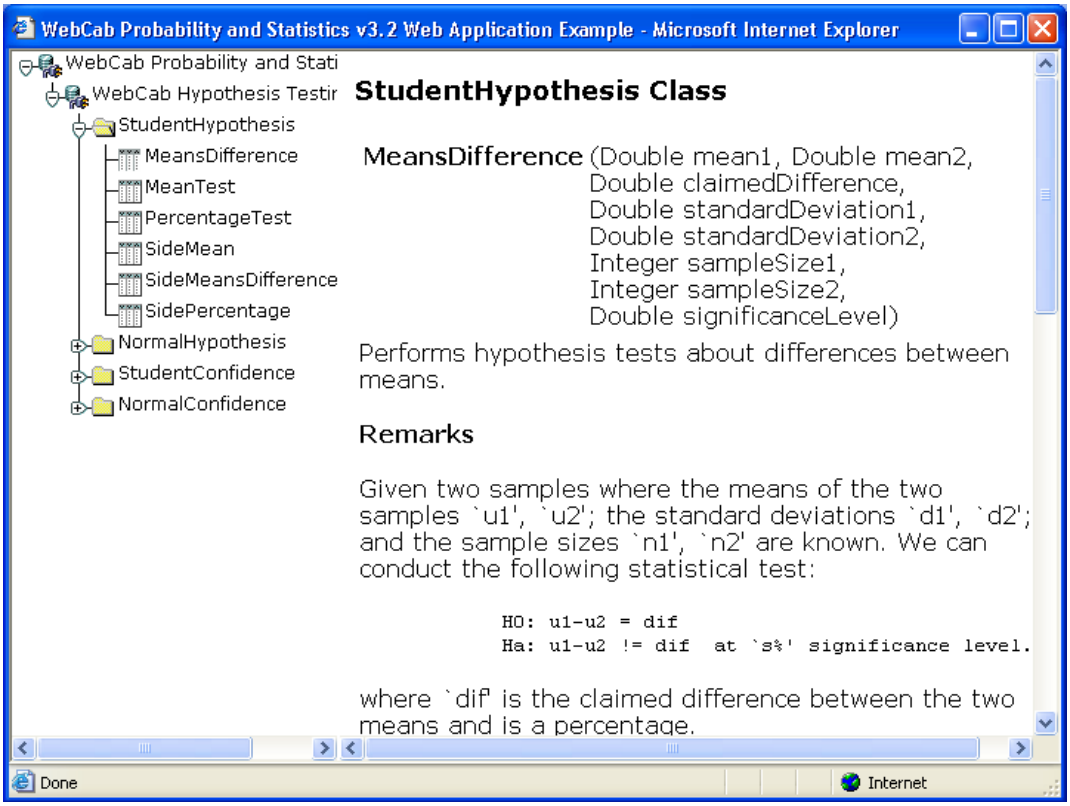


Fig: Interface of the ASP.NET online Web Application

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:

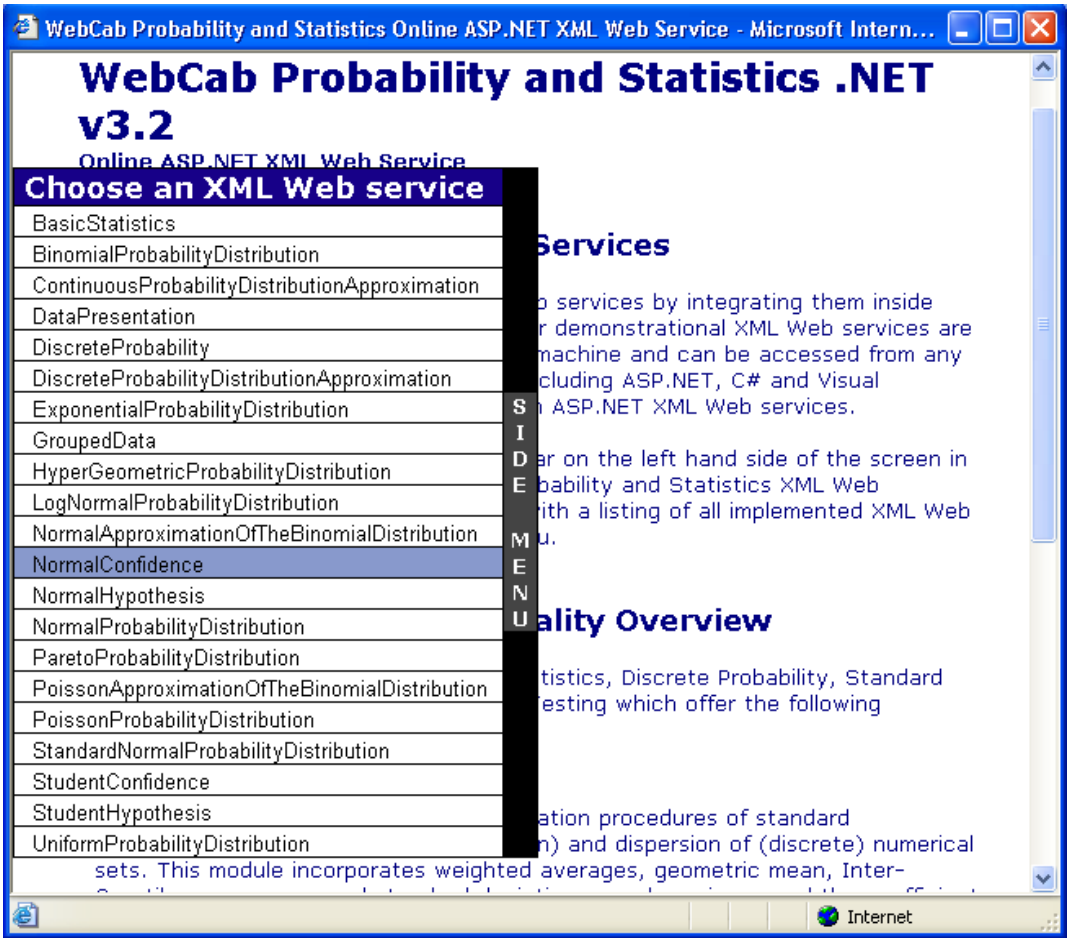


Fig: Interface of the Online Web Services

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

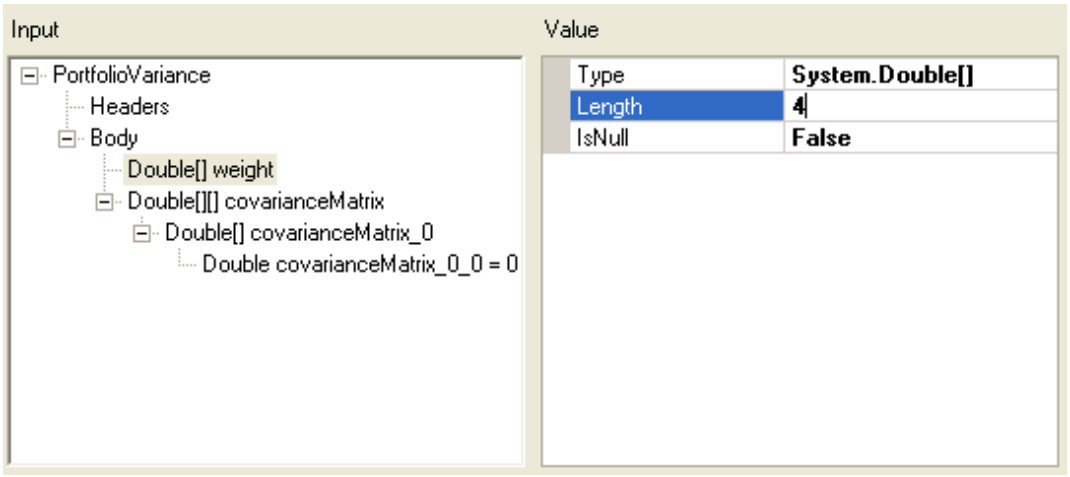
Copy from your browser's Address bar the WSDL URL to the XML Web service you have chosen to test and paste it into the `WSDL EndPoint` text field inside WebService Studio. Click the `Get` button next to this text field and wait for the tool to establish a connection to the XML Web service.

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

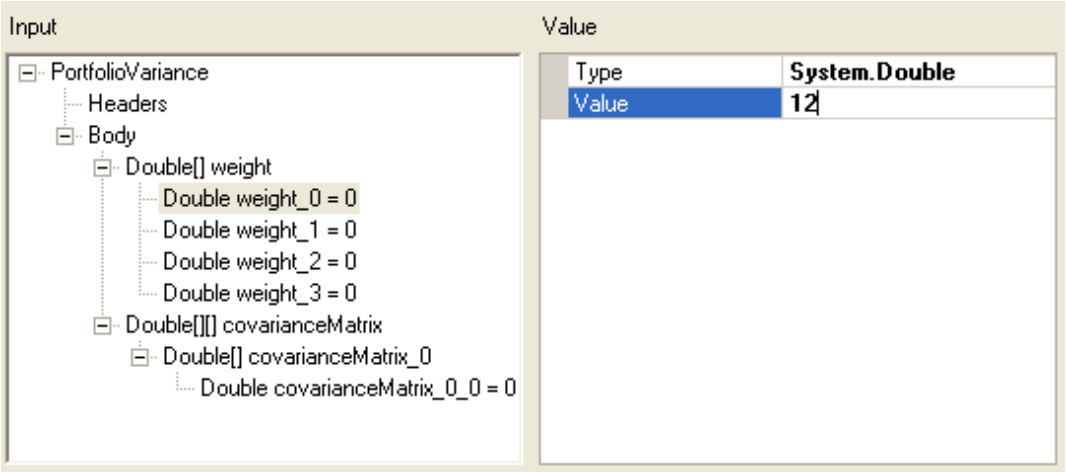
Inside the *Body* tree of the *Input* panel you may click on every parameter type and then set its value inside the *Value* panel to its right. This is accomplished by editing the right side of the *Value* column in the right hand side *Value* panel.

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

Statistics Documentation

This chapter accompanies the Statistics module of the WebCab Probability and Statistics .NET Service. Statistics is the science of collecting, presenting and analyzing data. We include formulae and methods which assist in the graphical representation of data within frequency tables. Evaluation procedures of standard quantitative measures of centrality (mean) and dispersion of (discrete) numerical sets are covered.

3.1 Data Presentation

Related to the graphical presentation of data by bars and charts are various calculations which manipulate the raw data. These procedures are usually embedded within the graphical client side charting component but here we separate them so that the developers may run these calculations on the server.

3.1.1 Frequency Tables

A raw data set particularly when its range lies within a continuum makes it difficult to detect any underlying patterns or rules. By grouping the values into a frequency table as shown below the data becomes more accessible. We detail below how the data set may to assigned to intervals of the real line for the purposes of analyzing the data set.

Given a discrete data set $D = \{d_0, d_1, \ldots, d_m\}$, and interval boundaries $\{a_0, a_1, \ldots, a_m\}$, where $a_0 < a_1 < \ldots < a_j$; we introduce two devices known as the Open Left Boundary and Open Right Boundary by which to assign the data set to a finite number of sub-intervals of the real line.

Open Left Boundary Convention

We define the frequency values (with open left boundary) as:

$$\alpha_j = \sharp\{d_i : a_j < d_i \leq a_j,\ i = 0, 1, \ldots, m\},\ \text{for } j = 0, 1, \ldots, m \tag{3.1.1}$$

$$\alpha_{-1} = \sharp\{d_i : a_0 \geq d_i,\ i = 0, 1, \ldots, m\} \tag{3.1.2}$$

$$\alpha_{m+1} = \sharp\{d_i : a_j < d_i,\ i = 0, 1, \ldots, m\} \tag{3.1.3}$$

where $\sharp$ denotes the number of elements in the set. That is, α_{-1} is the number of elements of the data set which are less than or equal to the smallest boundary value, and α_{m+1} is the number of elements which are greater than the largest boundary value.

Open Right Boundary Convention

Similarly, we define the frequency values (with open right boundary) as:

$$\beta_j = \sharp\{d_i : a_j \geq d_i > a_j,\ i = 0, 1, \ldots, m\},\ \text{for } j = 0, 1, \ldots, m \tag{3.1.4}$$

$$\beta_{-1} = \sharp\{d_i : a_0 > d_i,\ i = 0, 1, \ldots, m\} \tag{3.1.5}$$

$$\beta_{m+1} = \sharp\{d_i : a_j \leq d_i,\ i = 0, 1, \ldots, m\} \tag{3.1.6}$$

where $\sharp$ denotes the number of elements in the set. That is, $beta_{-1}$ is the number of elements of the data set which are less than the smallest boundary value, and α_{m+1} is the number of elements which are equal to or greater than the largest boundary value.

Remarks

- If the data set does not contain a value equal to one of the boundaries then the frequencies for a given set of boundary points in accordance with the open left and open right boundary convention will be the same. If one of the values is equal to a boundary point then this is not the case.
- If the data sets has several variables then we can easily construct the corresponding frequency table with several frequency variable.
- Each value within the data set will be counted exactly once.

Example

Consider the set of boundaries $\{1, 2, 3, 4, 5\}$, which divide the real line into six sub-intervals. Now if we use the open left boundary convention then the real line will be divided into the sub-intervals:

$$(-\inf, 1],\ (1, 2],\ (2, 3],\ (3, 4],\ (4, 5],\ (5, \inf)$$

Note that, each point on the real line can be assigned to one of these sub-intervals and therefore when assigning a data point to one of these intervals there will only be one sub-interval in which it belongs.

Now if we use the open right boundary convention then the real line will be divided into the sub-intervals:

$$(-\inf, 1),\ [1, 2),\ [2, 3),\ [3, 4),\ [4, 5),\ [5, \inf)$$

Therefore, if we consider the data set $\{0.5, 1.4, 1.3, 2.0, 2.3, 4.5, 5.5\}$, if the assign this data set in accordance with the above the conventions then we will have:

Using Open Left Boundary (OLB) convention

- Within the interval $(-\inf, 1]$, we assign the data element 0.5; and hence the frequency of this interval is 1.
- Within the interval $(1, 2]$, we assign the data element $1.4, 1.3, 2.0$; and hence the frequency of this interval (wrt OLB convention) is 3.
- Within the interval $(2, 3]$, we assign the data element 2.3, and hence the frequency of this interval (wrt OLB convention) is 1.
- Within the interval $(3, 4]$, we assign no data elements, and hence the frequency of this interval (wrt OLB convention) is 0.
- Within the interval $(4, 5]$, we assign the data element 4.5, and hence the frequency of this interval (wrt OLB convention) is 1.
- Within the interval $(5, \inf)$, we assign the data element 5.5, and hence the frequency of this interval (wrt OLB convention) is 1.

Using Open Right Boundary (ORB) convention

- Within the interval $(-\inf, 1]$, we assign the data element 0.5; and hence the frequency of this interval is 1.
- Within the interval $[1, 2)$, we assign the data element $1.4, 1.3$; and hence the frequency of this interval (wrt OLB convention) is 2.
- Within the interval $[2, 3)$, we assign the data element $2.0, 2.3$, and hence the frequency of this interval (wrt OLB convention) is 2.
- Within the interval $[3, 4)$, we assign no data elements, and hence the frequency of this interval (wrt OLB convention) is 0.
- Within the interval $[4, 5)$, we assign the data element 4.5, and hence the frequency of this interval (wrt OLB convention) is 1.
- Within the interval $[5, \inf)$, we assign the data element 055, and hence the frequency of this interval (wrt OLB convention) is 1.

Remark Note that the only difference for this data set when assigning it in accordance with one of these conventions is that the element 2.0, in assigned to the sub-interval $(1, 2]$, in the case when the OLB convention is used and $[2, 3)$ in the case when the ORB convention is used.

3.1.2 Relative Frequency Table

If we are comparing two or more data sets then the frequencies should be normalized to reflect the possible different sizes of the data sets themselves. To normalize a data set we much first divide the data set into a collection of classes into which the elements are assigned. Here we assign the data set in accordance with the open right boundary convention where the class frequencies are just the number of elements within each of the sub-intervals of the real line in accordance with the open right boundary convention (see example below).

To evaluate the relative frequency we apply the following formula to each class:

$$\text{Relative frequency} = \frac{\text{class frequency}}{\text{total frequency}} \tag{3.1.7}$$

where the class frequency is the number of data points within a given sub-interval of the real line, and the total frequency is the total number of elements within the data set considered.

3.1.3 Cumulative Frequency Tables

Cumulative frequency tables are suitable for continuous or discrete data sets and show the number of elements within a data set either below the highest value (or above the lowest values) of the present frequency interval. Within the following discussion we use the notation which has been introduced within the section on frequency tables and provide precise definition of the Cumulative Frequency using the four possible combinations of convention.

Cumulative Frequency Table from below with OLB

Using the above notation the Cumulative Frequency g_i, from below for a frequency table with open left boundary is given by:

$$g_j^l = \sum_{i \in \mathbb{N}: i \leq j} \alpha_i \tag{3.1.8}$$

Cumulative Frequency Table from below with ORB

Similarly, the Cumulative Frequency h_i, from below for a for a frequency table with open right boundary is given by:

$$g_j^r = \sum_{i \in \mathbb{N}: i \leq j} \beta_i \tag{3.1.9}$$

Cumulative Frequency Table from above with OLB

The Cumulative Frequency h_i, from above for a frequency table with open left boundary is given by:

$$h_i^l = \sum_{i \in \mathbb{N}: i \geq j} \alpha_i \tag{3.1.10}$$

Cumulative Frequency Table from above with ORB

The Cumulative Frequency h_i, from above for a frequency table with open left boundary is given by:

$$h_i^r = \sum_{i \in \mathbb{N}: i \geq j} \beta_i \tag{3.1.11}$$

3.2 Basic Statistics

Within this section we details the functionality which has been implemented with the class :WebCab:Statistics:Statistics:BasicStatistics:. Within this class we offer the following measures of a numerical data set:

- **Centrality:** Including Arithmetic Mean, Median, Mode, Geometric Mean, Weighted Average.
- **Dispersion:** Including Range, Inter-Quartile Range (IQR), Mean Deviation, Sample Variance, Sample Standard Deviation, Coefficient of Variation.
- **Relative Location:** Including Percentile, z-Score, Chebyshev's Theorem.

3.2.1 Measures of Centrality

Within this subsection we present the relationships which have been implemented which allow the center of a data set (in some sense) to be measured.

Arithmetic Mean

The **Arithmetic Mean** of n observations $x_1, x_2, \ldots, x_n$, is:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3.2.12}$$

Median

The **Median** is the middle value when the observations have been arranged in order of size. That is, if there are n observations then the median is the $[(n+1)/2]^{\text{th}}$ observation.

Mode

The **Mode** is the most frequently occurring observation. For example, the data set $\{1, 3, 7, 3, 1, 5, 1\}$, has a median of 1.

Geometric Mean

The **Geometric Mean** is the n^{th} root of the product of all the n observations, that is:

$$\text{Geometric Mean} = (x_1 x_2 x_3 \ldots x_n)^{\frac{1}{n}} \tag{3.2.13}$$

Weighted Average

If each sample observation x_i is associated with a weight w_i, where:

$$\sum_{i=1}^{n} w_i = 1$$

The **weighted mean** of $(x_1, \ldots, x_n)$ is given by:

$$\bar{x} = \sum_{i=1}^{n} w_i x_i \tag{3.2.14}$$

3.2.2 Measures of Dispersion

Within this subsection we present the relationships which have been implemented which allow the level of dispersion of a data set (in some sense) to be measured.

Range

The **Range** is simply the largest observation (a_l) minus the smallest observation (a_s), namely:

$$\text{Range} = a_l - a_s$$

Inter-Quartile Range

The **inter-quartile range (IQR)** is a measure of dispersion, which is not affected by extreme values. We define:

- The lower quartile is the value such that $\frac{1}{4}$ of the observations lie below it
- The upper quartile is the value such that $\frac{1}{4}$ of the observations lie above it

Then:

$$\text{IQR} = \text{upper quartile} - \text{lower quartile} \tag{3.2.15}$$

Mean Deviation (from the mean)

The **mean deviation** is given by:

$$\text{Mean Deviation} = \frac{1}{n} \sum_{i=1}^{n} |x_i - \bar{x}| \tag{3.2.16}$$

where $\bar{x}$ is the arithmetic average of $x_1, \ldots, x_n$, and $|x|$ denotes the absolute value of x.

Sample Variance

The **sample variance**, s^2 is given by the following expressions:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{3.2.17}$$

where $\bar{x}$ is the arithmetic average of $\{x_1, \ldots, x_n\}$, the data set being considered.

Sample Standard Deviation

The units of the sample variance are the squares of the observation units. This is rather inconvenient and so we usually quote the square root of the sample variance, known as the **sample standard deviation**, which has the same units as the observations from which it was derived.

$$s = \sqrt{\text{Sample variance}} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{3.2.18}$$

where $\bar{x}$ is the arithmetic average of $\{x_1, \ldots, x_n\}$, the data set being considered.

Coefficient of variation

The **Coefficient of Variation** measures to the relative values of the Standard Deviation with regard to the mean of the data set considered. Note that the mean is a general proxy for the approximate magnitude of the elements of the data set being considered. The Coefficient of Variance is given by the following expression:

$$\text{Coefficient of variation} = \frac{\text{standard deviation}}{\text{mean}} * 100 \tag{3.2.19}$$

Remark Clearly the Coefficient of Variation is undefined if $\bar{x} = 0$, and generally will be of little value if the mean is close to zero.

3.2.3 Measures of Relative Location

Within this subsection we present the relationships which have been implemented which allow the relative location of an element of collection elements within a data set (in some sense) to be measured.

Percentile

The i-th percentile of a data set is the value such that at least i percent of the data set items are less than or equal to this value, or equivalently if at least $(100 - interest)$ percent of the items are greater than or equal to this value.

Remarks

- The 50-th percentile is the measure of centrality known as the median of a data set (defined above).
- The 25-th, 50-th, 75-th percentiles are often referred to as the 1-st, 2-nd (median) and 3-rd quartile respectively. The term quartile refers to the fact that these three values will roughly divide the data set considered into four equal parts.

z-Score

Evaluates the z-score (often referred to as the standardized value) of an element x_i of the data set considered. The z-score denotes the number of standard deviations of a given element of the data set is from the mean of the data set. Formally, the z-Score of a element x_i is given by:

$$\frac{x_i - \bar{x}}{\text{Standard Deviation}}$$

where $\bar{x}$ is the mean of the data set considered and Standard Deviation, is the standard deviation of the data set defined in the preceding subsection.

Chebyshev's Theorem

Using a direct application of Chebyshev's Theorem we evaluate the percentage of items that must be within a specified number of standard deviations from the mean. Chebyshev's Theorem states the following:

> **Chebyshev's Theorem:** At least $(1 - \frac{1}{z^2})$ of the items in any data set must lie within z standard deviations of the mean of the same data set when z is any real number greater than 1.

Example: If we take the number of standard deviation to be 2, then by applying Chebyshev's Theorem we are able to evaluate the proportion of the data set which lies within the interval $[m - (2 * SD), m + (2 * SD)]$, where m is the arithmetic mean of the data set considered and SD is the standard deviation of the data set considered. Note that if A is the proportion returned then the corresponding percentage of items within the interval is $(A * 100)$.

3.3 Grouped Data

Within this section we details the functionality which has been implemented with the class :WebCab:Statistics:Statistics:BasicStatistics:. Within this class we offer measures of:

- Sample Mean of Grouped Data
- Sample Variance of Grouped Data
- Sample Standard Deviation of Grouped Data

3.3.1 Example of a Grouped Data Set

In order to assist in the analysis of a given data set it is often convenient (and in fact often necessary) to collect the data into categories. For example, consider a data set which consists of the heights of a set of 100 people. Now in order to derive qualitative properties concerning this data set it is natural to group the data set into height intervals (i.e. such intervals could be 0m - 1.20m, 1.21m - 1.50m, 1.51m - 2.00m, 2.01m - 3.00m. For each interval the number of members (i.e. frequency) of the original data set which fall within the interval would be recorded. Moreover the midpoints of the intervals are: 0.60, 1.35, 1.75, 2.50; respectively.

Say for a given collection of people in accordance with the above grouping (i.e. classes) we have:

- Height 0m - 1.20m, Number of people (i.e. frequency) within this group (or class) is equal to 5
- Height 1.21m - 1.50m, Number of people (i.e. frequency) within this group (or class) is equal to 25
- Height 1.51m - 2.00m, Number of people (i.e. frequency) within this group (or class) is equal to 30
- Height 2.01m - 3.00m, Number of people (i.e. frequency) within this group (or class) is equal to 3

3.3.2 Quantitative Measures of Grouped Data

Within this subsection we describe means by which the Sample Mean, Sample Variance and Sample Standard Deviation of a Grouped Data Set can be evaluated.

Sample Mean of Grouped Data

In a grouped frequency table with k class intervals, where x_i refers to the mid-point of the i^{th} interval and f_i to the frequency in that interval, the **Sample Mean** for this Grouped Data set is:

$$\bar{x} = \frac{\sum_{i=1}^{k} x_i f_i}{\sum_{i=1}^{k} f_i} \tag{3.3.20}$$

Sample Variance of Grouped Data

Using the notation as above the Sample Variance is given by:

$$\text{Sample Variance} = \frac{\sum_i f_i (x_i - \bar{x})^2}{n-1} \tag{3.3.21}$$

where $\bar{x}$ is the Sample Mean of the Grouped Data and n is the Sample Size used.

Sample Standard Deviation of Grouped Data

Using the notion as above the Sample Standard Deviation is given by:

$$\text{Sample Standard Deviation} = \sqrt{\text{Sample Variance}} \tag{3.3.22}$$

Chapter 4

Discrete Probability Documentation

Within this chapter and the associated module we consider the (probabilistic) study of experiments with a finite number of events, where each event can have one of a finite number of possible outcomes. This study is generally referred to as Discrete Probability.

4.1 Random Variables

As mentioned above the theory of Discrete Probability, is the study of experiments with a finite number of events where each of these events can have one of a finite number of possible outcomes. Here we will describe one approach to the study of such experiments using random variables. A (discrete) Random Variable is a numerical description of the outcomes of such an experiment. That is, a **random variable** associated to a (finite) experiment, is a mapping of a finite set of events (with numerical identifiers) onto their associated outcomes which also have a numerical identifier.

Formally speaking a random variable $X : E \to S$, of a discrete set of events E, is a function which maps the events E, onto a discrete index set S. In the following discussion we will often use this formal notation (due to efficiency) however sight of the origin and motivation of random variables should never be far from the users mind.

Note: In the following discussion where formal notation is used we will always take $S = 0, 1, \ldots, N$ or $1, 2, \ldots, N$, for some positive integer N.

4.1.1 Examples of Random Variables

Below we provide two examples of experiments which can be studied by identifying variables with the experiment with numerical values. Within the commuter example given below the outcomes of the experiment is the magnitude of the time taken in order to perform a daily commute, whereas in the second nursery example the random variable describes the **number** of children within a given age group. The way in which the random variable can be used to describe the experiment will depend on the nature of the experiment. Moreover, if there is a degree of choice in the association of the random variable with the experiment

considered then the choices should be made in view of the particular properties of the experiments you wish to study.

Commuter Example

An example of a random variable could correspond to the days of a given working week (i.e. 1, 2,...,5) and the number of minutes a daily commute to work took. For example,

- Monday's commute took 28 minutes
- Tuesday's commute took 30 minutes
- Wednesday's commute took 26 minutes
- Thursday's commute took 32 minutes
- Friday's commute took 30 minutes

The random variable denoted by f, for this experiment would correspond to:

- $f(1) = 28$
- $f(2) = 30$
- $f(3) = 26$
- $f(4) = 32$
- $f(5) = 30$

Recall that a random variable is a numerical description of an experiment, and therefore in the above example we have made the following associations:

- Monday $\longleftrightarrow$ 1, i.e. the first day of the experiment.
- Tuesday $\longleftrightarrow$ 2, i.e. the second day of the experiment.
- Wednesday $\longleftrightarrow$ 3, i.e. the third day of the experiment.
- Thursday $\longleftrightarrow$ 4, i.e. the fourth day of the experiment.
- Friday $\longleftrightarrow$ 5, i.e. the fifth day of the experiment.

The fact that random variables are numerical descriptions of experiments allows us to apply numerical techniques in order to study the associated experiments. Clearly different experiments will be represented by differing random variables, however it is also the case that the same experiment can justifiably be represented by two different random variables. However, the way in which the random variable describes the experiment will effect the interpretation and even applicability of the numerical techniques (for example, the evaluation of the mean, variance and rho statistic).

Nursery Example

Considering the experiment of counting the number of children of the ages $1, 2, 3, 4$ or 5, within a nursery. Say the random variable f, for this experiment is:

- $f(1) = 28$, i.e. the number of 1 year old children is 28.
- $f(2) = 30$, i.e. the number of 2 year old children is 30.
- $f(3) = 26$, i.e. the number of 3 year old children is 26.
- $f(4) = 32$, i.e. the number of 4 year old children is 32.
- $f(5) = 30$, i.e. the number of 5 year old children is 30.

Here we have identified the number of children of a given age as the outcome of the experiment. Therefore, the random variable used here is a mapping which maps an integer corresponding to a age group of children within the nursery to the number of children within the nursery of that age.

4.1.2 Associated Probability Distribution

Each random variable has an associated probability distribution pp, function where the magnitude or number of the outcomes of the experiment is replaced with the percentage size a given outcome has in the set of all possible outcomes. As mentioned above the interpretation of the probability distribution will depend on the nature of the random variable. Within our two examples given the associated probability distribution would be described as follows:

- Commuter Example: The associated probability distribution maps the journey time on a particular day to the percentage of the total commuting time used during the period considered. That is, if a random moment is selected during the commuting period then the probability distribution returns the probability of that moment being within a given day
- Nursery Example: The associated probability distribution maps an age group to the probability of a randomly selected child from t he nursery is a member of that age group.

Formally, the associated probability distribution $p : E[0.1]$, of a random variable $X : E \to S$, of a discrete set of events E, onto a discrete index set S, is given by:

$$p(a) = \frac{R(a)}{\sum_{x \in S} R(x)},$$

where $a \in E$.

Evaluating the Associated Probability Distribution for the nursery example

The associated (discrete) probability distribution p, for these results which gives the probability of a child randomly selected from the nursery taking a given age is:

- $p(1) = \frac{28}{(28+30+26+32+30)}$, i.e. probability of a child being 1 years old.
- $p(2) = \frac{30}{(28+30+26+32+30)}$, i.e. probability of a child being 2 years old.
- $p(3) = \frac{26}{(28+30+26+32+30)}$, i.e. probability of a child being 3 years old.
- $p(4) = \frac{32}{(28+30+26+32+30)}$, i.e. probability of a child being 4 years old.
- $p(5) = \frac{30}{(28+30+26+32+30)}$, i.e. probability of a child being 5 years old.

4.1.3 Cumulative Distribution Function

Cumulative distribution function (c.d.f), denoted $F : E \rightarrow [0, 1]$, is just the sum of the probability distribution for all value less than or equal to a given evaluation point.

Formally, the cumulative distribution function is given by:

$$F(x) = \sum_{x_i \leq x : x_i \in E} p(x_i), \tag{4.1.1}$$

where p is the probability distribution associated to the underlying random variable as defined in the previous subsection.

Evaluating the Cumulative Distribution Function of the Nursery Example

Here we illustrate the evaluation of the associated cumulative probability distribution $g : 1, 2, 3, 4, 50, 1]$, of the nursery example defined above. The cumulative distribution function, in this case takes the values:

- $g(1) = p(1)$, the probability of a child being a 1 year old.
- $g(2) = p(1) + p(2)$, the probability of a child being 2 years old or younger.
- $g(3) = p(1) + p(2) + p(3)$, the probability of a child being 3 years old or younger.
- $g(4) = p(1)+p(2)+p(3)+p(4)$, the probability of a child being 4 years old or younger.
- $g(5) = p(1) + p(2) + p(3) + p(4) + p(5) = 1$, the probability of a child being 5 years old or younger. Note that since all children in the sample are 5 years old or younger the sum must be unity.

The value which is returned by this method is precisely the value of the corresponding function g. Moreover, we allow the evaluation point to be set equal to any double and therefore for example if we set the evaluation point to be equal to 2.5, then the values corresponding to $g(2)$, rather than $g(2.5)$ will be returned.

4.1.4 Expected Values of a Random Variable

Here we consider the Expected Value of a Random Variable which is the weighted arithmetic average of the possible outcomes. Returning to the nursery example used above the mean of the random variables which describes the nursery experiment corresponds to the average age of a child within the nursery.

Formally, if we assume that the random variable X must have one of the values $x_1, x_2, \ldots, x_k$, where the associated probability distribution takes the values $p(x_1), p(x_2), \ldots, p(x_k)$ where:

$$\sum_{i=1}^{k} p(x_i) = 1$$

Then the **expected value of the random variable** E, is given by:

$$E[X] = \sum_{i=1}^{k} x_i p(x_i) \tag{4.1.2}$$

In general, if $g(X)$ is a function of the random variable X,

$$E[g(X)] = \sum_{i=1}^{k} g(x_i) p(x_i) \tag{4.1.3}$$

Auxiliary Remarks

If $g(X)$ and $h(X)$ be functions of a random variable X, let Y be another random variable and let c and k be constants, then:

- $E[cg(X)] = cE[g(X)]$
- $E[g(X) + c] = E[g(X)] + c$
- $E[g(X) + h(X)] = E[g(X)] + E[h(X)]$
- $E[cX + kY] = cE[X] + kE[Y]$

If X and Y are independent random variables then:

- $E[XY] = E[X]E[Y]$

4.1.5 Variance of a Random Variable

Here we consider the variance of a Random Variable which is roughly speaking the weighed sum of the squares of the differences from the mean of each of the values taken. Intuitively speaking, the variance is a measure of the amount by which the values which the random variable takes deviate from its mean. Returning to the nursery example (discussed in the above sections) the variance of the associated random variable describes to what degree

the we would expect a randomly selected child's age to differ from the mean of the ages of the children within the nursery.

Formally, the **variance of a random variable** σ^2, is given by:

$$\sigma^2 = \sum_{x \in E} (x - \mu)^2 p(x) \tag{4.1.4}$$

where μ is the mean of the random variable, E is the range of the underlying random variable and $p(x)$ is the value of the associated probability distribution.

4.2 Discrete Probability

Within this section we introduce the foundations of discrete probability which has been implemented within the Probability class of the Discrete Probability module. This includes discrete probability measures, the addition law, union and intersection laws, and conditional/complementary probability.

4.2.1 Calculating the Discrete Probability

The definition and evaluation of the probability of a set can be presented on a apriori or relative frequency basis.

Apriori Definition

If an experiment has n equally likely outcomes, the probability of an event, E, denoted by $P(\mathrm{E})$, is given by:

$$P(\mathrm{E}) = \frac{\text{Number of different ways in which } E \text{ can occur}}{n} \tag{4.2.5}$$

Relative Frequency Definition

The probability of an event is the relative frequency of its occurrence in the long run:

$$P(\mathrm{E}) = \frac{\text{Number of times E occurred}}{\text{Number of trials}} \tag{4.2.6}$$

The set of all possible outcomes of an experiment is called the *sample space*, denoted by ϵ.

Notation We denote the union of two sets A and B by $A \cup B$, and the intersection by $A \cap B$, and the empty set by $\emptyset$.

4.3 Basic Laws of Probability

We start by introducing the following notation. The set of all possible outcomes of an experiment is called the *sample space*, denoted by ϵ. We denote the union of two sets A and B by $\mathrm{A} \cup \mathrm{B}$, and the intersection by $\mathrm{A} \cap \mathrm{B}$, and the empty set by $\emptyset$.

4.3.1 The Addition Law

Using the above notation the addition law is given by:

$$P(\text{A or B}) = P(\text{A}) + P(\text{B}) - P(\text{A} \cap \text{B}) \tag{4.3.7}$$

In particular, if A and B are independent i.e. $\text{A} \cap \text{B} = \emptyset$, then:

$$P(\text{A or B}) = P(\text{A}) + P(\text{B})$$

4.3.2 Conditional Probability

The conditional probability of A given that event B occurs is denoted by $P(\text{A}|\text{B})$, where:

$$P(\text{A}|\text{B}) = \frac{P(\text{A} \cap \text{B})}{P(\text{B})} \tag{4.3.8}$$

Therefore, if $\text{A} \cap \text{B} = \emptyset$, then:

$$P(\text{A}|\text{B}) = P(\text{A})$$

and the multiplication rule reduces to:

$$P(\text{A} \cap \text{B}) = P(\text{A})P(\text{B}) \tag{4.3.9}$$

Remark The above formulae can be used to test for the independence of two sets of events.

4.3.3 Complementary Events Probability

The complementary events probability of A, denoted $P(\text{A}')$ is given by:

$$P(\text{A}') = 1 - P(\text{A}) \tag{4.3.10}$$

Chapter 5

Correlation and Regression Documentation

Within this chapter we describe in detail the numerical statistical procedures which will be applied to the problem of measuring the correlation and linear regression between two variables. Where appropriate we will motivate and justify the applicability of the procedures introduced.

5.1 Overview

To give the reader a global perspective of our chosen collection of statistical procedure and how these form a justifiable basis from which one can make conclusions. We consider the notion of correlation and well as remark on related issues which may effect the validity of any results found.

The notion of 'correlation' is a measure of the degree to which two variables (possibly more) are moving in step. Here we will chiefly be interested in linear correlation, that is forming a methodology from which we can conclude with a 'Reasonable' level of confidence that two variables exhibit a linear relation.

Remark If two variables are correlated it does not necessarily imply that there exists a functional relationship between them. For example they might move in step with a third variable.

5.2 Significant Figures

This component will produce results which are accurate to 16 significant figures given sufficiently good initial data. Below to detail two influences:

- Computational Accuracy - When the implementation of the computational procedures where performed all numerical values where represented as doubles. Hence no signif-

icant loss of accuracy will occur during the computational procedure and given good initial data the component is able to return results to at least 13 significant figures.

- Input Data - If the input data is accurate to n significant figures then the component will return correlation results which can be claimed to $n-2$ significant figures.

5.3 Linear Correlation

We will now introduce models from which we can detect in a given sense whether a linear relationship exists between two variables.

5.3.1 Pearson's Product Moment Correlation Coefficient

For a given numerical data set $(x_i, y_i), i = 1, \ldots, N$, a natural way (illustrated below) in which to measure correlation is Pearson's Product Moment Correlation Coefficient, which we will denote by r. This measure is widely used and is often referred to as the sample correlation component.

Motivation

When first thinking about how one might measure the 'Correlation' between two variables it may seem natural to consider the covariance, that is:

$$\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}),$$

where $\bar{x}$ and $\bar{y}$ are the arithmetic averages of the numerical data sets under consideration.

The problem with using the covariance as a measure of correlation is that is depends upon the way in which the values within the data set are measured. It may appear that a large positive value for the covariance implies a strong positive linear relationship. But the user should be careful because the *covariance* will depend upon the units which are being used for the measurement of the two variables. For example, say we are interested in investigating the relationship between the height x and the weight y of a set of people. Clearly the relationship should be the same whether we measure height in feet or inches. When height is measured in inches, however, we get a much higher numerical value for $(x_i - \bar{x})$ than when it is measured in feet. Thus, with height measured in inches we would obtain a larger value for the covariance. Hence their result is not invariant of the choice of units. By factoring out this problem of units we are lead directly to what is known as Pearson's coefficient or the linear correlation coefficient.

The Formulae

The linear correlation coefficient r, is given by the formula:

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N s_X s_Y} \tag{5.3.1}$$

where s_X and s_Y are the sample deviation for the two variables. If we expand these terms, we get the following expression:

$$r = \frac{N\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i}{\sqrt{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}\sqrt{N\sum_{i=1}^{N} y_i^2 - \left(\sum_{i=1}^{N} y_i\right)^2}} \tag{5.3.2}$$

Remarks

- The coefficient r lies in the interval $[-1, 1]$
- The variables x and y are *uncorrelated* if the value of r is ear' to zero
- The following terminology is often used:
  - *complete positive correlation* if $r = 1$
  - *complete negative correlation* if $r = -1$

5.3.2 Significance Test for Pearson's Coefficient, r

The data used to calculate the correlation coefficient are samples from a population of possible observations, so r is subject to sample fluctuations. The entire populations correlation coefficient, ρ is generally unknown.

When a correlation is known to be significant Pearson's coefficient r, is one conventional way of summarizing its *strength*. But since in most instances a sample of the entire population is used to determine the correlation of the entire population. We introduce methods which allow the user to determine to what degree results for the sample hold for the entire population.

Remark Questions and methods related to significance testing of point wise statistics falls within the area of statistics known as statistical inference. This topic is covered is much greater depth within the component WebCab Confidence Intervals and Hypothesis testing. For more details, motivation and examples concerning hypothesis testing please feel free to download the demo version of this product.

We will allow hypothesis testing methods against the null hypothesis that our variables x and y are uncorrelated. Thus, we will test the null hypothesis:

$$H_0 : r = 0$$

against the alternative hypothesis:

$$H_A : r \neq 0$$

where r is Pearson's correlation coefficient.

t Test Statistic

If the number of data points N is not large, then the test statistic is:

$$t = r\sqrt{\frac{N-2}{1-r^2}} \tag{5.3.3}$$

Remark Under the null hypothesis of there being no correlation present the t test statistic follows the Student's t distribution, with $N-2$ degrees of freedom.

Fisher's z-transformation

If N is only moderately large (i.e. ≥ 10) we compare whether the difference of two significantly nonzero r from different experiments is itself significant. To do this we use *Fisher's z-transformation* to associate each measured r with a corresponding z,

$$z = \frac{1}{2}\ln\left(\frac{1+r}{1-r}\right) \tag{5.3.4}$$

Remark If the joint probability distribution of the variables x and y is 'too different' from a binomial distribution then it is not possible to give meaningful interpretations of the Pearson's coefficient statistic r.

5.4 Non-parametric or Rank Correlation

The uncertainty in interpreting the significance of the linear correlation coefficient, r leads to the concept of *non-parametric or rank correlation*. We replace the value of each x_i by the value of its rank among the other x_i in the sample. We use exactly the same procedure for the y_i's, replacing each value by its rank among the other y_i's in the sample. If some of the x_i's have identical values, it is conventional to assign to all these *ties* the mean of the ranks that they would have had if their values had been slightly different. This mid rank will sometimes be an integer, sometimes a half-integer. Non-parametric correlation is more robust than linear correlation, more resistant to unplanned defects in the data, in the same sort of sense that the median is more robust than the mean.

5.4.1 Spearman's Rank Correlation Test

To measure the association between the variables x and y we consider the Spearman's Rank Correlation coefficient, r_s, for when the data are defined by their position. Thus, the data can be placed in order but is not given an absolute size. Let r_i be the rank of x_i

among the other x's, s_i be the rank of y_i among the other y's. The rank-order correlation coefficient is defined to be the linear correlation coefficient of the ranks, namely,

$$r_s = \frac{\sum_{i=1}^{N}(r_i - \bar{r})(s_i - s)}{\sqrt{\sum_{i=1}^{N}(r_i - r)^2}\sqrt{\sum_{i=1}^{N}(s_i - s)^2}} \tag{5.4.5}$$

The significance of a nonzero of r_s is tested by computing:

$$t = r_s\sqrt{\frac{N-2}{1-r_s^2}} \tag{5.4.6}$$

which is distributed approximately as Student's distribution with N - 2 degrees of freedom.

We denote by D the *sum squared difference of ranks*:

$$D = \sum_{i=1}^{N}(r_i - s_i)^2 \tag{5.4.7}$$

When there are no ties (i.e. the data set has no two identical entries) in the data, then the exact relation between D and r_s is:

$$r_s = 1 - \frac{6D}{N^3 - N} \tag{5.4.8}$$

When there are ties the exact relation between D and r_s is slightly more complicated.

Let f_k be the number of ties in the k^{th} group of ties among the r_i 's, and let g_m be the number of ties in the m^{th} group of ties among the s_i's. Then we have the formula:

$$r_s = \frac{1 - \frac{6}{N^3-N}\left[D + \frac{1}{12}\sum_k(f_k^3 - f_k) + \frac{1}{12}\sum_m(g_m^3 - g_m)\right]}{\left[1 - \frac{\sum_k(f_k^3-f_k)}{N^3-N}\right]^{\frac{1}{2}}\left[1 - \frac{\sum_m(g_m^3-g_m)}{N^3-N}\right]^{\frac{1}{2}}} \tag{5.4.9}$$

Remark It is useful to calculate the significance level using (5.4.8) and (5.4.9), when there are ties in order to get some feel for the level of accuracy which this method will provide us with. Note that a small perturbation of the underlying data set could result in all the ties being lost. Hence, by calculating both figures and comparing them we can see to what degree the final outcome has been effected by the existence of 'equal values' within the data set. The smaller the difference the more rigidity the data set exhibits with regard to ordering in the sense of Spearman's test.

5.4.2 Kendall's Rank Correlation Coefficient

A somewhat different approach to the problem of agreement between two rankings is given by the τ coefficient (small Greek tau) due to M.G. Kendall. The Kendall coefficient relies

upon the following principle of inversion:

or a data set which is ranked according to two ordering mechanism. An inversion (in ordering) exists between *any pair* of individuals b and c when $b > c$ in one ranking and $c > b$ in the other.'

Instead of treating the ranks themselves as though they were scores and finding a correlation coefficient as in the case of Spearman's coefficient, in the computation of τ we depend only on the number of inversions in order for the pairs of individuals in the two rankings.

We have the following definition of the τ statistic:

$$\tau = 1 - \frac{2(\text{Number Of Inversions})}{\text{Number Of Pairs Of Objects}} \tag{5.4.10}$$

Remark When two rankings are *identical* there are no inversions in order needed and so in this case $\tau = 1$. On the other hand when one ranking is exactly the reverse of the other every pair of points an inversion exists between *each pair* of individuals and so $\tau = -1$. Which agrees with what we would expect.

Various methods exist for the computation of τ, which will be more or less appropriate according to the nature of the data set under consideration. The method which we applied to calculate τ, was based around the following procedure:

- We list the ordered data according to the first and second ranking
- Then for each member of the data set, we draw a straight lines connecting the same member in each of the rankings
- After drawing lines between each member we count the total number of times in which a pairs of lines cross
- The number of line crossings is the number of 'inversions in order'

In order to judge the significance of the results we need a **null hypothesis**. The null hypothesis in this case is taken to be the assumption that there is no association between the variables x and y. In this case Kendall's coefficient τ, is approximately normally distributed, with an expectation of zero and a variance of:

$$\text{Var}(\tau) = \frac{4N + 10}{9N(N-1)}, \tag{5.4.11}$$

where N is the total number of members of the set under consideration.

By calculating Kendall's Coefficient for many different pairs of rankings and plotting the results we are able to ascertain the degree we can have confidence in the association between the two variables.

5.5 Linear Regression

If a scatter plot and the value of a correlation coefficient indicate a linear association between two variables we use a regression line in order to predict the value of one variable from the value of the other.

In this case we have two kinds of variables:

- *independent variables* which we use to make predictions about the other variables
- *dependent variables* which we can be predicted to a certain degree of confidence from the other variables

5.5.1 The Least Squares Regression Line

The most common method for the construction of a regression line is the *method of least squares* which we will apply to the problem at hand. For a numerical data set of ordered pairs $(x_1, y_1), \ldots, (x_n, y_n)$, $n \in \mathbb{N}$. The least-squares regression line y on x is placed on a scatter diagram so that the sum of the squared vertical distances d_i, from the points to the line is minimized. Thus, the line is positioned so that $\sum_{i=1}^{n} d_i^2$, has the smallest value possible.

By a straight forward calculation we conclude that the regression line is given by $\hat{y} = ax+b$, where $a, b \in \mathbb{R}$, where:

$$b = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} {x_i}^2 - (\sum_{i=1}^{n} x_i)^2} \tag{5.5.12}$$

and

$$a = \frac{\sum_{i=1}^{n} y_i - b\sum_{i=1}^{n} x_i}{n} \tag{5.5.13}$$

5.5.2 The Least Square Regression Line for y on x

Clearly by transposing the axis the formulae for this regression line for y on x are identical to the formulae for x on y, except that the variables are interchanged. In this case the methodology is also the same except that the distances between the points and the regression can measured 'vertically' i.e. along the -axis' rather than horizontally along the axis'. Summarizing we get the regression line $\hat{x} = a + by$, where:

$$b = \frac{n\sum_{i=1}^{n} y_i x_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n\sum_{i=1}^{n} {y_i}^2 - (\sum_{i=1}^{n} y_i)^2} \tag{5.5.14}$$

$$a = \frac{\sum_{i=1}^{n} x_i - b\sum_{i=1}^{n} y_i}{n} \tag{5.5.15}$$

Remark This regression line is not used within the subsequent analysis. But by calculating both of the regression lines of x on y and y on x, and 'measuring' the difference between these lines we are able to show the rigidity of the method over ordering of the two variables.

5.5.3 Average of the Regression Error

We also record the average regression error. This is calculated by taking each data point and evaluated the arithmetic mean of the modulus of the minimal distance in the horizontal or vertical direction respectively between the data points and the regression line.

5.6 Prediction Intervals for the Conditional Mean

Regression lines are constructed from sample data and therefore subject to sample variation. Thus, two samples may result in two different regression lines:

$$\hat{y} = a_1 + b_1 x$$

$$\hat{y} = a_2 + b_2 x$$

Linear regression theory assumes that there is a *true* regression line:

$$\mu_{y|x} = \alpha + \beta x$$

We use a predicted value $\hat{y}$ to construct a confidence interval for the value of $\mu_{y|x}$, that is we find to what degree of accuracy and for what confidence level our regression line can predict elements within the data set. We refer to this value as the *prediction interval for the conditional mean*, $\mu_{y|x}$.

Remark In financial applications where for example we are considering the correlation between two assets closing days prices. It may seem in this instance that there is little sample variation since the closing price is set by the exchange. But say we had chosen to plot the regression line with the data set which consisted of the closing price on every other day (or we could use any other subset of the original data). The natural question is, would the regression line constructed from this data subset be significantly different? Prediction intervals address these issues and quantify to what degree the data set which we are considering has 'rigidity' in terms of the construction of regression lines. This information indicates to level of confidence which we can have with using the regression line to predict additional future values.

Within our construction we will use the Student's Distribution (also known as the t-distribution). The basic idea behind the t-distribution is that if the underlying data set only contains a small number of members, even if the underlying dynamics may to normal the normal distribution is not the most appropriate distribution to use because the tails tend to be too flat. The Student's distribution is a family of probability distributions which are parameterized by the number of degrees of freedom (related the size of he data set). The student distribution is used to develop interval estimates of a population mean whenever the population standard deviation is unknown and the population has a normal or near-normal probability distribution.

The values of $\hat{y}$ follow the Student's distribution with $(n-2)$ degrees of freedom where n

is the number of elements within the data set. The interval at a particular point on the regression line $(x_0, \hat{y_0})$ is constructed using:

$$\hat{y_0} \pm t s_{y|x} \sqrt{\frac{1}{n} + \frac{(x_0 - \frac{1}{n}\sum_{i=1}^{n} x_i)^2}{\sum_{i=1}^{n} {x_i}^2 - (\sum_{i=1}^{n} x_i)^2}} \tag{5.6.16}$$

where $s_{y|x}$ is the standard error of the estimate $\hat{y}$, given by:

$$s_{y|x} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-2}} \tag{5.6.17}$$

$$= \sqrt{\frac{\sum_{i=1}^{n} {y_i}^2 - a\sum_{i=1}^{n} y_i - b\sum_{i=1}^{n} x_i y_i}{n-2}}$$

5.6.1 The Coefficient of Determination

The coefficient of determination is a measure of the goodness of fit of the estimated regression line. It can be interpreted as the proportion of the variation in the dependent variable that is explained by the estimated regression line.

This coefficient is closely related to a number of other concepts which we will discuss. Say for the ith observation within our sample which we used to generate the regression line the observed value of the dependent variable y_i, and the estimated value of the dependent variable $\hat{y}_i$, which is implied from the regression line is called the ith *residual*. The ith residual represents the error in using $\hat{y}_i$ to estimate y_i. Thus, for the ith observation, the residual is $y_i - \hat{y}_i$. The sum of squares of these residuals or errors is the quantity that is minimized by the least squares method. This quantity, also known as the *sum of squares due to error*, is denoted by SSE. That is:

$$\text{SSE} = \sum_i (y_i - \hat{y}_i)^2 \tag{5.6.18}$$

The value of SSE is a measure of the error in using the estimated regression equation to estimate the values of the dependent variables in the sample.

Suppose that we wish to estimate values from a sample but do not know the size of the entire set. In this case we would use the sample mean $\bar{y}$, of the i sample members to estimate further values. The difference $y_i - \bar{y}$, provides a measure of the error involved in using $\bar{y}$ to estimate further values. The corresponding sum of the squares, called the total sum of squares, is denoted by SST and given by:

$$\text{SST} = \sum_i (y_i - \bar{y})^2 \tag{5.6.19}$$

To complete the circle, we can measure to amount to which the $\hat{y}$ values on the estimated regression line deviate from $\bar{y}$. This sum of squares is called the sum of squares due to regression, is denoted by SSR and given by:

$$\text{SSR} = \sum_i (\hat{y}_i - \bar{y})^2 \tag{5.6.20}$$

Naturally there is a nice relation between these three sums of squares which is one of the most important results within statistics. The relationship between SST, SSR and SSE is given by:

$$\text{SST} = \text{SSR} + \text{SSE} \tag{5.6.21}$$

where,

- SST = total sum of squares
- SSR = sum of squares due to regression
- SSE = sum of squares due to error

We will now introduce the coefficient of determination which uses the above measures in order to construct a measure of the goodness of fit for the estimate regression line. Using our above relation, the coefficient of determination denoted r^2, is given by:

$$r^2 = \frac{\text{SSR}}{\text{SST}} \left(= \frac{\text{SST}}{\text{SST}} - \frac{\text{SSE}}{\text{SST}} \right) \tag{5.6.22}$$

The estimate can be thought of as a 'normalized squares error', the normalizing is done by dividing through by SST. The coefficient r^2 satisfies $0 \leq r^2 \leq 1$ and further if the regression line is a perfect fit hence SSE = 0, then $r^2 = 1$, which intuitively is exactly what one would expect.

5.6.2 Residuals

We will also record the average of the residuals which does not form a useful statistical test since it is always zero by definition. What this does offer us is a means the test the computational reliability of the implementation which we use.

The residuals are the amount of variation of the Y values around the regression line. The populations errors, ϵ_i are the differences between the observed values and the regression line:

$$\epsilon_i = y_i - (\alpha + \beta x_i) \tag{5.6.23}$$

Remark In order to check the integrity of the numerical calculations we calculate all the residuals for each value and then take the average of all of these values. Theoretically this is zero and hence if we get a non-zero entry from the numerical calculations which we perform we know that the batch should be rerun. for the sake of completeness we give the formula below:

$$\sum_{i=1}^{n} \epsilon_i = \sum_{i=1}^{n} (y_i - (\alpha + \beta x_i)) = 0 \tag{5.6.24}$$

Chapter 6

Standard Probability Distributions Documentation

This chapter accompanies the Standard Probability Distributions module of the WebCab Probability and Statistics .NET Service. Within this chapter (and software module) we cover discrete and continuous probability distributions. In particular, for each probability distribution we will detail the variables which the distribution depends on, the domain of definition of the distribution, the probability density and associated cumulative density functions which the distribution is constructed from, the mean and variance of the distribution, and its skewness, kurtosis and mode.

For each of the probability distributions we provide a description of its parameters, the density function, evaluation of the probability, the mean and variance, the Skewness and Kurtosis, and the mode of the distribution. Where appropriate we will include further notes regarding issues such as the mean by which the inverse of the distribution in found.

6.1 Discrete Probability Distributions

6.1.1 Binomial Distribution

Parameters

This distribution considers n independent trials of independent events which have two possible outcomes (i.e. 'success' or 'failure'). The probability of whether a 'success' or 'failure', occurs is the same from one trial to the next. Therefore, we can describe what is referred to as a Binomial experiment with the following two parameters:

1. n: a positive integer which represents the number of independent trials considered within the Binomial experiment.

2. p: a real number within the closed interval $[0, 1]$, which represents the probability of a success occurring for each of the trials. Since, there are only two possible outcomes this implies that the probability of a 'failure' is $(p - 1)$.

Probability Density Function

If p is the probability of a 'success' in a single trial then the probability $p(x)$ of obtaining x 'successes' from n trials is given by:

$$p(x) = C_x^n \mathrm{p}^x (1-\mathrm{p})^{n-x} \tag{6.1.1}$$

where $C_x^n = \frac{n!}{x!(n-x)!}$.

Mean and Variance

The mean μ, and variance σ, of this distribution is given by:

$$\mu = n\mathrm{p} \tag{6.1.2}$$

$$\sigma^2 = n\mathrm{p}(1-\mathrm{p}) \tag{6.1.3}$$

Skewness and Kurtosis

The Skewness and Kurtosis are given by:

$$\text{Skewness} = \frac{1-2p}{\sqrt{np(1-p)}}$$

$$\text{Kurtosis} = 3 - \frac{3}{n} + \frac{1}{np(1-p)}$$

Mode

The Mode of this distribution is given by one of the following:

1. If $p(n+1)$ is an integer then the mode is:

   $$p(n+1) - 1 \text{ and } p(n+1)$$

   This case is referred to as the bimodal case since both of these two points have the greatest value, and hence are the mode.

2. Otherwise, we have a (uni)-modal case where the mode is given by:

   $$\text{largest integer less than } p(n+1)$$

Cumulative Distribution and Inverse Functions

The cumulative distribution function ($F(x, p, n)$) of the Binomial distribution in accordance with the notation introduced above, is given by:

$$F(x, p, n) = \sum_{i=0}^{x} p(x) = \sum_{i=0}^{x} C_x^n \mathrm{p}^x (1-\mathrm{p})^{n-x}$$

which leads us to the following definition of the inverse of the Binomial distribution ($F^{-1}([0,1] \to \mathbb{N})$:

$$F^{-1}(y) = \min\{x : y \leq F(x, p, n)\}$$

Random Number Generator

Within our component we offer methods by which random numbers are generated from the Binomial probability distribution. The procedure by which these random numbers are generated is as follows. For a random number a, within the range [0,1] (i.e. a probability), we return the **integer** value for which the cumulative Binomial distribution function is greater than or equal to the randomly generated number. More precisely, we solve the following expression:

$$F^{-1}(a) = \min\{x : a \leq F(x, p, n)\}$$

where $F(x)$ is the cumulative distribution function, (F^{-1} is the inverse (in the sense defined above) of the cumulative distribution function and x is randomly number generated from the Binomial distribution.

6.1.2 Poisson Distribution

The **Poisson distribution** is a discrete distribution used for modeling rare events. The rarity of the events is characterized by the fact that the probability density function has a factorial on its denominator, hence the density function is getting very small very quickly.

The Poisson distribution is associated with the Poisson experiment which has the following two properties:

1. The probability of an occurrence is the same for any two interval of the same length
2. Each occurrence an an interval is independent of any other occurrence of any other interval

That is, if the above two properties are satisfied then the number of occurrences is a random variable described by the Poisson probability function.

Parameters

The only parameter of this distribution is λ, and corresponds to the expected value or mean number of occurrences in an interval. This parameters λ, is required to be strictly positive (if $\lambda = 0$ then the distribution reduces to the trivial case).

Probability Density Function

The Poisson probability density function is given by:

$$p(x) = \frac{e^{-\lambda}\lambda^x}{x!} \tag{6.1.4}$$

where x is the number of occurrences in an interval and the parameter λ corresponds to the mean (or expected value) of the number of occurrences in an interval.

Mean and Variance

The mean and variance are both equal to λ, that is:

$$\text{Mean} = \text{Variance} = \lambda$$

Skewness and Kurtosis

The Skewness and Kurtosis is given by:

$$\text{Skewness} = \frac{1}{\sqrt{\lambda}}$$

$$\text{Kurtosis} = 3 + \frac{1}{\lambda}$$

Mode

The Mode is this distribution is given by one of the following:

1. If λ is an integer then the mode is:

$$\lambda \text{ and } \lambda + 1$$

   This case is referred to as the bimodal case since both of these two points have the greatest value and hence are the mode.

2. Otherwise, we have a (uni)-modal case where the mode is given by:

$$\text{largest integer less than } \lambda$$

Cumulative Distribution and Inverse Functions

The cumulative distribution function $F(x) : \mathbb{N} \to [0, 1]$, of the Poisson distribution is given by:

$$F(x) = \sum_{i=0}^{x} \frac{e^{-\lambda}\lambda^i}{i!}$$

From the cumulation distribution it follows that the **Inverse**, $F^{-1} : [0, 1] \to \mathbb{N}$, of the cumulative distribution function is given by:

$$F^{-1}(y) = \min\{x : y \leq F(x)\}$$

Random Number Generator

Within our component we offer methods by which random numbers are generated from the Poisson probability distribution. The procedure by which these random number are generated is as follows. For a random number a, within the range [0,1] (i.e. a probability), we return the **integer** value for which the cumulative Poisson distribution function is greater than or equal to the randomly generated number. More precisely, we solve the following expression:

$$F^{-1}(a) = \min\{x : a \leq F(x)\}$$

where $F(x)$ is the cumulative distribution function, (F^{-1} is the inverse (in the sense defined above) of the cumulative distribution function and x is randomly number generated from the Poisson distribution.

6.1.3 The Poisson approximation to the Binomial Distribution

Given a Binomial distribution for which the number of trials n is large and their probabilities p is small, the Poisson distribution with mean $\lambda = np$ provides an adequate approximation for most situations and is easier to calculate.

6.2 Continuous Probability Distributions

6.2.1 Normal Distribution

Parameters

The parameters of the Normal distribution are μ, which can be any real number and a scale parameter σ which is can be strictly positive real number. These parameters actually correspond to the mean and standard deviation of the distribution.

Density and Probability Function

The Normal distribution is a continuous probability distribution with probability density function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \equiv \frac{1}{\sqrt{(2\pi)}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{6.2.5}$$

where μ is the means of the distribution and σ is standard deviation.

Remarks

- A graph of this function is a bell-shaped curve which is symmetrical about the mean value, μ. The dispersion about the mean is determined by the value of the standard deviation, σ.
- We essentially rewrite the formula twice above to emphasize that on the denominator the fractional coefficient of the function of exponential type has a factor of σ. Sometimes this factor is written under the square root it which case it becomes σ^2.

Density and Probability Function

The **density function** of the distribution is given by:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

where μ is the mean and σ is the standard deviation of the deviation.

The **probability (or cumulative) function** of an observation lying between a and b is:

$$\int_a^b f(x)dx = \int_a^b \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx \tag{6.2.6}$$

where μ is the mean and σ is the variance of the given events.

Remark Since the above integral is not analytic we are forced to use a numerical procedure in order to evaluate its value we have applied the Extended Trapezoidal rule. We describe the details of the Extended Trapezoidal rule in last section of this chapter. The Inverse of the Normal Distribution Within this component we offer methods which allow for the inverse of this probability distribution to be evaluated. The particular inverse problems we have covered and there corresponding formulae and method names are as follows:

1. **Inverse** method solves for $b \in \mathbb{R}$, the following equation:

   $$P = \int_{-\infty}^b \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx$$

   where $P \in [0, 1]$, corresponds to the probability of the distribution.

2. **InverseFromValue** method solves for b there $b > a$, the following equation:

   $$P = \int_a^b \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx$$

   where $a \in \mathbb{R}$ corresponds to the lower bound of the interval considered and $P \in [0, 1]$, corresponds to the probability of the distribution.

3. **InverseAroundMean** method solves for $b \in \mathbb{R}$, the following equation:

   $$P = \int_{\mu-b}^{\mu+b} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx$$

   where μ is the mean of the distribution and $P \in [0, 1]$, is the probability of the distribution.

Mean and Variance

As mentioned before the mean is given by μ (one of the parameters), and the standard deviation by definition is σ^2, where σ is the variance of the distribution which is also one of the parameters of the distribution.

Skewness and Kurtosis

The Skewness and Kurtosis is given by:

$$\text{Skewness} = 0$$

$$\text{Kurtosis} = 3$$

Mode

The mode of the distribution is given by:

$$\text{Mode} = \mu$$

where μ as mentioned before is the mean of the distribution. Note that the mean is one of the parameters of the Normal distribution.

Random number generator

We provide a random number generator for the Normal distribution in the following sense. For a random number $a \in [0, 1]$, we return the value of the parameter for which the inverse Normal Distribution is equal to this randomly generated number. In particular, we find for which $b \in \mathbb{R}$, the following equality is satisfied.

$$a = \int_{-\infty}^{b} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx$$

Remark For convenience we also provide a related method in which generates a string of random numbers from the Normal probability distribution.

6.2.2 Standard Normal Distribution

The Normal distribution is difficult to work with because the probability integral given above is not analytic. Therefore in most applications a simplification known as the **Standard Normal distribution** is used. The Standard Normal distribution assumes that the Normal distribution satisfies:

- the mean, $\mu = 0$
- the standard deviation, $\sigma = 1$

To transform a Normal distribution with mean μ and standard deviation σ, into a Standard Normal Distribution we apply to each observation x the transformation:

$$z = \frac{x - \mu}{\sigma} \tag{6.2.7}$$

the resulting distribution will have mean equal to zero and standard deviation equal to one.

The above transformation is simply a change of scale:

$$P(X \leq A) = P\left(\frac{X-\mu}{\sigma} \leq \frac{A-\mu}{\sigma}\right) \tag{6.2.8}$$

All of the functionality available for the Normal distribution such as density, probability, inverse and random number generator; is also available for the Standard Normal Probability Distribution.

6.2.3 The Normal approximation to a Binomial Distribution

If we have a Binomial distribution for which the number of observations n is "large" and the probabilities p are not "too extreme" we can use the Normal distribution as an approximation to the Binomial distribution. We state this precisely by requiring:

- the mean of this distribution is $\mu = np$
- the standard deviation is $\sigma = \sqrt{npq}$

Remark Discrete observations are converted into continuous intervals.

6.2.4 Binomial Proportions

Using the Normal approximation to a **Binomial distribution**, the transformation above transformation becomes $z = \frac{x-np}{\sqrt{npq}}$. Denoting by $\hat{p}$ the proportion $\frac{x}{n}$, we obtain:

$$z = \frac{\hat{p}-p}{\sqrt{\frac{pq}{n}}} \tag{6.2.9}$$

Thus,

- $\mu_{\hat{p}} = p$
- $\sigma_{\hat{p}} = \sqrt{\frac{pq}{n}}$

6.2.5 Lognormal Distribution

Parameters

A random variable X has a Lognormal distribution with parameters μ and σ, if $Y = \ln(X)$ has the Normal distribution with mean μ and standard deviation σ.

Density Function

The probability density function for this distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2 x}} e^{-\frac{1}{2}(\frac{\ln x - \mu}{\sigma})^2} \quad , x > 0 \tag{6.2.10}$$

With corresponding probability (or cumulative) function:

$$\int_0^x f(x) = \int_0^x \frac{1}{\sqrt{2\pi\sigma^2 x}} e^{-\frac{1}{2}(\frac{\ln x - \mu}{\sigma})^2} dx$$

Mean and Variance

The Lognormal distribution has mean:

$$m = e^{\mu + \frac{1}{2}\sigma^2} \tag{6.2.11}$$

and variance:

$$V = e^{(2\mu + \sigma^2)}(e^{\sigma^2} - 1) \tag{6.2.12}$$

Equivalently:

$$V = m^2\alpha^2 \tag{6.2.13}$$

where α is the coefficient of variation.

Remarks

- The Lognormal distribution is useful for modeling investment returns and the distribution of insurance claim sizes.
- When it is important to guard against underestimating the probability of large claims, the Pareto distribution given below may be preferred.

Inverse of the Lognormal Distribution

Within this component we offer methods which allow for the inverse of this probability distribution to be evaluated. In particular the inverse problems we have solved and there corresponding formulae and method names are as follows:

1. **Inverse** method solves in $x > 0$; the following equation:

   $$P = \int_0^x \frac{1}{\sqrt{2\pi\sigma^2 x}} e^{-\frac{1}{2}(\frac{\ln x - \mu}{\sigma})^2} dx$$

   where $P \in [0, 1]$, is the probability.

2. **InverseFromRange** method solves for $x > 0$; the following equation:

   $$P = \int_{\text{lowerBound}}^x \frac{1}{\sqrt{2\pi\sigma^2 x}} e^{-\frac{1}{2}(\frac{\ln x - \mu}{\sigma})^2} dx$$

   where $P \in [0, 1]$, is the probability and lowerBound is the given lower bound on the interval considered.

Random number generator

We provide a random number generator for the Lognormal distribution in the following sense. For a random number $a \in [0, 1]$, we return the value of the parameter for which the inverse Lognormal Distribution is equal to this randomly selected value. In particular, we find for which $x \in \mathbb{R}$, the following equality is satisfied.

$$a = \int_0^x \frac{1}{\sqrt{2\pi\sigma^2 x}} e^{-\frac{1}{2}(\frac{\ln x - \mu}{\sigma})^2} dx$$

Remark For convenience we also provide a related method in which generates a string of random numbers from the Lognormal probability distribution.

6.2.6 Pareto Distribution

The **Pareto distribution** is useful for modeling the distribution of insurance claims when we wish to be cautious in assessing the probability of large claims.

Parameters

The **Pareto distribution** depends on two continuous real parameters, $\alpha \in \mathbb{R}$, and $\beta \in \mathbb{R}/\{0\}$.

Density and Cumulative Distribution Functions

The Pareto distribution is a continuous distribution with probability density function:

$$f(x) = \frac{\alpha}{\beta}\left(\frac{\beta}{x}\right)^{\alpha+1} \quad , x > \beta \tag{6.2.14}$$

where $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}/\{0\}$, and cumulative distribution function for $x > \beta$:

$$F(x) = 1 - \left(\frac{\beta}{x}\right)^{\alpha} \tag{6.2.15}$$

The Inverse of the Pareto Distribution

Within this component we offer methods which allow for the inverse of this probability distribution to be evaluated. The particular the inverse problems we have covered and there corresponding formulae and method names are as follows:

1. **Inverse** method solves for $x \in \mathbb{R}$, the following equation:

$$P = 1 - \left(\frac{\beta}{x}\right)^{\alpha}$$

where P is the given level of probability, i.e. a number within the closed interval $[0, 1]$.

2. **InverseFromValue** method solves for $x > A > \beta$, the following equation:

$$P = 1 - \left(\frac{\beta}{x}\right)^{\alpha}$$

where P is the given level of probability (i.e. $\in [0, 1]$) and A is the given lower bound.

Mean and Variance

The mean m of this distribution for $\alpha > 1$, is:

$$m = \frac{\alpha\beta}{\alpha - 1} \tag{6.2.16}$$

and the variance V, for $\alpha > 2$ is given by:

$$V = \frac{\alpha\beta^2}{(\alpha - 1)^2(\alpha - 2)} \tag{6.2.17}$$

Random number generator

We provide a random number generator for the Pareto distribution in the following sense. For a random number $a \in [0, 1]$, we return the value of the parameter for which the inverse Pareto Distribution is equal to this randomly selected value. In particular, we find for which $x \in \mathbb{R}$, the following equality is satisfied.

$$a = 1 - \left(\frac{\beta}{x}\right)^{\alpha}$$

Remark For convenience we also provide a related method in which generates a string of random numbers from the Lognormal probability distribution.

6.2.7 Uniform Probability Distribution

The **Uniform Probability Distribution** with its associated uniform density function is a continuous probability distribution. A random variable is uniformly distribution is uniformly distributed if the probability is proportional to the length of the interval. Where the probability that the random variable will assume a value in any two intervals is identical if the intervals are of the same length.

More precisely, the **uniform probability density function** $f(x)$ for a random variable x, is given by:

$$f(x) = \frac{1}{b - a}, \qquad \text{for } a \leq x \leq b$$
$$f(x) = 0, \qquad \text{otherwise}$$

The probability of a random event lying in the interval $[c.d]$, where $a \leq c \leq d \leq b$, is:

$$\text{Probability of event} = \frac{d - c}{b - a}$$

Mean and Variance

The mean (or expected value) $E(x)$, of the uniform probability distribution is:

$$E(x) = \frac{a+b}{2}$$

The variance $Var(x)$, is given by:

$$Var(x) = \frac{(b-a)^2}{12}$$

where as before $a \leq b$.

Random number generator

Since all 'events' have the same probability the random number generated from the uniform distribution is just a random element selected from the finite interval in which it is defined, namely $[a, b]$.

6.2.8 Exponential Probability Distribution

The continuous probability distribution known as the **exponential probability distribution** is useful when wishing to describe the time it takes for a given task to be completed[1]. The **density function** $f(x)$, is given by:

$$f(x) = \frac{1}{\mu} \exp\left(-\frac{x}{\mu}\right), \qquad \text{for } x \geq 0,\ \mu > 0$$

where μ in the mean of the distribution. It then follows that the probability function $P(x \leq x_0)$, is given by:

$$P(x \leq x_0) = 1 - \exp\left(-\frac{x_0}{\mu}\right)$$

which gives the probability of obtaining a values from the exponential distribution less than or zero to some given value $x_0 \in \mathbb{R}^+$.

6.3 Supplementary Material on Extended Trapezoidal rule

In order to evaluate the probability with respect to the Normal Distribution of an observation lying within the interval $[a, b]$, we have applied the Extended Trapezoidal rule. Within this section we details the implementation of the Extended Trapezoidal rule.

[1]The Exponential distribution is closely related to the Poisson distribution by the following fact: If the time of events follows a Poisson distribution, then the time between events follows an exponential distribution.

We approximate the integral of a function $f(x)$ defined on the interval $[a, b]$, by dividing the region on which the function is integrated over into trapeziums.

The interval $[a, b]$, is divided into n intervals, using $n-1$ equidistant points $x_1 < \ldots < x_{n-1}$. We approximate the integral of $f(x)$ on the interval $[a, b]$ by:

$$\frac{b-a}{2n}\left(f(a)+f(b)+2\sum_{k=1}^{n-1}f(x_k)\right) \tag{6.3.18}$$

For further details concerning the Extended Trapezoidal rule please see:

WebCab Differentiation and Integration Documentation

Chapter 7

Hypothesis Testing Documentation

Within this module we consider two aspects of inferential statistics known as confidence intervals and hypothesis testing. Inferential statistics addresses the following question, "can we say something about the population, given only the sample evidence?". There are a number of point statistics (i.e. mean, variance etc) which can be applied to a sample set which attempt to capture a quality of a entire population set. For a given confidence level the amount which this statistic can vary for different data set samples is referred to as the confidence interval. Note, that within the estimate of the confidence interval we assume that the underlying data set is of normal or close to normally distributed.

Often a statistician is not solely interested in estimation but in deciding if some hypothesis seems reasonable in light of the sample evidence. In many instances the problem is to judge which of several possible hypothesis is best supported by the sample data. Methods are known as hypothesis testing and we develop this theory set data sets which are normal or close to normally distributed.

7.1 Confidence intervals

An interval estimate, or a confidence interval, is an interval of values with a given probability of covering the true population parameter. This parameter could be a mean, a variance, a proportion or similar point wise statistic.

It is clear that the sample mean, or proportion will seldom exactly equal the corresponding population mean, or proportion.

The point estimate alone does not give any idea of the magnitude of the possible sampling error. A procedure that provides a convenient way of indicating the general magnitude of the sampling error in any given situation is interval estimation.

Where there is a large degree of sampling error, the confidence interval estimated from any sample will be large; the range of values likely to cover the population parameter is wide. On the other hand, if the sampling error is small the parameter is likely to be covered

by a small estimated range of values.

A confidence interval takes the following form:

$$\bar{x} \pm z\sigma$$

where $\hat{x}$ is the sample parameter, z is known as the critical value and depends on the confidence level (p) and σ will be the samples standard error. The probability of the population parameter lying in the interval $\hat{x} \pm z\sigma$ is p.

This is the type of confidence interval we will use. It claims that the true population parameter (which we do not know) will lie within z standard errors of the sample statistic with a $p\%$ level of confidence.

The way the z-value is calculated from the confidence level depends on the way the entire population is distributed (i.e. how close to a normal distribution) or on the type of interval we want to compute (two-sided or one-sided). If the sample size is big enough (> 30) the normal distribution should be used, but if it is small (< 30) we will use the t-distribution (also known as the student distribution).

According to the population parameter we are dealing with, the interval type and the sample size we can compute different types of confidence intervals.

7.1.1 Confidence intervals for large or small samples

When the population standard deviation is known or the sample size is large enough (> 30) so that the sample's standard deviation would be a good estimate for the population value we can use the normal distribution to calculate the critical values.

When the sample size is small (< 30) we can not use the normal distribution anymore because the standard deviation of the sample (s) does not provide a good estimate for the population standard deviation (σ). In this case we will use the t-distribution to calculate the critical values.

7.1.2 One or two sided confidence intervals

To find $a \in \mathbb{R}$ such that the probability of the statistic x is between $-a$ and a is $p\%$, (which can be written as: $P(-a \leq x \leq a) = p\%$) you must calculate the two sided confidence interval for the statistic x at $p\%$ confidence level. In this case z will be the $1-(1-p/100)/2$ percentile of the distribution.

So if you want to find a confidence interval for the statistic at a certain confidence level looking at both ends of its limits you will compute a two-tailed confidence interval. The confidence interval will be of the form $(-z\sigma \leq \mu \leq z\sigma)$, where σ is the standard error.

To find $a \in \mathbb{R}$ such that the probability of the statistic x is greater than a or smaller than a is greater than $p\%$ (i.e. $P(x > a) = p\%$, or $P(x < a) = p\%$), we must calculate the one-sided confidence interval for x at $p\%$ confidence level. In this case z will be the right-tailed $(1 - p/100)$ percentile of the distribution in the first case and the left-tailed $(1 - p/100)$ percentile of the distribution in the second case.

If you want to find a confidence interval for the mean at a certain confidence level focusing on only one end of its limits, you will compute a one-sided confidence interval. The interval will be of the form $(z\sigma < \mu)$ or $(z\sigma > \mu)$, where σ is the standard error.

Remark The distribution used will be either the standard normal distribution (if we are in the large sample case) or the student distribution with $(n - 1)$ (single samples) or $(n_1 + n_2 - 2)$ (two samples) degrees of freedom where n, n_1 and n_2 are the sizes of the small samples being used.

7.1.3 Confidence intervals with one or two samples

Sometimes we will have the results from two samples and we will want to compute a confidence interval for the difference between the samples parameters.

Let assume our sample parameters are $\bar{x_1}$ and $\bar{x_2}$ (means or proportions) and we want to calculate a confidence interval for $\bar{x_1} - \bar{x_2}$. The two populations must be normally distributed so that in the large sample case the sampling distribution of $\bar{x_1} - \bar{x_2}$ can be approximated by a normal probability distribution, and in the small sample case by a t-distribution.

The critical value z is calculated in the same way as with single samples but the way we calculate the standard error differs.

7.1.4 Confidence intervals for the mean and percentages

The most common types of population parameters used are means and percentages.

- **A mean** is calculated by dividing the sum (or integral) of the values by the number (or range) in the sample. When we calculate confidence intervals for the mean we need to know the standard error of the sample. For single samples the standard error is given by:

  $$\frac{s}{\sqrt{n}}$$

  where s is the standard deviation and n is the sample size. For two samples the standard error is given by:

  $$\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

  where s_1 and s_2 are the two standard deviations and n_1, n_2 are the two sample sizes.

- **A percentage** is the proportion of data with a certain characteristic in the whole population. The standard error for proportions is given by:

$$\sqrt{\frac{p(100-p)}{n}}$$

for single samples where p is the proportion, n is the sample size. For two samples the standard error is given by:

$$\sqrt{\frac{p_1(100-p_1)}{n_1}+\frac{p_2(100-p_2)}{n_2}}$$

where p_1, p_2 are the proportions and n_1, n_2 are the samples sizes.

7.2 Hypothesis (or significance) testing

Hypothesis (or significance) testing makes statements about a population parameter, or parameters, on the basis of sample evidence. This role also applies to confidence intervals. In the same way that we created a confidence interval for the differences between two samples, we can also test for such differences.

Hypothesis testing is concerned with accepting or rejecting ideas these ideas are known as *hypothesis* . If wish to test one in particular, we refer to it as the *null hypothesis* .

As a procedure, we first state a null hypothesis; something we wish to judge as true or false on the basis of statistical evidence. Then we check whether or not the null hypothesis was consistent with the confidence interval.

Remark A confidence interval can be regarded as a set of acceptable hypothesis.

When we conduct a significance test we have two hypotheses; one related to the supposition that we are testing and one which describes the alternative situation.

The first hypothesis relates to the claim, supposition or previous situation and is called the *null hypothesis* and is labeled H_0. The null hypothesis could be written out as a sentence but it is more usual to abbreviate it to:

$$H_0 : \mu = \mu_0 \tag{7.2.1}$$

for a mean where μ_0 is the claimed or previous population mean.

For a percentage we have:

$$H_0 : \pi = \pi_0 \tag{7.2.2}$$

where π_0 is the claimed or previous population percentage.

The second hypothesis summarizes what will be the case if the null hypothesis is not true. It is called the *alternative hypothesis* and is labeled H_A. The alternative hypothesis could be written out as a sentence, but a shorter notation is usually preferred.

For a mean we write:

$$H_A : \mu \neq \mu_0 \tag{7.2.3}$$

For a percentage we write:

$$H_A : \pi \neq \pi_0 \tag{7.2.4}$$

For each hypothesis test that we wish to conduct, the basic procedure will remain the same and can be summarized in the following steps:

1. State hypotheses
2. State significance level
3. State critical (cut-off) values
4. Calculate the test statistic z
5. Compare the z value to the critical values
6. Make a conclusion

7.2.1 Hypothesis tests for means or percentages

A test statistic for the population mean

In the case of testing for a population mean we calculate the test statistic z (the fourth step above) with the formula:

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}} \tag{7.2.5}$$

If we use the sampled standard deviation s instead of the population standard deviation σ and we assume that the null hypothesis is true than the formula for the z-statistic is given by:

$$z = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \tag{7.2.6}$$

A test statistic for a population percentage

We calculate the z-value with the formula:

$$z = \frac{p - \pi_0}{\sqrt{\left[\frac{\pi_0(100-\pi_0)}{n}\right]}} \tag{7.2.7}$$

where p is the sample percentage and π_0 is the claimed population percentage.

The formula for the sampling error is:

$$\sqrt{\frac{\pi_0(100 - \pi_0)}{n}} \tag{7.2.8}$$

Remark The standard error for a percentage is different from the standard error for the mean.

One-sided significance tests

In this section we specify whether the real value is above or below the claimed value in the case where we are able to reject the null hypothesis.

We test whether or not the percentage of a sample has decreased. The hypotheses are:

$$H_0 : \pi = \pi_0$$

$$H_A : \pi < \pi_0$$

If we want to test if the percentage has increased, then the hypotheses are:

$$H_0 : \pi = \pi_0$$

$$H_A : \pi > \pi_0$$

For the mean the procedure is the same.

Remark One-sided test are sometimes referred to as testing *producers' risks* or *consumers' risks* .

7.2.2 Hypothesis testing with one or two samples

Previously we worked only with single samples. In many cases we may know little about the actual population value, but will have available the results of another survey. This could be a situation where we have two surveys conducted in different parts of the country, or at different points in time.

Tests may be carry out at the 5% or 1% level, and the interpretation of the results will depend upon the calculated value of the test statistic and its comparison to a critical value. We will rewrite the hypotheses to take account of the two sample and to find a new formulation for the test statistic.

Test for a difference of means

We will refer to the samples using the suffixes 1 and 2 (i.e. sample 1 and sample 2). Our assumption is that the two samples come from the same population. This assumption gives us the null and the alternative hypotheses. The null hypothesis is:

$$H_0 : \mu_1 = \mu_2$$

The alternative hypothesis will depend upon whether we are conducting a two-sided test or a one-sided test. For a two-sided test the alternative hypothesis is:

$$H_A : \mu_1 \neq \mu_2$$

and for a one-sided test, it is:

$$H_A : \mu_1 < \mu_2$$

or

$$H_A : \mu_1 > \mu_2$$

The test statistic we use is closely related to a confidence interval test for the difference between the two means.

Tests for a difference in population percentage

By modifying the hypotheses and using to corresponding test statistic we are able to perform hypothesis tests for a difference between population percentages.

7.2.3 Hypothesis testing with large or small samples (t-distribution)

Until now we performed hypothesis tests for large samples with more than 30 elements. In a similar fashion to confidence intervals we can also perform hypothesis tests for small samples.

Single samples

When the samples are small we need to use the t-distribution. The basic assumption that the sampling distribution for the sample parameters is a Normal distribution only holds when the samples are large.

For a single sample, the test statistic for a population mean is calculated by using:

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}} \tag{7.2.9}$$

with $(n-1)$ degrees of freedom.

For a single sample, the test statistic for a population percentage is calculated using:

$$t = \frac{p - \pi_0}{\sqrt{\frac{\pi_0(100-\pi_0)}{n}}} \tag{7.2.10}$$

with $(n-1)$ degrees of freedom.

Two samples

In the case of estimating a confidence interval for the difference between two means the pooled standard error (assuming that both samples have similar variability), is given by:

$$s_p = \sqrt{\frac{(n_1-1){s_1}^2 + (n_2-1){s_2}^2}{n_1+n_2-2}} \tag{7.2.11}$$

We perform the test statistic:

$$t = \frac{(x_1 - x_2) - (\mu_1 - \mu_2)}{s_p\sqrt{\frac{1}{n_1} - \frac{1}{n_2}}} \tag{7.2.12}$$

with $(n_1 + n_2 - 2)$ degrees of freedom.

Remark A special case arises when we are considering tests of the difference of means if the two samples are related in such a way that we may pair the observations. This may arise when two different people assess the same series of situations, for example, different interviewers assessing the same group of candidates for a job.

7.3 Types of error

Samples can only give us a partial view of a population; there will always be some chance that the true population value really does lie outside the confidence interval, or that we will come to the wrong decision when conducting a significance test. In fact these probabilities are specified in the names that we have already used: a 95% confidence interval and a test of 5% level of significance. Both imply a 5% chance of being wrong.

There are two different types of error:

- **Type 1 error** is the rejection of a null hypothesis when it is true. This probability is known as the **significance level** of the test. This is usually set at either 5% or 1% for most business applications, and is decided upon before the test is conducted.
- **Type 2 error** is the failure to reject the null hypothesis, which is false. The probability of this error cannot be determined before the test is conducted.

7.4 Conclusions

Confidence intervals and hypothesis testing are a very precise way of statistical inference. But we must not forget that for these methods to work the population we study must be normal or near to a normal distribution. The standard normal distribution is a continuous function but when we are dealing with finite samples we are handling discrete functions. For this reason in order to attain the most reliable conclusions we should use a sample size which is as large as practically possible.

However, in some situations the samples are small and in these cases the student t-distribution is used. The student distribution cases should be treated with much more care because of the higher likely hood of misleading results. In some instances we have omitted a methods for the small sample case because the chance of obtaining misleading results is to high.

Within this component we have methods that work in a wide range of situations for both large and small sample sets. We provide methods for the most widely used types of confidence intervals and methods of hypothesis testing.

Chapter 8

Programmer's Guide for Microsoft® Office

This chapter describes the steps to take in order to integrate WebCab Probability and Statistics 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.

8.1 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

8.1.1 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 8.1.

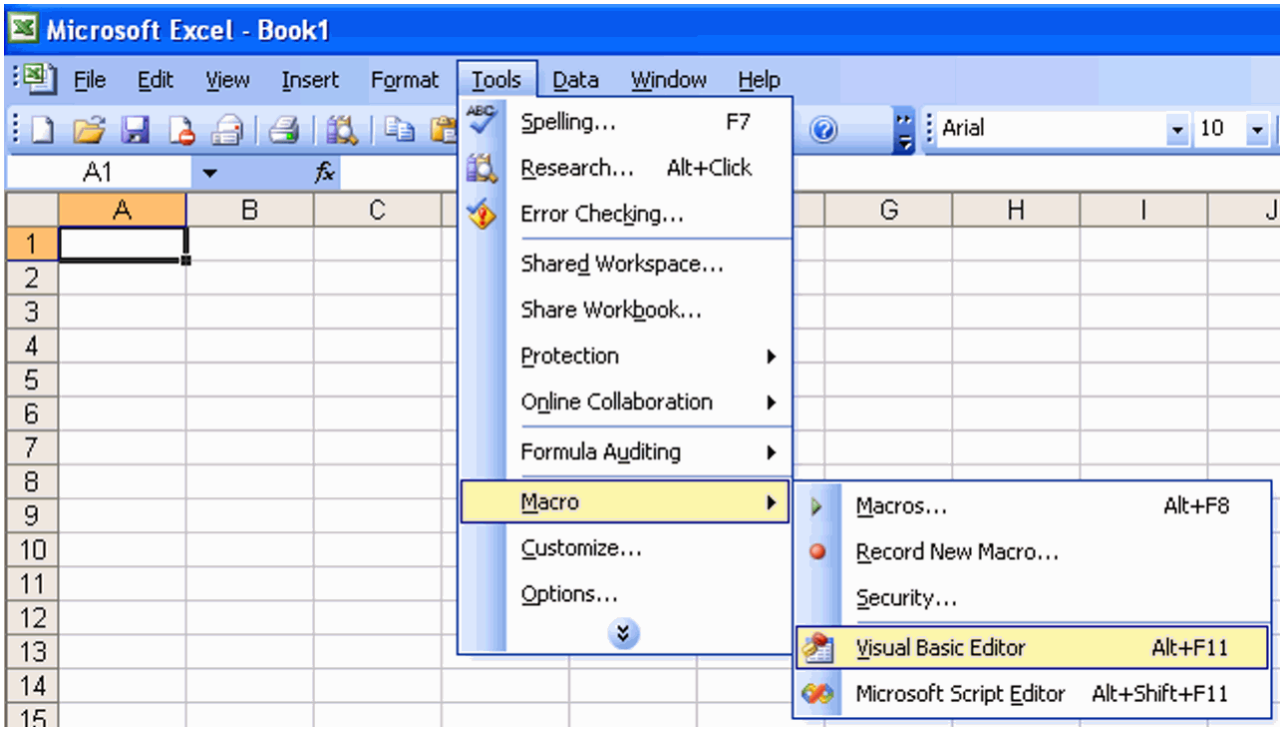


Fig. 8.1: Starting the Visual Basic Editor in Excel

8.1.2 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. 8.2 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.

8.1.3 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. 8.3.

8.1.4 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.

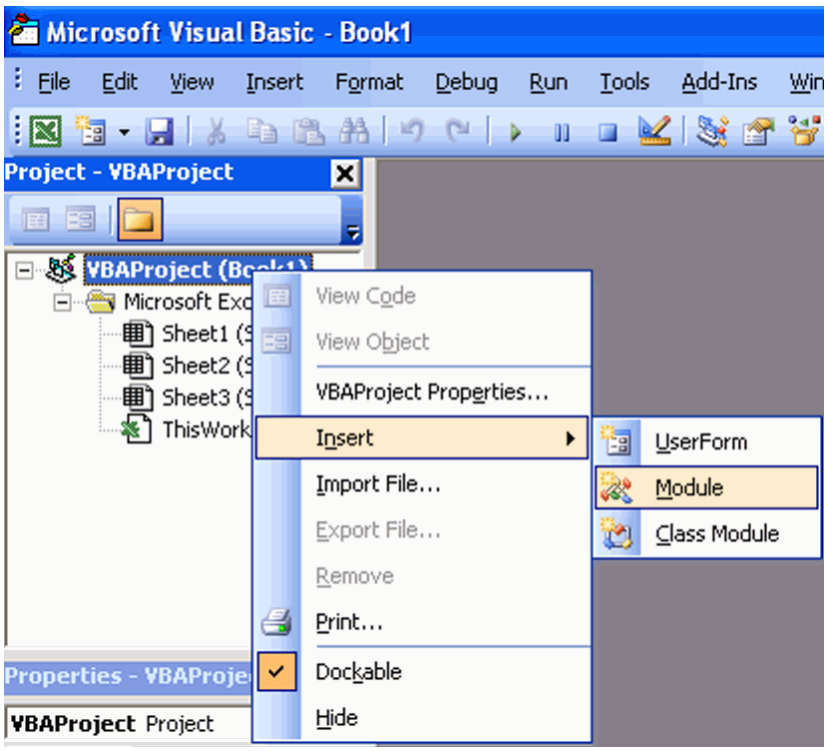


Fig. 8.2: Adding a module to a VBAProject

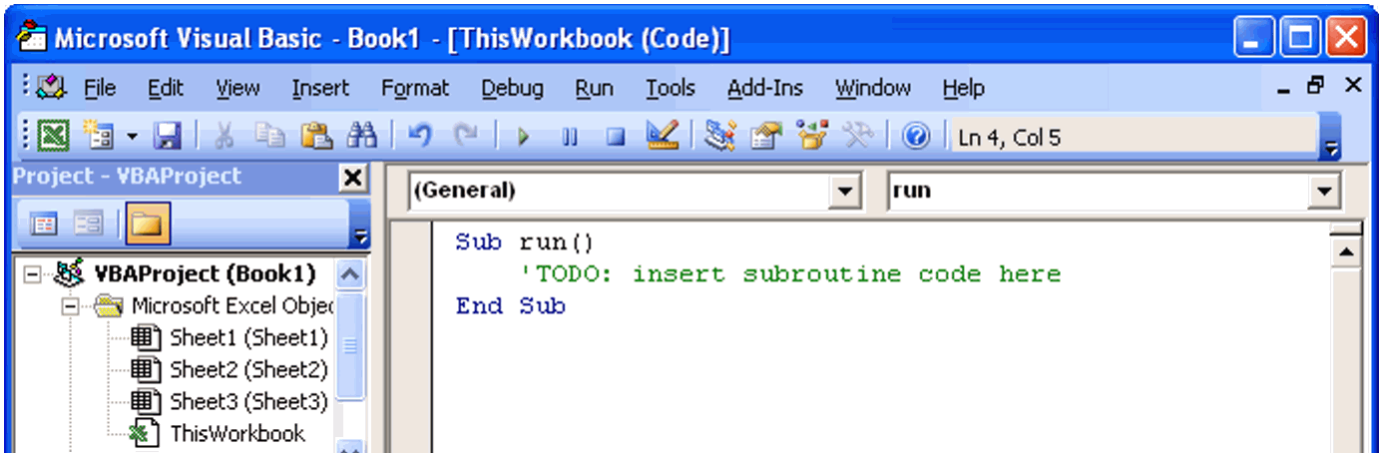


Fig. 8.3: Declaring a subroutine named *run* in VBA

To add a reference to this product, go to the Tools | References... menu as shown in Fig. 8.4 in order to open the "References" dialog window. Scroll down and select the entry named *WebCab Probability and Statistics Demo* from the list of available references in the dialog window and then click OK. Figure 8.5 shows how to add a reference to *WebCab Functions Demo*.

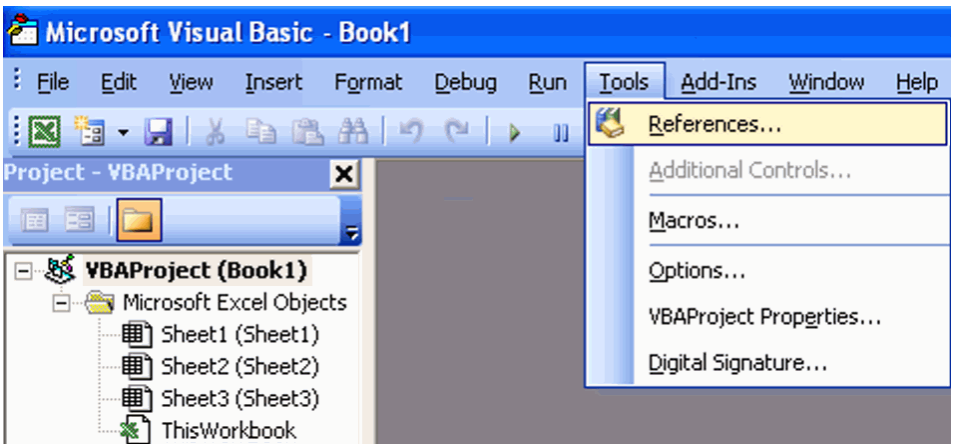


Fig. 8.4: Opening the "References" window in the Visual Basic Editor for Office

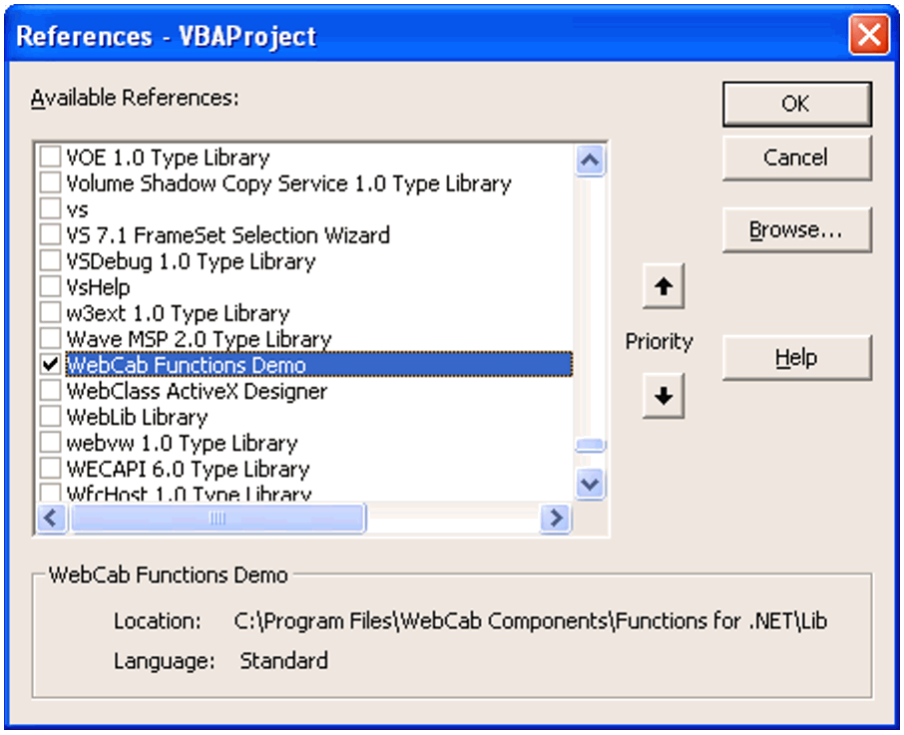


Fig. 8.5: Selecting a reference to a WebCab product, here 'Functions'

8.1.5 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 | Probability and Statistics 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.

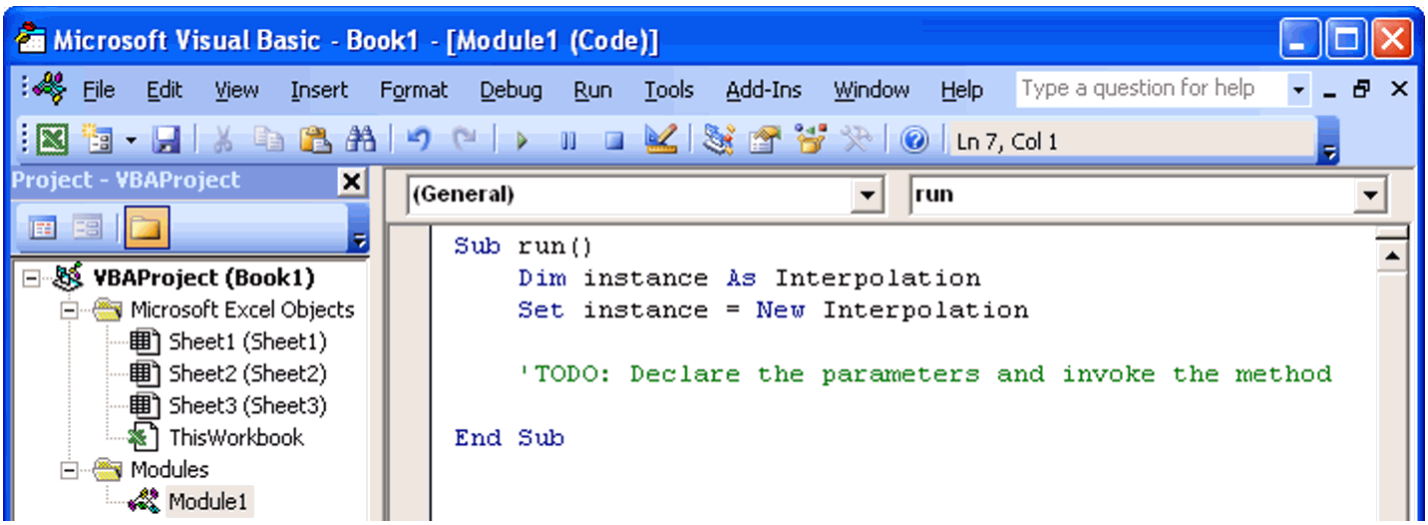


Fig. 8.6: Creating a new instance of a class named "Interpolation"

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
```

8.1.6 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.

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 8.6 shows how to create an instance of this class and how to assign it to the `instance` variable declared in the step above.

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

8.1.7 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 8.1.10.

8.1.8 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.

8.1.9 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.
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. 8.7 to open the "Macro" dialog window. Select the macro, which has the same name as your subroutine and click Run (see Fig. 8.8).

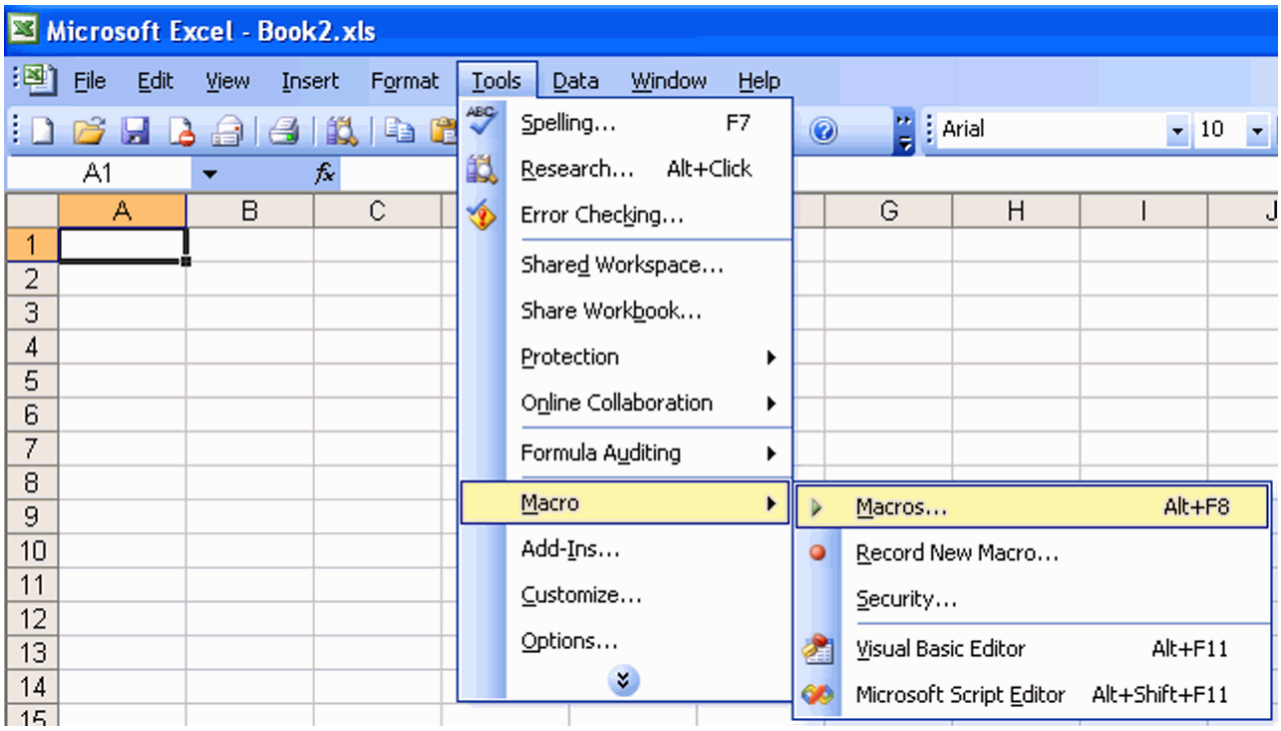


Fig. 8.7: Opening the "Macro" window from the menu in Excel.

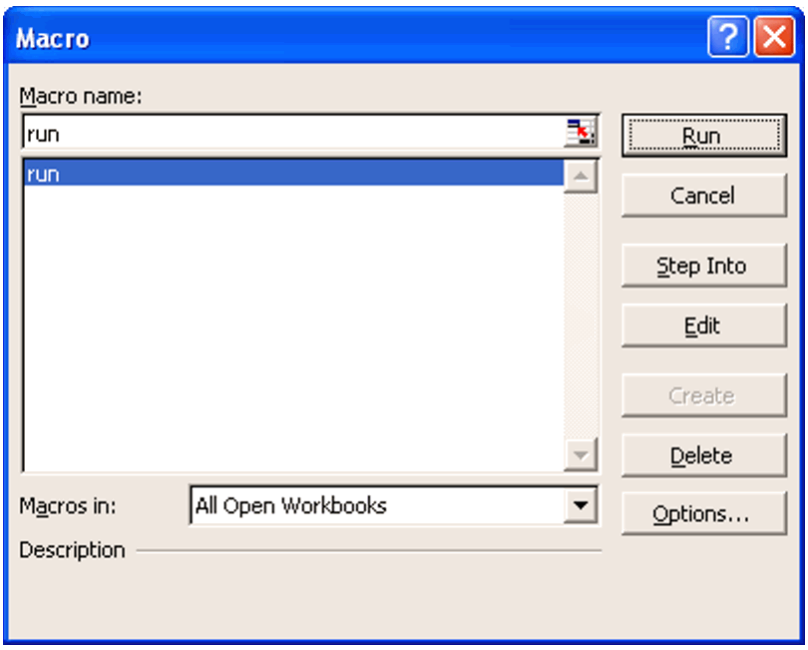


Fig. 8.8: Running a subroutine macro named *run* from Office.

```
' 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 8.1: Generic Office VBA Example

8.1.10 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 8.1 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.

8.2 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

8.2.1 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 8.1. The Visual Basic Editor will allow you to write the necessary VBA code that uses the functionality provide by this product.

8.2.2 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. 8.2 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.

8.2.3 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 8.9 shows how to declare a function named *MyFunction*.

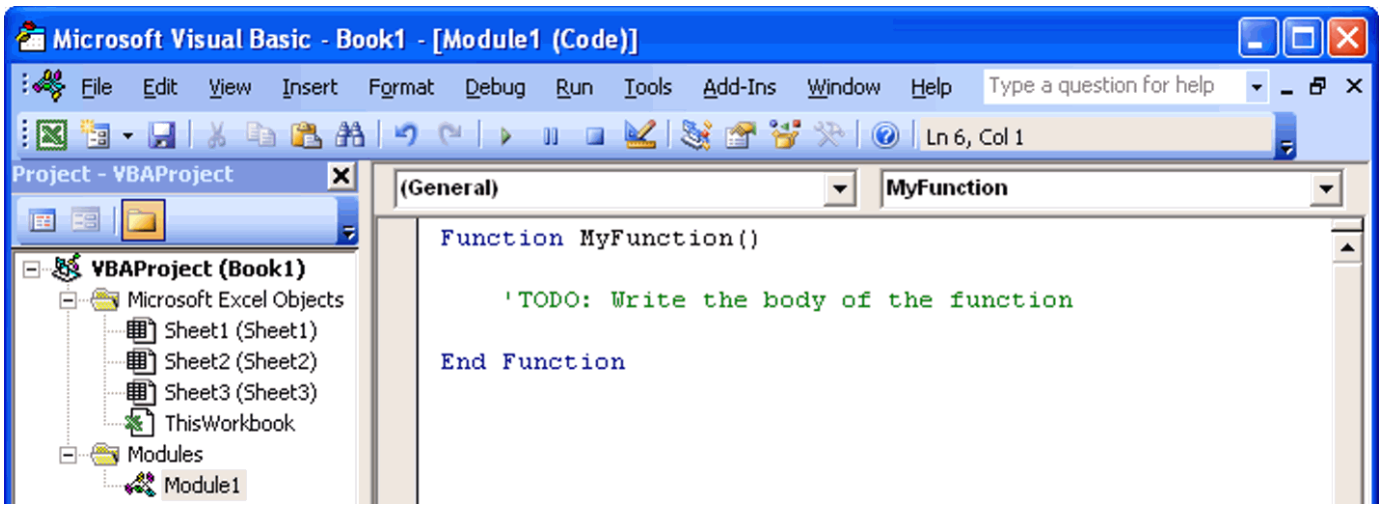


Fig. 8.9: Declaring a function named "MyFunction" in VBA

8.2.4 Add a Reference to This Product

As described in section 8.1.4, in order to add a reference to this product, you will need to go to the Tools | References... menu as shown in Fig. 8.4. This will open the "References" dialog window. Scroll down and select the entry named *WebCab Probability and Statistics Demo* from the list of available references in the dialog window and then click OK. Figure 8.5 shows how to add a reference to *WebCab Functions Demo*.

8.2.5 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 | Probability and Statistics 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
```

8.2.6 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
```

8.2.7 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, 8.2.8.

8.2.8 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)
```

The complete source code of a function named *MyFunction* is shown in table 8.2. The example calls the same method as the generic VBA example in section 8.1.10.

8.2.9 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. 8.10). From the dialog window, click the drop-down list that lists all the categories, and select the category named "User Defined", as shown in

```
' 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 8.2: The VBA code for a user-defined function in Excel

figure 8.11.

The name of your function will appear in the list of user-defined functions. Select it and click the OK button (Fig. 8.12). 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 8.13.

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 8.14. 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,

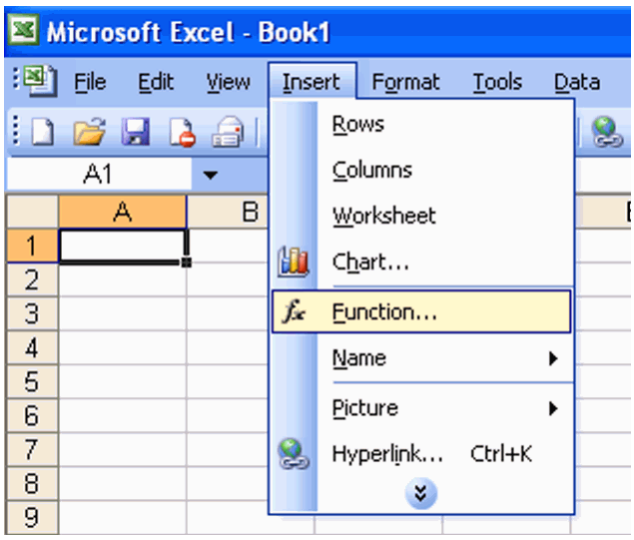


Fig. 8.10: How to open the "Insert Function" window in Excel

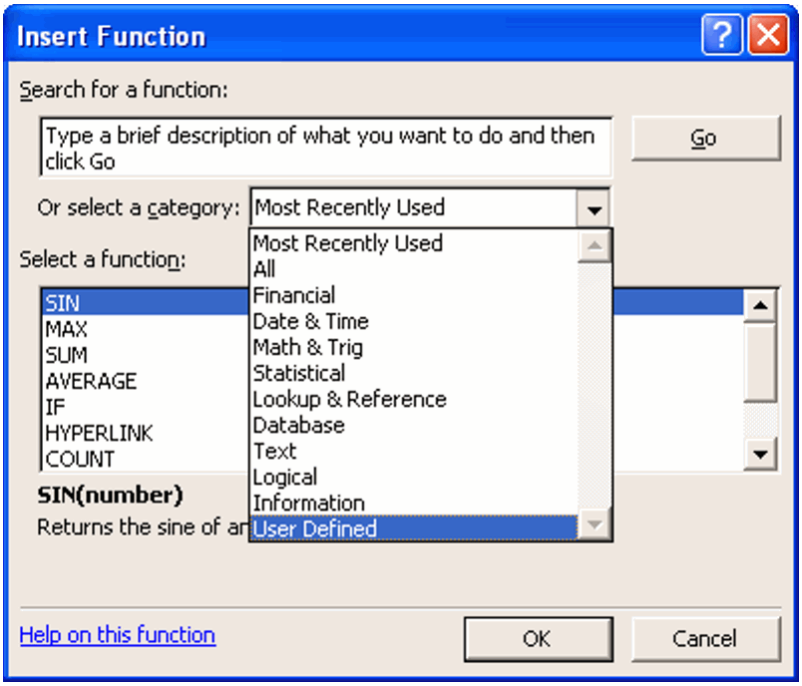


Fig. 8.11: Selecting the "User Defined" category

the same way you would call a standard Excel formula.

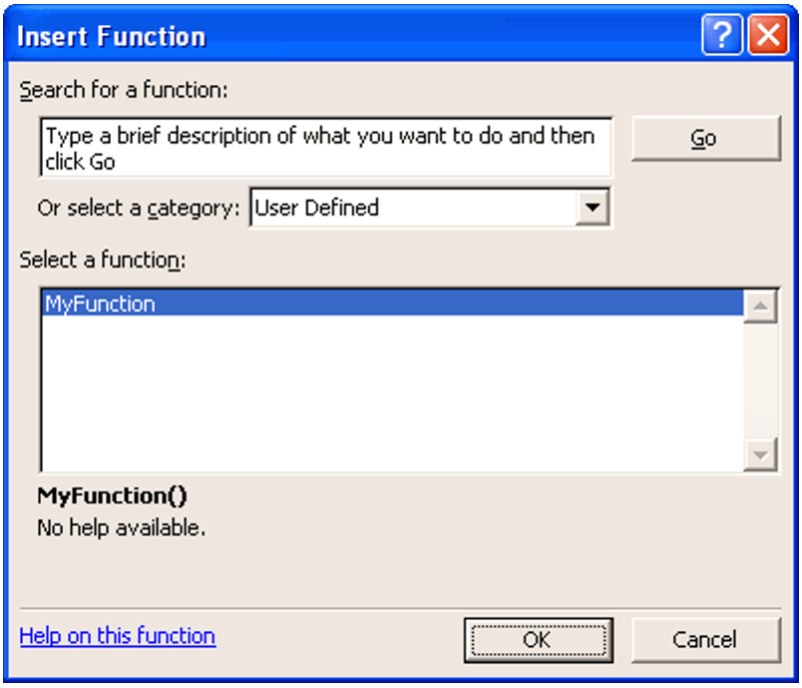


Fig. 8.12: Selecting the "MyFunction" user-defined function

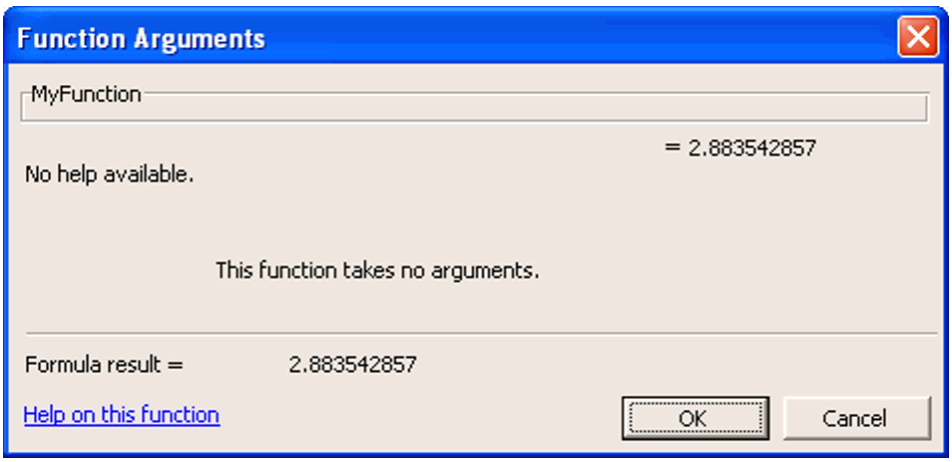


Fig. 8.13: Inserting the "MyFunction" user-defined function into the worksheet

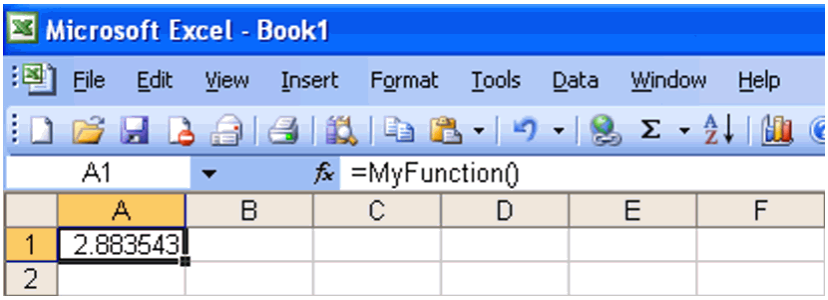


Fig. 8.14: The value returned by the "MyFunction" formula in an Excel worksheet

Chapter 9

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 Probability and Statistics 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.

9.1 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 Probability and Statistics 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

9.1.1 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.

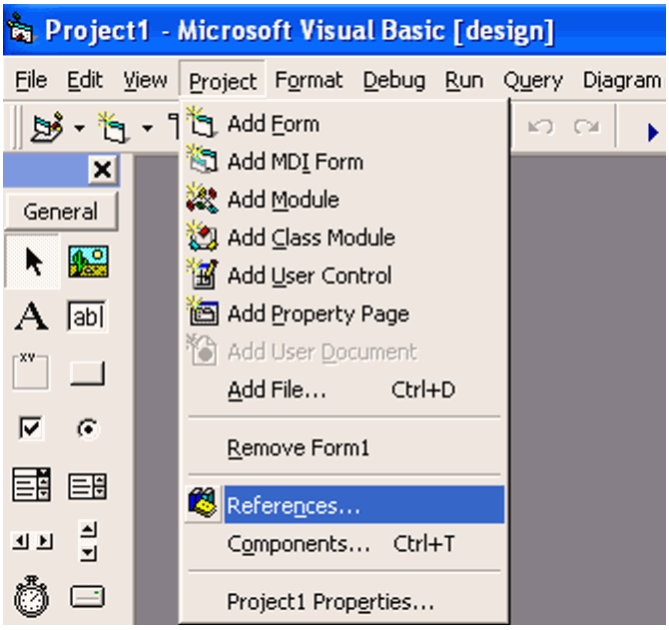


Fig. 9.1: Opening the "References" window in Visual Basic 6

To add a reference to this product, go to the Project | References... menu as shown in Fig. 9.1 in order to open the "References" dialog window. Scroll down and select the entry named *WebCab Probability and Statistics Demo* from the list of available references in the dialog window and then click OK. Figure 9.2 shows how to add a reference to *WebCab Functions Demo*.

9.1.2 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 | Probability and Statistics 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

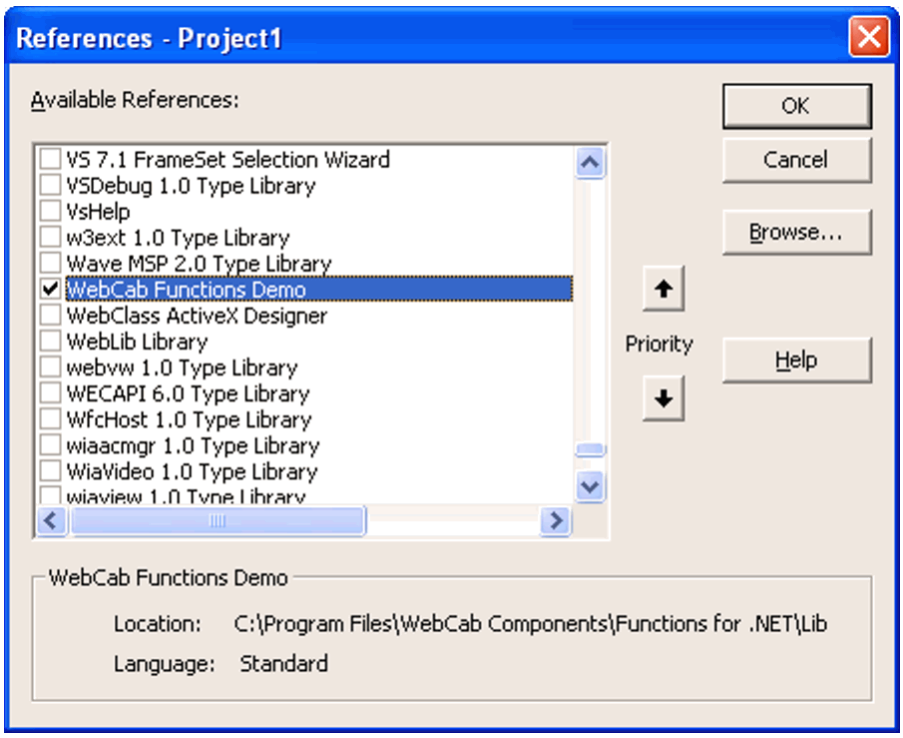


Fig. 9.2: Selecting a reference to a WebCab product, here *WebCab Functions*

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

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

9.1.3 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
```

9.1.4 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, 9.1.5.

9.1.5 A Generic Visual Basic Example

In this section we provide a generic VB6 client example, which invokes a method of a business class and prints the result to the screen. Although it might not directly apply to this product, this example uses a real WebCab product, a real business class, and a real method name, so that you can test it for yourself by downloading the corresponding Demo package.

The code below 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.

You can use the code in table 9.1 by copy-pasting it inside any of your subroutines or functions. In our example, we placed the code inside the `Form_Load` subroutine, which is being triggered at the time the Application Form is first started.

```
Private Sub Form_Load()
    Dim instance As Interpolation
    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 in a Message Box
    MsgBox "The value of the function at the point 1.2 is " & result
End Sub
```

Table 9.1: A generic Visual Basic client example

9.2 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

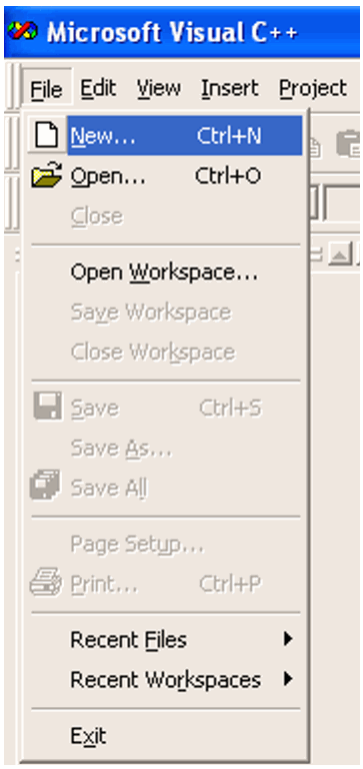


Fig. 9.3: Starting a new project

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`"

9.2.1 Open a New or Existing Project

You can start a new Visual C++ Project by going to the File | New... menu (see Fig. 9.3), 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. 9.4.

In the next window, select "A simple application" (see Fig. 9.5), 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. 9.6. Note that the source code snippets in this section apply both to a new project and to one of your existing projects.

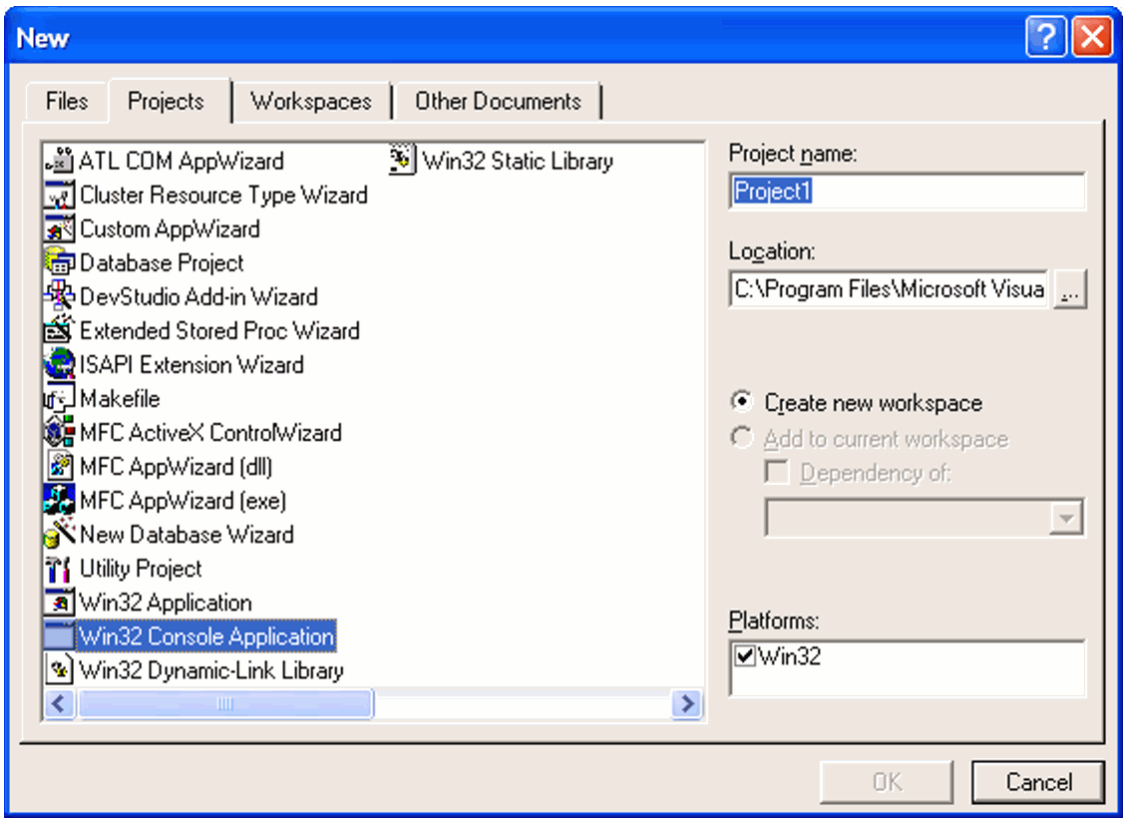


Fig. 9.4: Starting a Console Application named "Project1"

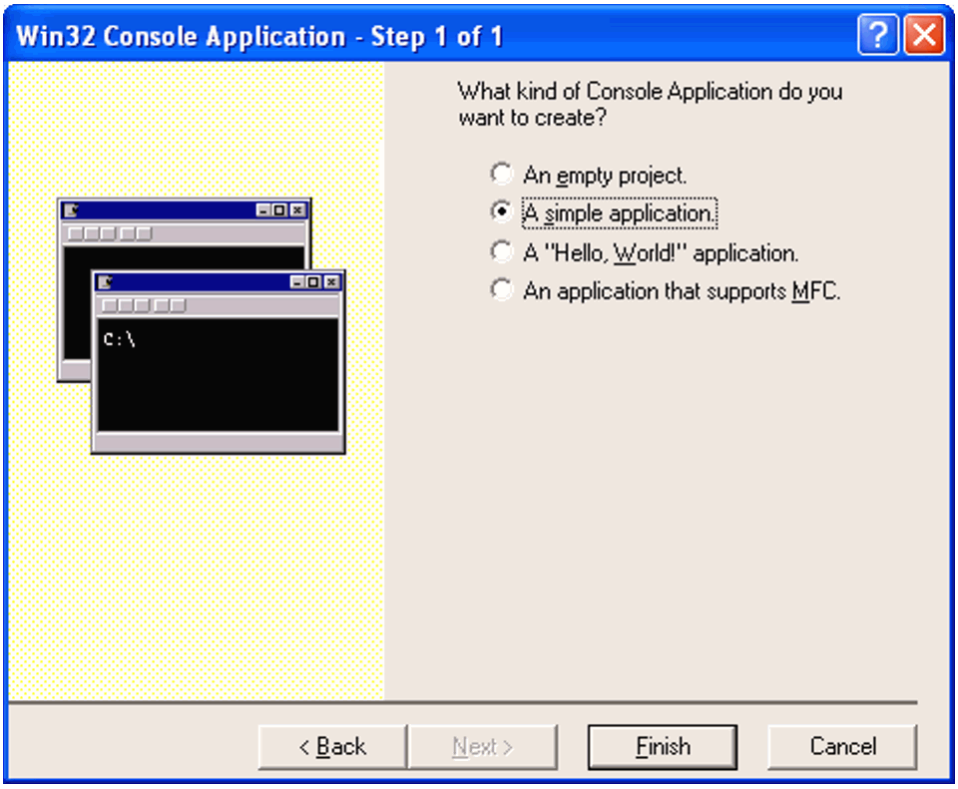


Fig. 9.5: Creating a simple Console Application

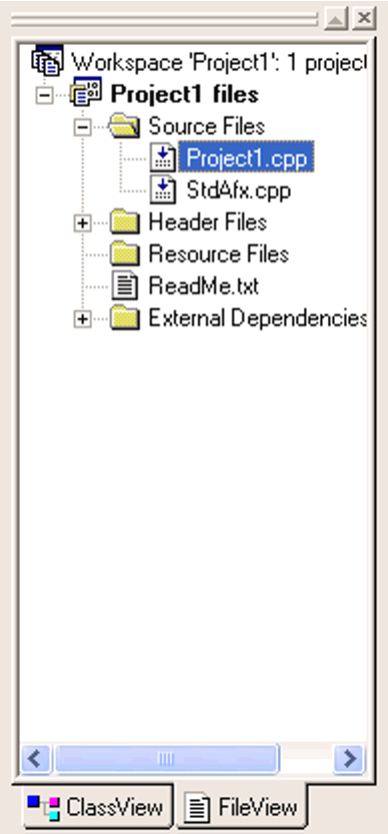


Fig. 9.6: Location of the main C++ source code file, *Project1.cpp*

9.2.2 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"
```

9.2.3 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);
```

9.2.4 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\Probability and Statistics for .NET*, while the default installation path for a user without Administrator privileges is *C:\Documents and Settings***User-Name***\Local Settings\Application Data\WebCab Components\Probability and Statistics 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\Probability and Statistics for
.NET\
\COM Libraries\WebCab.COM.StatisticsDemo.tlb" no_namespace
```

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

9.2.5 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.

9.2.6 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 corresponding 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.

```
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 9.2: Handling a two-dimensional safe array in C++

9.2.7 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.

9.2.8 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 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 9.2.7, 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 9.2 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.

9.2.9 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();
```

9.2.10 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 9.2.1. 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 9.3: 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 9.4: Generic VC++ example (continued)

Chapter 10

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.

10.1 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`"

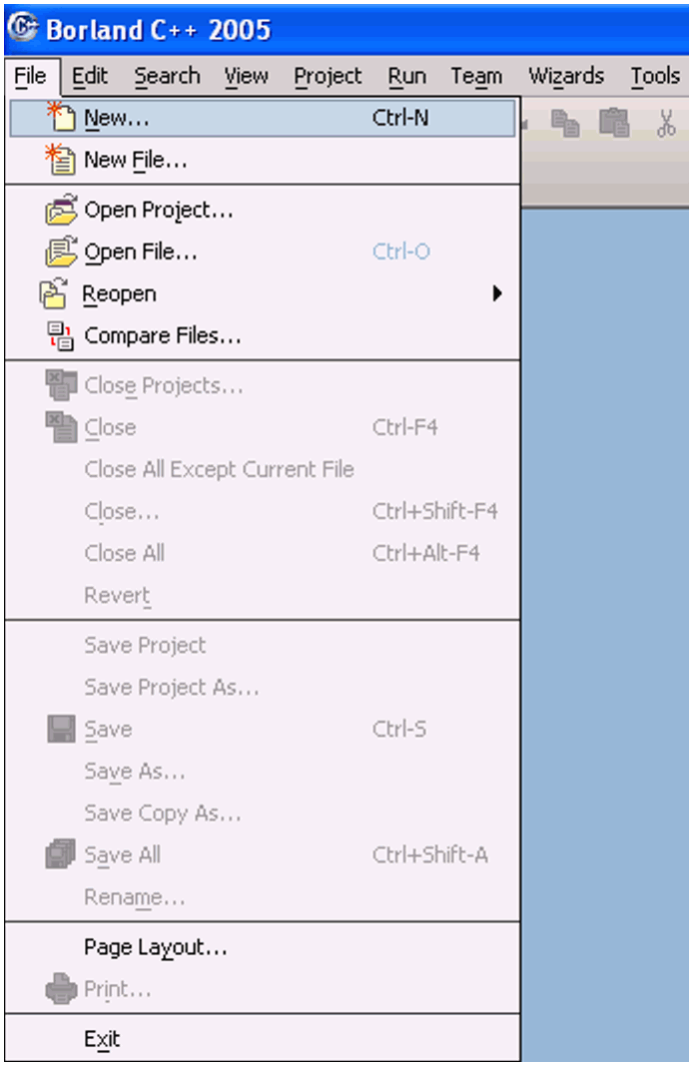


Fig. 10.1: Starting a new project

10.1.1 Open a New or Existing Project

You can start a new Borland C++ Builder Project by going to the File | New... menu (see Fig. 10.1), 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. 10.2.

Type in the name of your project in the next dialog window and click Next (see fig. 10.3). In the next window (figure 10.4) 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 10.5.

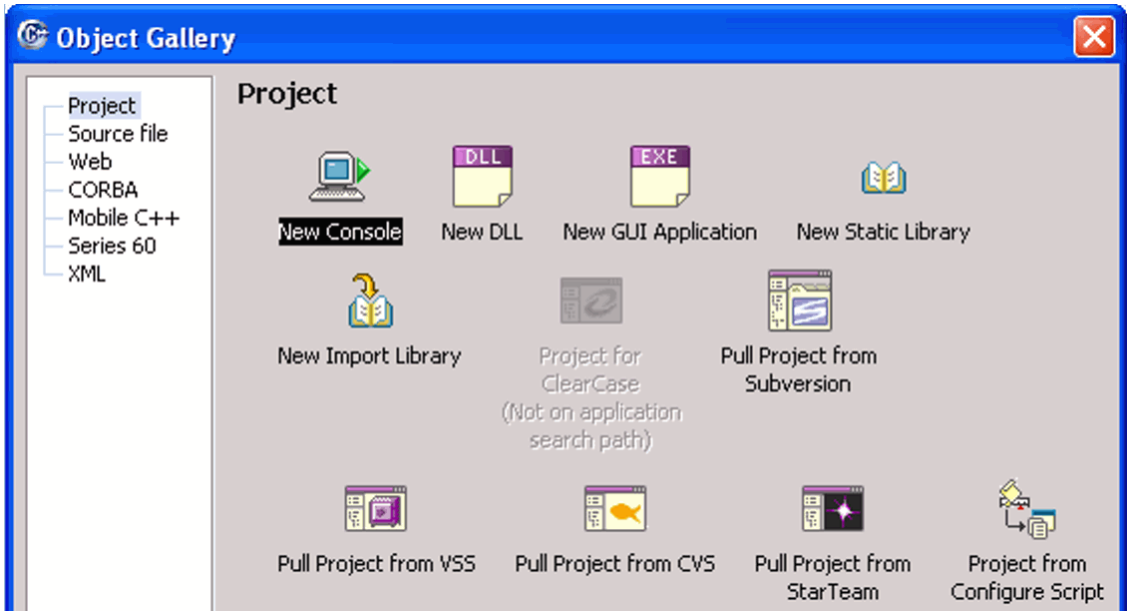


Fig. 10.2: Starting a Console Application

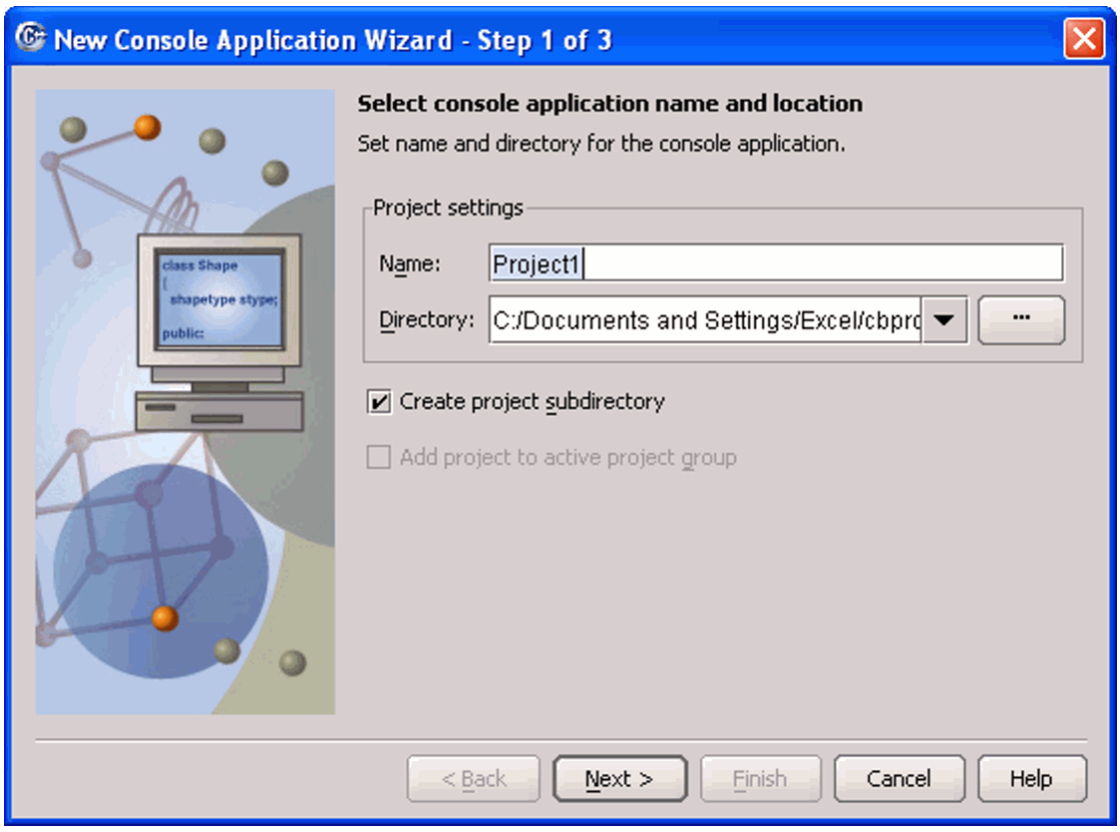


Fig. 10.3: Choosing a name for a new Borland C++ Console Application

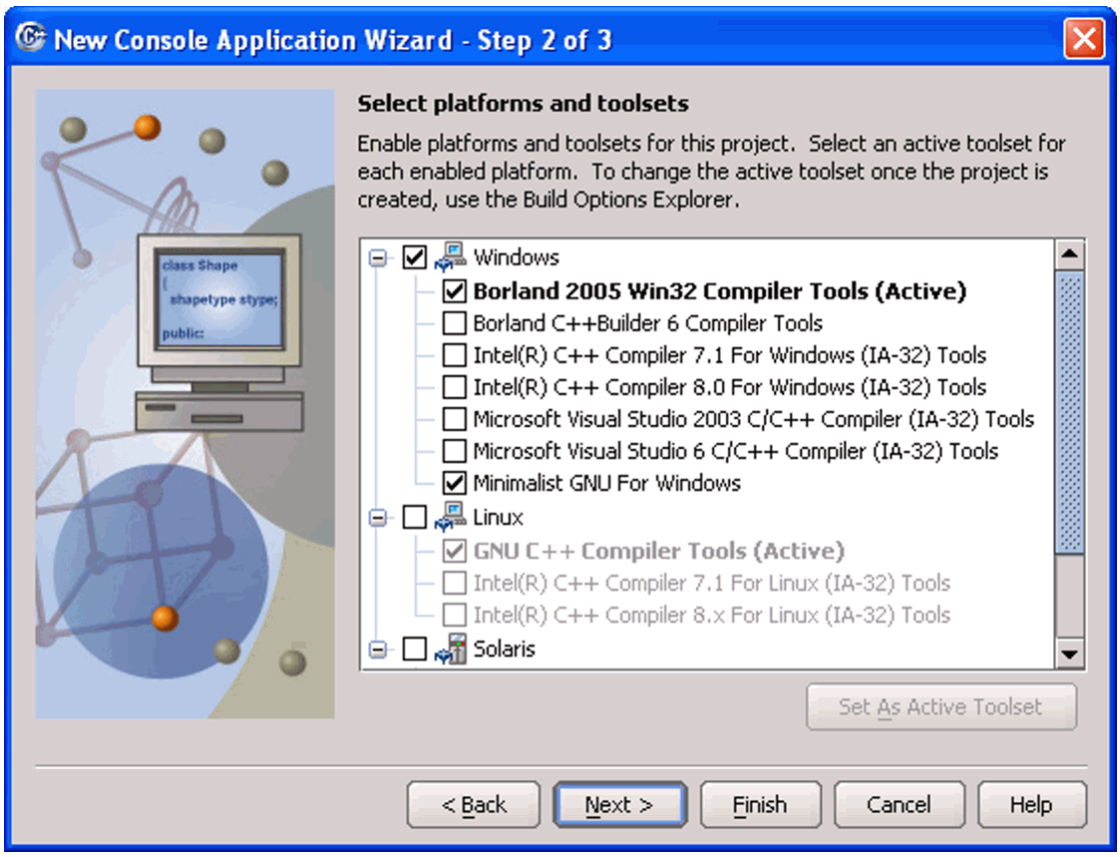


Fig. 10.4: Selecting platforms and tools sets for a new Borland C++ Console Application

10.1.2 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"
```

10.1.3 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);
```

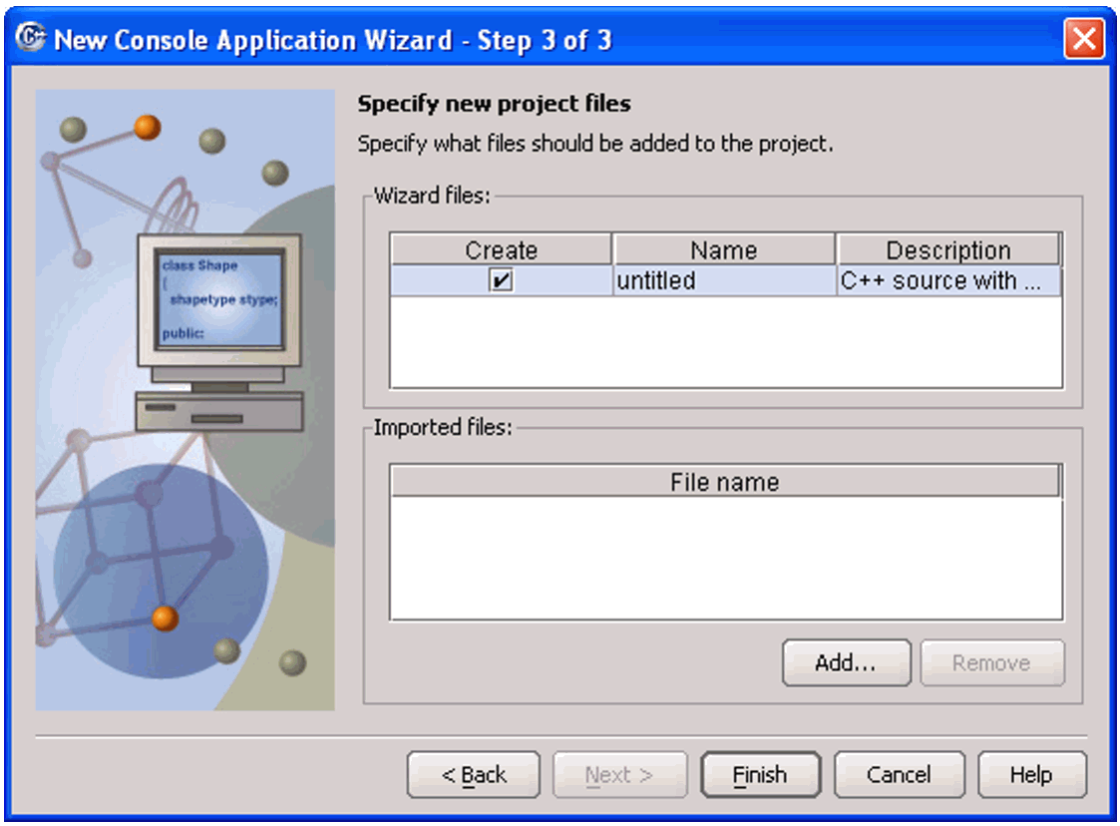


Fig. 10.5: Specifying the project files for a new Borland C++ Console Application

10.1.4 Create a Class Instance

In order to create an instance of a WebCab Probability and Statistics 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.

10.1.5 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);
```

10.1.6 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 10.1 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 10.1: Declaring parameter types and values for a COM method

10.1.7 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);
```

10.1.8 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;
```

10.1.9 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();
```

10.1.10 A Generic Borland C++ Builder Example

In this section (tables 10.2 and 10.3) 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 10.1.1. 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 10.2: 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 10.3: Generic VC++ example – Part 2

Chapter 11

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.

11.1 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 Probability and Statistics for .NET v3.3 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.

11.1.1 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.
```

11.2 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.

11.2.1 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.

11.2.2 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.

11.2.3 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
  ```

  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.

[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.

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

- 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));
    }
}
```

11.2.4 Importing Web services into Visual Studio .NET projects

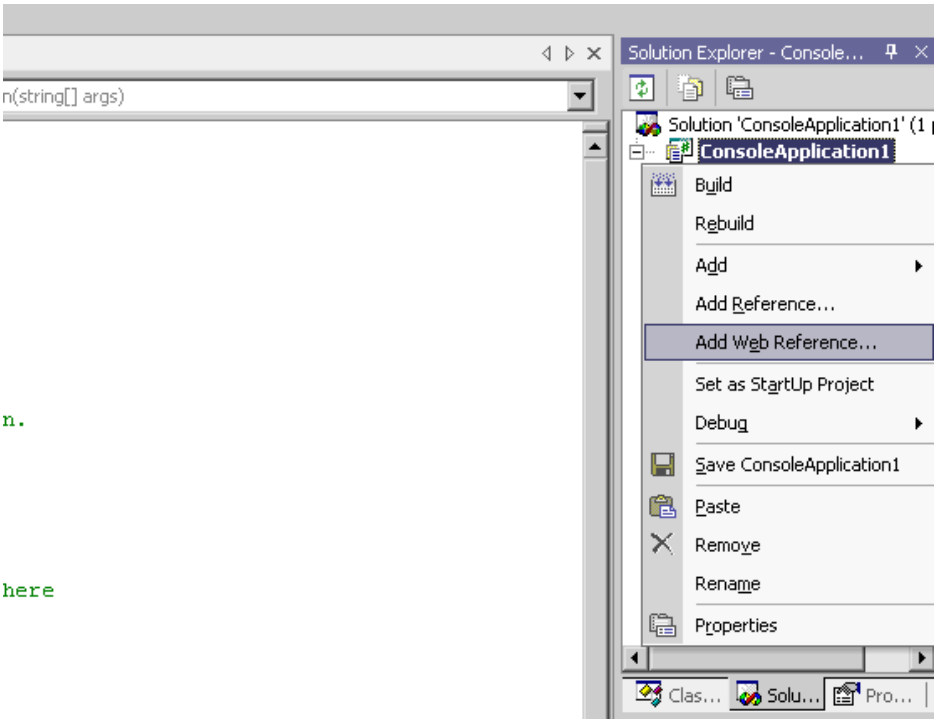


Fig: Adding a Web reference.

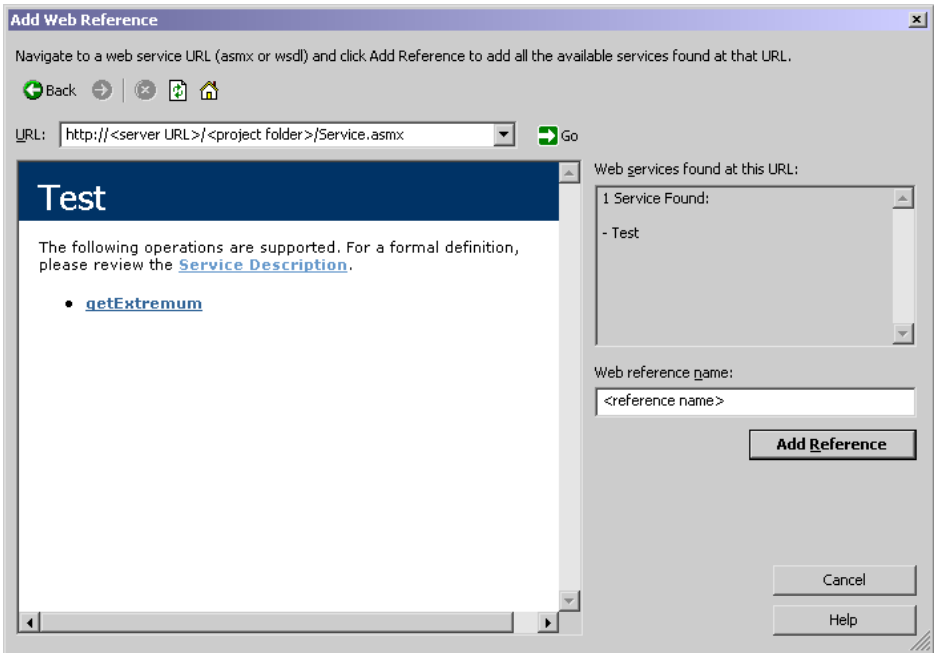


Fig: Adding a Web reference.

11.3 Connecting to a Database with our .NET Libraries

11.3.1 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 Probability and Statistics for .NET v3.3:

- The **"Hypothesis Testing"** Module
- The **"Correlation and Regression"** Module
- The **"Standard Probability Distributions"** Module

11.3.2 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.

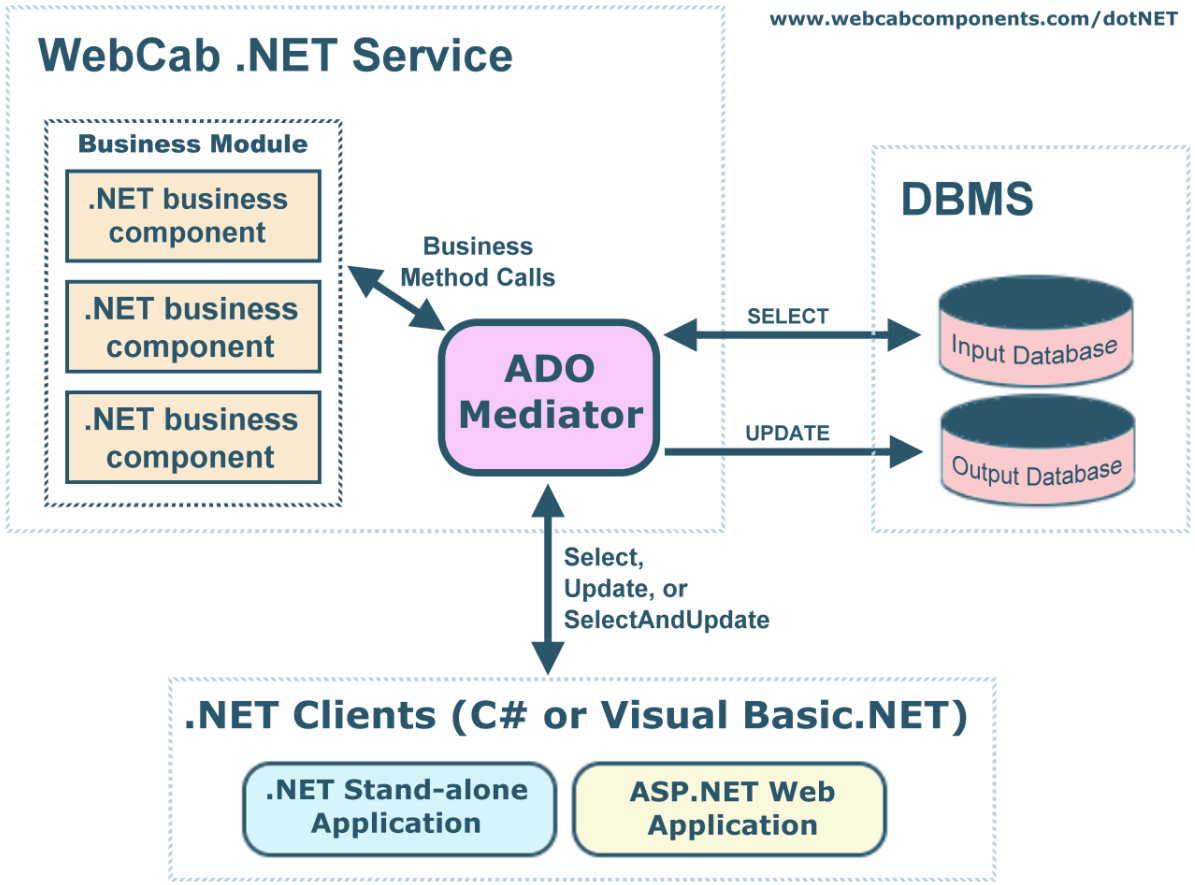


Fig: This diagram describes how the ADO Mediator passes onto the Database and the .NET components its client requests.

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.

11.4 Statistics Module Functionality

This module helps to solve statistics problems related to:

- Measures of Mean
- Measures of Dispersion
- Frequency Tables

The input data can be passed to the component using the methods `SetArray` and `SetBoundaries`. The component keeps the two arrays inside itself so the methods can use them. If you want to change the set of input data there can be called again the methods `SetArray`, `SetBoundaries` for the new data.

To get the set of input data use the `get/setArray` and `Get/SetBoundaries` methods.

11.5 Discrete Probability Module Functionality

The Discrete Probability module offers methods and procedures from the theory of discrete probability. It can be seen as consisting of three parts:

1. Discrete Probability

   Let there be a family of elementary events (represented by a finite set of integer numbers), in a simple space $\mathbb{S}$, and a probability function (also represented by a finite set of real numbers) that associates with each event its probability as a real number. The two sets can be passed to the `SetProbability` method, after the component's creation.

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

Methods: The component offers methods to calculate:

- The probability of an event that is represented by a subset of the initial set of elementary events
- The probability of the union of two events
- The probability of the intersection of two events
- The probability of the complementary of an event

Exceptions: The component will signal the user with the two following exceptions, if any mishaps take place:

- **ProbabilityNotSetException** - thrown if any of the previous methods are called before actually providing the two sets for the component to work with
- **IncompatibleEventException** - thrown if the specified event is not a subset of the initial family of elementary events

Example Suppose there are two unknown events, A and B, and only three out of following four probabilities are known:

- the probability of A
- the probability of B
- the probability of union (i.e. $A \cup B$)
- the probability of intersection (i.e. $A \cap B$)

By applying methods related to the implementation of Bayes Theorem the fourth probability can be calculated.

2. Discrete Probability Distributions and Random Variables

This component also provides methods for dealing with random variables. A random variable represents a function, which associates a real number to each and every elementary event in a simple space $\mathbb{S}$. For a swifter approach, we consider the simple space $\mathbb{S}$ to be the set $\{1, 2, 3, \ldots, k, \ldots\}$. Thus, to work with a random variable, you will only send to the component the *function* and the *probability distribution*, that are represented by sets of real numbers.

Passing Random Variables: If you do not wish to pass the random variable each time you call a method that uses it, the component will let you use, as an alternative, a *random variables table*. You may choose to add random variables to this *table* at the creation of the component, or later, using the `AddRandomVariables` method. The created instance will also provide ways to insert and remove random variables from this table. Each entry will be assigned an index by which every method will be able to identify the random variables. This way of managing random variables is considerably faster and more robust.

Methods: For one or more random variables you may calculate:

- The **mean**

- The **variance**
- The **cumulative distribution function**. This represents the probability of a random variable to take a value below a given real number
- the **expected value** $E[g(X)]$, where X is the random variable and g is some function over X
- The **covariance for two random variables**
- The **population correlation coefficient** for two random variables

3. Basic Probability Laws and Simplifying Expressions

These methods are used to calculate basic probability laws for the *union* and the *intersection* of two or more events. The symbols used inside the component are: "+" for union, "∗" for intersection and $\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots$ for events.

A very important feature of this component is the *simplification of expressions involving random variables*. These methods can simplify expressions involving the means, variance, expected values, covariance and population coefficients of random variables. Within the component we use the following notation:

- M - mean
- V - variance
- E - expected value
- C - covariance
- R - population coefficient

Within the component we will always refer to random variables by the remaining upper-case letters $\mathbf{A}, \mathbf{B}, \mathbf{D}, \ldots$.

Notation: We denote the following operations applied to the random variables A and B:

- **mean** $\rightarrow M[\mathrm{A}]$
- **variance** $\rightarrow V[\mathrm{A}]$
- **expected value** $\rightarrow E[\mathrm{A}]$

Where A and B, are two random variables we have:

- **covariance** $\rightarrow C[\mathrm{A}, \mathrm{B}]$
- **population correlation coefficient** $\rightarrow R[\mathrm{A}, \mathrm{B}]$

We represent the following operators as:

- $+$ for addition
- $-$ for substraction
- $*$ for multiplication
- $/$ for division

We denote real constants by lower letter $a, b, c, \ldots$. Note that you can only use high level brackets. If you do not abide by this notation then the corresponding method will not return the correct result.

Chapter 12

Examples

This chapter is still under development. Please refer to the .NET client examples directories within this package for C# source code examples.

Chapter 13

Guide to WebCab Components

13.1 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.

13.2 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.

13.3 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.

13.4 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.

13.5 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.

13.6 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.

13.7 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