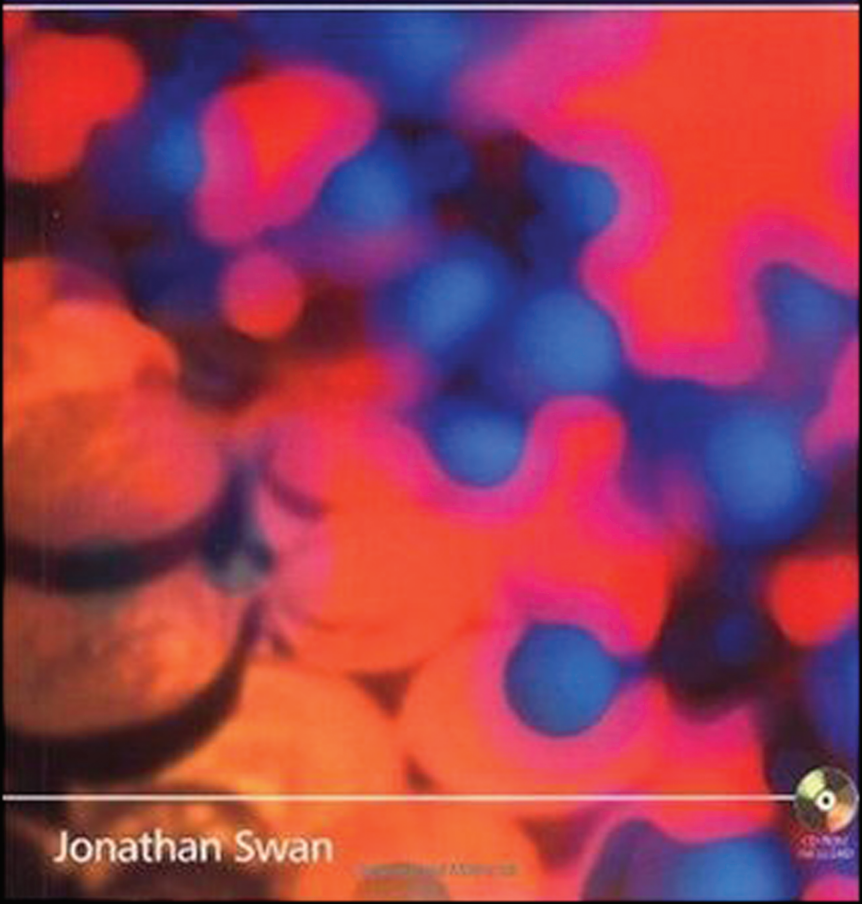
ELSEVIER

CIMA PUBLISHING

Practical Financial Modelling

A Guide to Current Practice



Jonathan Swan

Practical Financial Modelling

A Guide to Current Practice

CIMA

PUBLISHING

Practical Financial Modelling

A Guide to Current Practice

Jonathan Swan

ELSEVIER

AMSTERDAM BOSTON HEIDELBERG LONDON NEW YORK OXFORD
PARIS SAN DIEGO SAN FRANCISCO SINGAPORE SYDNEY TOKYO

CIMA Publishing
An imprint of Elsevier
Linacre House, Jordan Hill, Oxford OX2 8DP
30 Corporate Drive, Burlington, MA 01803

First published 2005

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 0 7506 6356 1

Typeset by Newgen Imaging Systems (P) Ltd., Chennai, India
Printed and bound in Great Britain

To Rebecca, Jack and Jeremy, who still don't understand
what this book is about

Contents

Preface *ix*
About the Author *x*
Context *xi*

1 Model structure **1**
Introduction 1
Two approaches 1
Purpose 3
Structure 3
Workbook structure 3
Inputs 4
Workings 5
Outputs 7
Variations 9
Documentation 10
Reporting 13
Reports 16
Model development 17
Navigation 20

2 Quality control **26**
Introduction 26
Taxonomies of error 26
Audit tools 28
Error values 30
Audit sheet 32
Structural checks 34
Arithmetical checks 42
Financial checks 43
Model map 46

3 Mainly formulae **48**
Introduction 48
Range names 49
Additional name functionality 60
BODMAS 71
Timing 71
Changing time periods 76

Circularities and iteration 79
Array formulae 88
Coercion 89

4 Mainly functions 90
Introduction 90
Logical 90
Lookup 99
Financial 109
Dates 111
Other useful functions 113

5 Model use 117
Introduction 117
Grouping and outlining 117
Data inputs 119
Conditional formatting 123
Custom formatting 126
Protection 131

6 Sensitivity analysis and scenarios 135
Introduction 135
Goal Seek 135
Data tables 136
Scenarios 138
Solver 147
Risk 147
Monte Carlo simulation 148

7 Automation 151
Introduction 151
Recorded macros 152
Iteration macro 153
Assigning macros 155
Written macros 157
Branching macros 158
Quarterly/annual macro 160
Error handling 160
User-defined functions 161

Appendix: Keyboard shortcuts 165

Index 170

Preface

Most of the books on financial modelling that I have come across tend to go long on the financial and short on the modelling. Most of them are full of genuinely useful financial calculations but they offer little insight into how to put them together in a robust and reliable model, in much the same way that a dictionary helps you with your spelling but does not help you to write good prose. To stay with this analogy for a moment, I would describe this book as a grammar that will provide you with a structural and conceptual basis for your financial modelling. I shall assume that you have a good working vocabulary, or the ability to refer to the appropriate dictionary, as required. This book sits between your Excel manual and your finance textbook.

I should state at the outset that there is no agreed 'best' practice in financial modelling – the methodology and techniques used are those which are best suited to the task at hand. In this book we will examine some of the common, generic, approaches you will encounter in financial models today, with a view to understanding the technical background and to appreciate that the same problem can often be solved in several ways, some of which appear better or more reliable than others, and some of which appear counter-intuitive and less satisfactory. The intention is to encourage you to reflect on your own practice in the light of these suggestions, and I am confident that you will be able to generate your own solutions to the problems and issues that follow. Even if you are not convinced by my arguments, by engaging with them you will have greater confidence in your own modelling abilities. You have picked this book from the shelf because at some point you have asked yourself the fundamental question – is this model right?

About the Author

Jonathan Swan is a director of Operis TRG Limited, the training arm of Operis Group plc. He has extensive experience in teaching the use of spreadsheets as a financial analysis tool. Over the past decade he has developed and delivered financial modelling training programmes to many investment banks, international financial institutions, management consulting and accounting firms, in the City of London and throughout Europe.

Jonathan holds an MBA from the East London Business School (University of East London) and is a member of the Securities Institute.

About Operis Group plc

Operis is a London-based project finance advisory firm, well known for its financial modelling expertise and experience. We:

- develop financial models of large transactions for a range of clients which includes financial institutions and project promoters in a variety of sectors and countries;
- advise government clients, companies and consortia in the PPP sector on project definition, bid strategy, funding routes, benchmarking, refinancing and project management;
- provide both formal and informal assurance advice for sponsors and funders in connection with financial models and project documentation developed by other firms;
- have a department of accountants and tax advisors in-house to provide additional advice in connection with such projects.
- are the largest provider of training in financial modelling, to over two thousand individuals in the last three years;
- market software valuable in the development and auditing of large financial models, which has been adopted by three out of four of the world's largest accountancy practices; and
- are currently the only European firm specifically accredited to ISO 9001:2000 for its financial modelling build, model audit and training activities.

The firm was established in 1990 and now has a headcount of 42, making it one of the largest teams devoted to its particular discipline.

Context

People make mistakes

Let us face up to it. The list of investment decisions based on flawed models is large and growing; for example, a cut-and-paste error cost a Canadian corporation $24m; an unchecked economic model resulted in a plaintiff being awarded more than $12m in damages; and a US company blamed a typographical error for misrepresenting its profits by $140 m.* These models were developed by skilled and professional analysts working for world class institutions. The international accounting firm Ernst & Young has estimated that some 80% of financial models contained errors, whilst at the 2003 European Spreadsheet Risks Interest Group (EuSpRIG) conference the model auditing team from PricewaterhouseCoopers declared that they had *never* found a model that did not contain mistakes, and my own auditing team would agree. The reason we so rarely hear of this appalling track record is that the organisations involved invariably close ranks and matters are resolved outside the court room.

It doesn't happen here

Although most firms would profess to have modelling standards and procedures, the reality is that responsibility for the financial modelling function is often diffused, and individual analysts apply their own interpretation of quality control. I have even heard directors claiming that 'we only recruit the best MBAs from the most prestigious business schools' as if this mantra somehow protects them from poor modelling and its consequences.

Human error has been the subject of academic and operational research for many years, and there is a rich literature which includes the psychological analysis of error, various taxonomies of error, and models of human performance. Financial modelling has been investigated in this context for over two decades and the published research, although somewhat limited and circumscribed, is remarkably consistent. In order to understand the causes of error the researchers have attempted to investigate the modelling process and those carrying

*Further examples can be found at the European Spreadsheet Risks Interest Group website: www.eusprig.org

out the modelling activity. Unfortunately investment banks and large corporations seem reluctant to allow their analysts and managers to be used as subjects, and given that most researchers are based in the business schools, the research guinea pigs are typically MBA or undergraduate business studies students.

A consequence is the difficulty in setting a meaningful modelling exercise for research purposes – most tend to be fairly limited, with a small number of inputs leading to a relatively simple set of calculations. Given these constraints however, the results highlight both inconsistencies in the way in which subjects develop models, and perhaps more importantly, a general lack of diligence in checking through completed work.

It might be assumed that the business school student is not representative of the financial analyst of the investment bank, but in fact there is one key similarity: it is highly unlikely that either of them have ever received formal training in financial modelling. Indeed, many organisations (and individuals) equate 'competency with Microsoft Excel' with 'competency in financial modelling', which reveals a fundamental lack of understanding of the skills and knowledge required.

The principle of error reduction

I have taught practical financial modelling for several years. The methodology I teach is based on that used by my colleagues, who have worked on some of the most complex financial modelling assignments in the industry worldwide. This methodology is not unique, it certainly is not rocket science, and I would never suggest that it is the only or 'best' methodology. It is my intention to introduce the methodology in this book, and in doing so, compare and contrast it with alternative approaches which are in common use and might be described as current practice.

The delegates attending my courses are highly motivated finance, banking, or management professionals who bring a range of modelling experience with them, and my exposition of our methodology is often the stimulus for robust debate. However, although we may disagree on the finer points, I am usually able to convince them of the validity of our approach because it is based on what I call the 'principle of error reduction'. This simple concept is based on our own experience and that of others in the business, where we recognise that certain modelling operations are more error-prone than others. Going back to the research referred to above, it seems that humans have a natural error rate to the order of 1%. Some of the research recognises that financial model development is an activity similar to computer programming and which has been extensively studied. It seems that computer programmers anticipate an error rate of around 3% and spend upto 40% of their time checking and reviewing their own work to reduce this rate even further. Is it worth asking how much time the average financial analyst spends on model audit and review? And yet I still come across very intelligent people who claim that their work is error-free. I believe in adopting a pragmatic approach which accepts that errors are inevitable but then seeks to minimise their occurrence and to enhance their detection.

There is no methodology for error-proofing: there is no way of ensuring that a plus is typed instead of a minus. The principle of error reduction enables us to recognise potential sources of error and to either substitute them with a more reliable technique, or to implement an audit check which can be used to test the validity of the routine.

In the early days of financial modelling, users and sponsors were willing to accept a certain element of ambiguity; that the model was, of course, only an approximation of the transaction. In recent years there has been a trend to see the model as the ultimate reality with generalised assumptions taking on the guise of hard fact. It might be helpful to remind ourselves of the simple adage: is it better to be vaguely right, than precisely wrong?

Definition

Here's a working definition: the principle of error reduction accepts that errors are inevitable. Some techniques are more prone to error than others. We reduce the risk of error by using alternative techniques and a consistent methodology that serves to enhance the detection of errors when they occur.

This book

The overall structure of this book reflects the modelling process: we begin by considering how model purpose can dictate model structure, followed by an exploration of various layouts which are designed in consultation with the users or the model sponsors. This then leads into techniques used in model building, ensuring that quality control is built in from the outset. We then look at techniques which enhance the usability of the model but at the same time protect the model from unwanted amendments.

Chapter 1: Model structure

This sets out a number of issues relating to model structure and some suggestions about what might constitute a good model layout. The structure of a financial model will depend in large part on its purpose and use, which generally means that there is no single blueprint. However, by introducing a top-down methodology which includes user involvement from the outset, and by focussing on the outputs required of the model, we can more easily work with and manage our users' expectations and in so doing clarify the modelling task. Model building is an intangible process and we must therefore emphasise the tangible evidence of our work: the printouts. Right from the very first moments of developing the model structure we must be prepared to generate reports that can be seen and used by the model owners.

Chapter 2: Quality control

Quality control should be an integral part of the model development process, and not a set of checks or procedures to be carried out on completion. A fundamental part of the process, even for relatively minor models, is to set up an audit sheet on which are listed the results of the key checks that should be carried out.

Chapter 3: Mainly formulae

A financial model is all about calculations, and this chapter sets out a number of ways in which we can make our formulae clearer and easier to understand. One of the most contentious issues in current modelling practice is the use of range names and the arguments for and against are rehearsed at some length. In many models the issue of timing is important,

where the occurrence and duration of key events impact on dependent routines. Techniques such as masking offer simple solutions to what can often appear to be difficult problems. The problem of circular formulae and the use of iteration is also described.

Chapter 4: Mainly functions

A high level of competence in modelling can be achieved through the knowledge of a handful of Excel functions. Building on the ideas expressed in chapter 3, we extend our abilities to solve complex problems by exploring the logic and the lookup-type functions, along with a handful of date and other functions.

Chapter 5: Model use

Our users rely on us to set up the calculations and functionality of the model for ease of use, and so we need to understand how they might approach the model and anticipate ways in which we can both help them use the model sensibly and without damaging the underlying code or structure. Functionality such as data validation and drop-down boxes can simplify user–model interaction.

Chapter 6: Sensitivity analysis and scenarios

One of the key reasons for building models, as opposed to spreadsheets, is that we wish to explore the effects of changing input values on the corresponding outputs. This is either by changing or flexing a small number of inputs (sensitivity analysis) or by running scenarios. Techniques for data tables are described, along with some of the common ways to manage scenarios, including CHOOSE, VLOOKUP, and multiple input sheets.

Chapter 7: Automation

Good modelling practice suggests that the analyst should generally avoid using macros and to write appropriate code in the worksheet. However, the sheer repetitiveness or complexity of some tasks means that automation is a genuine option, and so a good modeller should have an understanding of basic macro techniques in the context of good practice. We explore recording and writing macro code and assigning macros to keyboard shortcuts, worksheet buttons, and menus. We also look at user-defined functions.

Microsoft Excel

I have tested all the exercises and examples in the following chapters with all versions of Excel from 95 through XP and 2003, but not with Lotus 1-2-3 or Quattro Pro. We shall be using Microsoft Excel throughout, and I have endeavoured to ensure that we use native Excel functionality, without resorting to macros, add-ins or third party software. I am aware that many organisations do not care much for the endless round of Microsoft updates and new software versions and that you may be working, quite happily, on 97 or 2000. The screen shots are from Excel 2003, but the exercises and illustrations work in all recent versions of Excel, unless stated otherwise.

The Lotus 1-2-3 legacy

Microsoft Excel is the industry standard software. I don't feel it necessary to provide a history of the spreadsheet but it is important to recognise the role played by Lotus 1-2-3 and the way in which it still influences financial modelling practice today. Many analysts, myself included, achieved a high level of competence in the days when 1-2-3 dominated the market. In the time since, these individuals have progressed beyond the analytical function and are now in middle and senior management. Modelling is no longer part of their job description, and instead they manage, perhaps at a distance, those who have the day-to-day responsibility for developing financial models.

The problem is that the modelling methodologies and techniques of 1-2-3 are not necessarily the most appropriate for modern modelling, but management is unwilling or unable to perceive the need to discard the old way of working and therefore does not encourage their subordinates to learn new methods. We often see models in which the inputs have been coloured blue – a tradition that started in the old days of single sheet spreadsheets. Another 1-2-3 convention is that of starting all formulae with a + sign instead of =.* In the earlier versions of 1-2-3 cell contents could only be deleted using the cumbersome /Range Erase command sequence, rather than using the Delete key. Some users developed the workaround of using the spacebar to clear cells, which of course inserts an invisible space (text) character into the cell. This still causes problems such as generating spurious #VALUE! errors in dependent formulae. A further feature of the 1-2-3 modeller is an over-reliance on a very limited set of functions, typically IFs and VLOOKUPs, whereas we now have a range of additional functions and techniques available and which will be explored in this book.

Without doubt, 1-2-3 was the leading spreadsheet of the early 1990s, and that in good hands it was an impressive and robust tool. Personally I was not convinced about the value of the Microsoft product until Excel 5 was released. The point is that Excel dominates the market and is the spreadsheet application of choice for the vast majority of those who produce financial models.

Conventions

I have travelled quite widely in the course of my teaching and I am well aware of the international differences in formulae, functions, formatting, and most of all the keyboard shortcuts. With a view to an international readership I have tried to anticipate possible problems when working on the exercises in this book, and in several cases I point out where specific shortcuts do not work. In this book I will use UK/US settings for my routine work I use the following conventions:

```
=IF(E25>1000.00,E25,0)
```

*I have also found that accountants have the habit of doing this because of their tendency to use the number keypad and the + sign is close to hand whereas the = sign is on the main keyboard. This is also the case in some countries (Switzerland, Germany) where the keyboard layout means that the + sign is easier to type. In any case it makes absolutely no difference to the result but in my opinion just looks rather scruffy.

Elsewhere I should write this as

=WENN(E25>1.000,00;E25;0) or =SI(E25>1.000,00;E25;0), that is, using the local name for the function, the ; semicolon as the argument separator, and recognising the appropriate thousands and decimal notations.

Anticipating that some readers may wish to copy formulae directly from the page, I have elected to show them exactly as they should be written. This may be at the cost of clarity, but entering spaces into calculations can cause problems. For example,

= sum (E24:K24) generated a #NAME? error, because in this case Excel does not recognise the SUM.

Keyboard shortcuts

I encourage the use of keyboard shortcuts to make our work more accurate and efficient. Learn the shortcuts most relevant to the work you carry out routinely.

Menu commands

In this book I will use the following notation for Excel menu commands:
Insert, Name, Create

The underlined letters indicate the keys used in the shortcut: Alt+I, N, C

The plus sign indicates that the Alt+I are pressed together, released, and then the N is pressed and released, followed by the C.

In most dialog boxes, the OK button is the default, which means we can simply press Enter to confirm the command. Esc will of course cancel the operation.

Dialog box commands

The command sequence Tools, Options, Calculation, Iteration has no underline for the Calculation element. This is because Calculation is the name of a tab in the dialog box. If, for example, we use Alt+T, O, the Options dialog box normally opens on the View tab. To select the Calculation tab, press Ctrl+Page Down, or Ctrl+right arrow.

To select commands within the dialog box, use the Tab key (or Shift+Tab), or better, press Alt+the underlined letter in the command (check boxes and items in lists can be selected using the Spacebar).

The full keystroke sequence for Tools, Options, Calculation, Iteration is:

Alt+T, O, Ctrl+Page Down, Alt+I.

If you are using Excel in a language other than English, substitute the appropriate command and shortcut sequences.

Character shortcuts

Many shortcuts avoid the menus altogether. Ctrl+S is the shortcut for File, Save. The most basic of these are listed within the menus themselves. These shortcuts tend to be language-independent – Ctrl+P seems to print on all versions of Excel I have used so far. However, shortcuts which use specific characters may not work. For example, Ctrl+[(open square bracket) serves to select precedent cells on an English language installation of Excel. Although the [character exists on other keyboards, it may not work as a shortcut. The equivalent for Ctrl+[on a German keyboard is Strg+Ü.

Menu and toolbar shortcuts

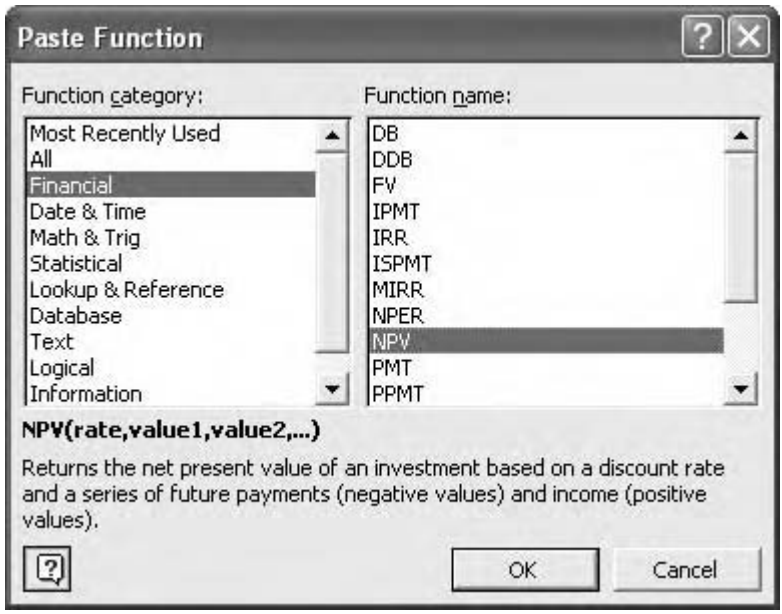
An alternative method of activating the menu bar is to press F10. The shortcut menu which is normally shown by right-clicking with the mouse, can be displayed by pressing Shift+F10.

You can even access the toolbars using the keyboard. Press F10 to activate the menu bar, then press Ctrl+Tab (repeatedly) to activate each toolbar in turn. Use the arrow keys to scroll across the toolbars and press Enter to select the button. If you press F10 followed by Shift+F10 you will see the Toolbars menu.

Further information

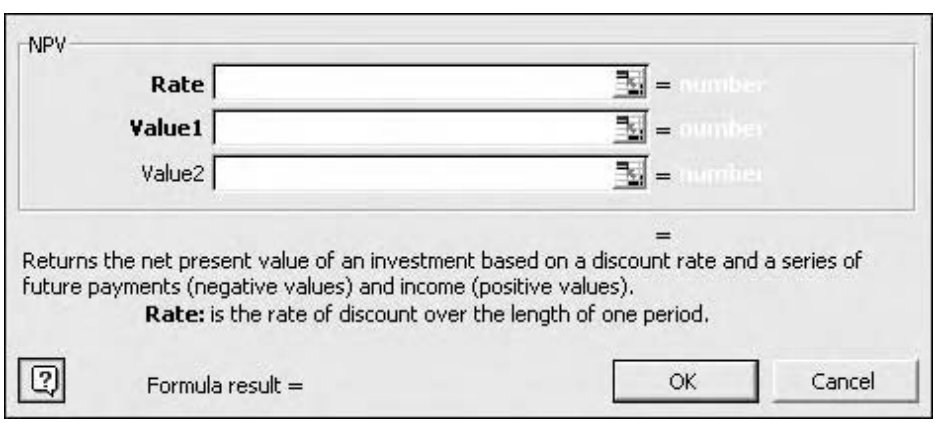
A list of keyboard shortcuts used in this book is provided in the appendix. For more information about these and other shortcuts, use Excel Help (F1) and search for 'keyboard shortcuts'.

Function Help



Shift+F3 Paste Function

When writing functions into the spreadsheet it can be difficult to remember the sequence of arguments and so I find it helpful to remember that we can press Shift+F3 to fire up the Paste Function dialog box; or once we have typed the function name we can press Ctrl+A to bring up the function window. It is worth noting that the function window can be detached and moved around the screen with the mouse.



Ctrl+A Function window

CHAPTER 1

Model structure

Introduction

A good model is easily recognisable – it has clearly identifiable results based on clearly defined inputs. The relationship between them can be tracked through a logical audit trail. There is little empirical research into the needs and expectations of model users, but our experience suggests that most users want to know the location of the key results. The ability to perform sensitivity and/or scenario analysis is also very important, so the location of the key inputs should be explicit.

In this chapter, we will consider some of the general conventions concerning model structure. It is tempting to refer to them as rules, but in almost every case the suggestion that 'we must always do this …' can be immediately countered by the observation 'except when we don't'. It is important to recognise that when setting out a rule-based methodology we should have techniques for proving conformance with such rules and for locating and identifying exceptions. This forms the basis of Chapter 2.

The demonstration workbooks for this chapter are located in the Chapter 1: Model structure folder on the CD-ROM.

Choosing the right tool

In a book about financial modelling it may seem obvious that we are talking about spreadsheets, but remember that this isn't always the case. Very recently I met with a client who was trying to design a model that would be manipulated in several ways to generate management information relating to the operational costs of a number of business units. The calculations were arithmetically simple, and I soon realised that the analyst wanted to perform *data* modelling rather than *financial* modelling. It would be far more efficient to use a database application than a spreadsheet. We had a quick refresher session on Microsoft Access, sketched out an appropriate table and query structure, and the task was completed by the end of the afternoon.

Two approaches

Although the structure of a model will depend for a large part on its purpose, there are a number of ground rules which should be recognised. I always recommend a **top-down**

approach: we identify the purpose or objective of the model first, followed by a consideration of the usage of the model. Consider the following simple examples:

1 Model A will be used to calculate the net present value and internal rate of return of a manufacturing project, to be used by the company's management.
2 Model B is a loan calculator, which will be used and re-used by a number of colleagues.
3 Model C is to produce consolidated monthly accounts using information from several business units, to be reported to the management.
4 Model D is a timesheeting system.
5 Model E is a budgeting model which will be used over a period of time, and will require the actual figures to be compared with the budgeted figures as they become available.

In the first model it is likely that we will be required to carry out sensitivity and scenario analysis, possibly using risk techniques. It is likely to be a one-off development, where we would build it, use it, and probably discard it once the project goes ahead. The loan calculator, however, is specifically designed for multiple use, and for multiple users of unknown modelling experience. In this case, we would need to think about providing documentation and perhaps writing macros to automate the use of the model (and restricting the ability of the users to break anything!). The third model will generate standard management reports but the complexity will lie in obtaining and organising the input information. It might be that we would have to think about linking to the spreadsheets developed by each individual business unit. The timesheeting system would probably be developed as a template, and a single sheet should be sufficient. The budgeting model is a work-in-progress, in that it will be used over a period of time, during which the actuals will be entered into the model for comparison with the forecast or budget values, and reference will be made to last year's results for comparison.

Already, in each case, we are thinking about the overall structure and function of the model, before concerning ourselves with the detail, and ideally we are engaging our users or sponsors in the process. Unfortunately, we find that the majority of modellers adopt a **bottom-up approach**, in which the collection and input of the raw data takes priority. This results in a rapid and unstructured early development phase, followed by a problematic and time consuming late development phase in which the analyst attempts to structure and restructure the earlier work. Quite often the model grows by a process of accretion, in which different model elements are bolted on to the existing code, with some being definite enhancements whilst others do not really seem to do much. I also refer to the bottom-up concept as the 'stream of consciousness approach': a sequence of ideas thought up one after another, but without necessarily taking into account the relationships within the model itself. This type of modelling also tends to be idiosyncratic, by which I mean that each model, regardless of purpose, is as distinct and individual as the analyst who created it. It is often quite difficult for colleagues to understand the model, and in the absence of the model builder it can be almost impossible to have full confidence that the model is actually doing what it is supposed to do, and because the users or sponsors have not been involved in the development process the results themselves may be unsatisfactory. Quite often discussions about such models become confrontational rather than co-operative.

Purpose

Returning to the top-down approach, we might summarise the key issues as being 'what is this model for?', and to an equal extent, 'who is this model for?' This means that we give careful consideration to model purpose and use before even thinking about firing up Excel. I often take a piece of paper and sketch out the model layout and structure. The first task in the spreadsheet is to design the model outputs, and by outputs we mean the physical reports that will be generated from the model. In doing this, we can then show others the outcomes we intend to achieve – without the numbers, of course, but in terms of the deliverable we hope to produce. On a recent training assignment with a German investment bank, I was asked if it was possible to develop a standard company valuation model. I was able to liaise with colleagues in London who prepared various drafts of the outputs, and by using an iterative process with local staff we were able to produce an agreed model structure by the end of the week. My colleagues were then able to set about the task of writing the model and the whole transaction was turned around in a very short time.

The top-down approach means that the outputs are agreed at the outset. From the modelling perspective, this then offers a work plan: the modelling assignment is simply to complete all the appropriate rows on the outputs sheet. And in doing so, the outputs then act as a work record, so that we can print the model at a moment's notice to show colleagues, management, or the client.

Although we are considering model development, we can use the same approach when reviewing models: the key question is still 'what is this model for?' I like to set myself what I call the "2 Minute Rule" – can I identify the key results of someone's model within the first two minutes of examining it?

Structure

As mentioned above, model structure will depend on model purpose. As with many issues in financial modelling, different modellers will have different opinions concerning model structure, and it would be foolish to suggest that there is a best practice that could work for all models at all times. But there is some agreement about what might constitute good practice, and so we will review the key ideas and then look at some variations on the theme. Some of the ideas which follow may seem quite counterintuitive, but generally the principle of error reduction applies, and the time taken on developing a robust model structure will be repaid by reduced audit time and a flattening of the learning curve for users. I will not present these ideas as rules to be followed rigidly, because in almost every case there are valid exceptions.

Workbook structure

A very old modelling principle, from the first days of multiple sheets, is that each sheet should have the same layout and that each column should have the same function on each sheet. For example, column E is quarter 1 of the first year of the forecast period, on every sheet in the workbook. The operational researchers have shown that if sheets have different layouts the risk of error increases as the developers or users have to orientate themselves to the layout of each sheet, and that levels of confidence are generally lower.

Inputs

The inputs or assumptions sheet

This is where you should store all the numbers that are used in your model. It is generally agreed that it is very sensible to isolate the inputs or assumptions of the model. The premise is that you or your users should be able to change the numbers used in the model, but not the formulae. When you look at your Excel screen, how do you know if you are looking at numbers or calculations? The simple answer is that you click on the cell and inspect the contents on the formula bar, but this is not particularly efficient. The suggestion is that you keep all your inputs on a separate sheet. If we have a separate inputs sheet, we can protect all other sheets in the file, so that users can flex the model and run sensitivity analysis without breaking anything.

You should always be able to track an assumption right back to its source, be it a data book or project document, and it should be expressed in the same units in the inputs, in the outputs, and in the documentation.

Documentation

I would recommend that inputs should be documented – there are three types of data you can put into a model: publicly available information, commercially sensitive information, and the 'plug' number (i.e. an imaginary or temporary number). The latter should be very clearly identified. A few years ago I wrote an example of an interest calculation in response to an enquiry from someone who had attended a course of mine. A couple of months later I was dismayed to see that the analyst had simply copied and pasted this routine from my email into his model. The interest rate I had used was purely hypothetical and we both learned an important lesson – I now clearly identify my plug numbers with colour and document them with cell comments.

Comments and text boxes

Excel is not particularly good at handling large amounts of text. Cell comments (Shift+F2) are very useful but of limited functionality. Do not be tempted to use merged cells as these break down the underlying structure of the spreadsheet. Although a cell can contain a substantial amount of text and can be formatted as required (use Alt+Enter to wrap text within the cell), the cell will not expand automatically, and we can end up with an irritatingly large entry in the formula bar. Cells can contain up to 32,767 characters, of which only the first 1,024 will appear in the cell (formulae are restricted to the 1,024 limit). Large amounts of text should be placed in text boxes. These can be created from the Drawing toolbar, and have the benefit that they can be easily edited, formatted, resized, and moved.

As the text box is an object, under Tools, Options, View, we can choose to Hide All so that the box is neither shown or printed. The print option can also be set using the text box shortcut menu (right-click on the box border) and choosing Properties in the Format Text Box dialog.

Accuracy

When setting up the inputs sheet it is appropriate to determine the level of accuracy and the level of detail that is required. In some types of model we might start out with ball-park

figures and then gradually refine the detail. With others we may be able to accept some imprecision or approximation, but some may require a high level of detail and accuracy from the outset. I often point out that it is the discrimination and common sense applied in selecting the correct inputs and excluding those that are trivial or irrelevant that can make or break a model. The problem is that this ability only comes with experience.

Colour

Some people suggest that it is sufficient merely to colour the inputs wherever they are located in the workbook. I am not too keen on this approach, because using the principle of error reduction these analysts have to remember to colour the cells every time they enter a value. Forgetting this even once means that the numbers are lost in the mass of calculations (although we do have techniques for locating them again in Chapter 2).

Generally a consistent approach to the use of colour throughout the model can be very helpful – if you have ever played around with Visual Basic you will have seen how colour is used to indicate different elements of the macro code. Sensible use of colour psychology can help others grasp the layout of your work. I prefer to use fill (background) colours – font colours don't always stand out, especially if the cell is currently blank or has a number format which returns a "–" for zero values. Don't use too many colours, do make them different, and always remember that around 10% of your colleagues are affected by some form of colour blindness.

Absence of calculations

The final point about the inputs sheet is that it contains no calculations whatsoever, because that is the function of the workings sheet. The immediate exception to this sweeping generalisation is that data tables (Chapter 6) must be located on the same sheet as the input being tested. Also, I would not consider a link formula as a calculation. If we want to use the same production figure for each year in the forecast period, it is a simple exercise to write the figure in the first cell and put link formulae in the rest of the row. Other than this, the inputs sheet is made up of raw numbers.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2	Financial year ending			2004	2005	2006	2007	2008	2009	2010	2011	*YearsIn*	
3													
4	Inflation rate			3%	3%	3%	3%	3%	3%	3%	3%	*InflationIn*	
5													
6	Oil												
7		real terms price (2004)		18	18	18	18	18	18	18	18	*OilPriceIn*	
8		production (thousands units/day)		0	4	4	6	6	6	6	0	*ProductionIn*	
9		production (days)		365	=D9	=E9	=F9	=G9	=H9	=I9	=J9	*ProductionDaysIn*	
10													
11	Operating costs, in real terms												
12		unit, variable (2004)		0	3	3	3	3	3	3	0	*CostsUnitVariableIn*	
13		fixed (2004)		0	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	10,000,000	0	*CostsFixedIn*	
14													

Inputs sheet with cell links

Workings

The workings or calculations sheet

This is perhaps the more controversial issue when considering model structure. The suggestion here is that all the calculations used in the model are located on a single sheet, which by implication can then be rather large. However, the operational researchers have shown that the use of multiple sheets increases the risk of error, especially in large models

where it can be difficult to form a mental map of the overall model layout and the relationships between different elements on different sheets. The principle of error reduction therefore applies, and we enter all the calculations on a single sheet.

Logic cascade

Quite often we find that in the process of building the workings, a cascade effect is introduced, in which logic flows from left to right and from top to bottom (but not always). The flow is in general linear, which can be of great benefit in tracking logic and debugging errors. The audit techniques and tools in the Chapter 2 work very effectively with this methodology.

The size of the sheet

Some people express concern about the potential size of the workings sheet. Variations might allow for the workings to be spread over several sheets to make them manageable, but the concern is that the overall linear flow of information from inputs to workings to outputs is compromised by workings logic flowing in the reverse direction. The judicious use of grouping and outlining techniques (Chapter 5) and the use of colour can make a large workings sheet less intimidating.

Some people complain that a lengthy workings sheet would be impossible to work with. I tend to point out that the conceptual model is already established if we have ever looked at a large document in a word processor. Imagine if Microsoft Word used the same sheet layout as Excel.

No numbers

As with the earlier observation about the absence of calculations on the inputs sheet, we should also note that there should be no values on the workings sheet (but we will consider an exception – the base column – later on). This means both input values proper, and also hard-coded values which have been typed directly into a calculation, for example, =E117*1.175. In Chapter 2 we will look at ways of detecting such slips. Values from the inputs sheet are brought through to the workings by the use of link formulae:

```
=Inputs!E5
```

I would strongly recommend that we avoid writing calculations that combine input links with workings formulae, for example

```
=D10*(1+Inputs!E5)
```

I am quite happy to accept that this formula works, but from an audit/review perspective it requires us to check references on two sheets rather than one. I describe this type of formula as a three-dimensional calculation and I will consider it further in Chapter 2. At this stage it is worth noting that by using links to pick up values from the inputs sheet, and then to write calculations based on the links, we end up with a full audit trail on the one sheet.

We will explore the functionality of the workings sheet in more detail in the following chapters.

	A	B	C	D	E	L	M
1							
2	Financial year ending			2004	2005		
3							
4	Inflation						
5		**rate**		**3%**	**=InflationIn**	***InflationRate***	
6		index	1	1.03	=D6*(1+InflationRate)	*InflationIndex*	
7							
8	Price						
9		**real terms (2004)**		**18**	**=PriceIn**	***PriceReal***	
10		money terms		18.54	=PriceReal*InflationIndex	*PriceMoney*	
11							
12	Production						
13		**production (thousands units/day)**		**0**	**=ProductionIn**	***ProductionTUD***	
14		**production (days)**		**365**	**=ProductionDaysIn**	***ProductionDays***	
15		total units/year		0	=ProductionTUD*ProductionDays*1000	*ProductionAnnual*	
16							
17	Revenue			0	=ProductionAnnual*PriceMoney	*Revenue*	

Workings sheet with links to inputs sheet

Outputs

Rationale

If the concept of the single workings sheet is controversial, the design principles of the output sheets are the most counterintuitive. The two key suggestions are that the output sheets are populated with links to the workings sheets, and that none of the output sheets are linked to each other. To explain this, let us consider the calculation of depreciation. If my outputs include the pro forma financial statements such as the Cash Flow, the Profit and Loss (P&L), and the Balance Sheet, we would expect depreciation to be reported on the P&L and the effect of it would be to reduce the book value of the fixed assets on the balance sheet. However, rather than feeding the P&L depreciation through to the balance sheet, the calculations are set out on the workings sheet, as shown below.

29	Fixed assets: Development						
30	depreciation rate		20%	20%	20%	...	*DeplRateDevelopment*
31	capex, real		40,000,000	0	0	...	*CapexDevelopmentReal*
32	NBV bf		0	32,960,000	26,368,000	...	*NBVDevelopmentBf*
33	capex, money		41,200,000	0	0	...	*CapexDevelopmentMoney* → goes to Cash Flow
34	depreciation		8,240,000	6,592,000	5,273,600	...	*DepreciationDevelopment* → goes to Profit and Loss
35	NBV cf	0	32,960,000	26,368,000	21,094,400	...	*NBVDevelopmentCf* → goes to Balance Sheet
36							

Workings results feed through to the outputs sheets

We should find that each line of output contains a single formula which links back to the workings. The benefit is that revenue on the cash flow is the same as revenue on the P&L because they both link to the original revenue as calculated on the workings.

I would recommend that each output sheet should contain summary calculations, that is, wherever the output heading is Total, Subtotal, or Net, there should be a simple sum or addition (or whatever) of the relevant output rows. This serves to make each output sheet internally consistent; the numbers always add up. The imbalance check on the balance sheet,

for example, should be based on the balance sheet itself and not on a suspense account tucked away on the workings sheet!

	A	B	C	D	E
1	**BALANCE SHEET**				
2					
3	Financial year ending			2004	2005
4					
5	Fixed assets			33.0)	=NBVTotalCf
6					
7	Current assets				
8		cash		28.8)	=CashCf
9					
10	Current liabilities				
11		overdraft		-	=OverdraftCf
12		dividends payable		(1.8)	=DividendsCf
13		tax payable		-	=TaxPayableCf
14		total		(1.8)	=E11+E12+E13
15					
16	Deferred liabilities				
17		debt		(50.0)	=DebtCf
18					
19	Net assets			10.0)	=E5+E8+E14+E17
20					
21	Equity				
22		invested		20.0)	=EquityCf
23		retained earnings		(10.0)	=RetainedEarningsCf
24		total		10.0)	=E22+E23
25					
26	Shareholders' funds			10.0)	=E24
27					
28	Imbalance			0	=E19-E26
29					

Outputs sheets contain links to workings, and summary calculations

Audit

In setting up such summary calculations it is worth recognising that similar calculations may already exist in the workings. We might have an operating cash flow line as a summary calculation on the Cash Flow report, as well as an operating cash flow formula in

the workings. This apparent duplication has a benefit, because, as the accountants teach us, if we can calculate the same thing using two different methods and the results agree, we can have confidence in our work. In Chapter 2, we will look at the idea of setting up an audit sheet in which we specifically check output results with workings for this purpose.

Variations

A reality check

As I teach this set of ideas concerning model structure on my courses, I normally find that at this stage I will have one or two individuals simmering with indignation at this ivory tower exposition of how models should be built, in contrast to the realities of model building in real organisations. But this is not ivory tower, and there are many equally valid approaches to model structure. The key point is that we have a standard sequence of inputs – workings – outputs. If we change input *I*, we should see a change in output *O*. There is a linear path between the two, most of which is on the workings sheet. This basic methodology, once understood, becomes a very flexible basis for different modelling situations.

Earlier in this chapter we considered five different models:

1 Model A is used to calculate the net present value and internal rate of return of a manufacturing project, to be used by the company's management.
2 Model B is a loan calculator, which will be used and re-used by a number of colleagues.
3 Model C produces consolidated monthly accounts using information from several business units, to be reported to management.
4 Model D is a timesheeting system.
5 Model E is a budgeting model which will be used over a period of time, and will require the actual figures to be compared with the budgeted figures as they become available.

I would suggest that model A best illustrates the straightforward inputs–workings–outputs structure described above. The management would want to flex the inputs sheet for the sensitivity analysis, and the results are set out on the outputs. Model B, the loan calculator, introduces the first variation – we would probably want to show the inputs and outputs on the same sheet. We would need to ensure that the outputs could not be overwritten, and I would still recommend that we have a separate workings sheet.

Model C is also a variation, this time perhaps with multiple inputs sheets, one for each business unit. These could be copied and pasted from the source spreadsheets, or we could link the files. In general terms it is best to avoid file links in models (explained below), but in this case it might be the only realistic solution. The workings sheet is then used to consolidate this information before feeding it into the calculations, indeed it might be possible to have individual workings sheets for each business unit input sheet, with a consolidation workings sheet to pull it all together. The output sheets, in our methodology, are based on the workings and as such need little attention.

The fourth model, the timesheeting system, is perhaps so simple it can be set out on one sheet. The constants would be the employee name and number, the dates, the hourly rate, and the overtime rate, and the only real variable is the number of hours worked. Once the

information has been input, the model (if we can call it that) is printed and submitted to management. As long as we can differentiate inputs from workings and outputs, the structure holds.

Model E, the budgeting model, is an interesting mixture of historic data and forecast assumptions or estimates, with the added factor of the actuals which will be entered during the year. The number-crunching is probably quite simple, with some year-to-date summary formulae and perhaps some extrapolations of budget under/overspend. Reporting should be straightforward, but the model will need to be set up so that the user is clear about what can and cannot be changed. A single, well laid out sheet might be an option, but it may be more appropriate to have a historic (previous years' figures) inputs sheet, a forecast assumptions inputs sheet, an actuals inputs sheet, and then the workings and the outputs. The historic sheet is not going to change, the forecast sheet might be manipulated in response to the information made available during the year, and the actuals sheet itself is for the ongoing data entry. The outputs sheets would need to display all three input elements for comparison, along with any variance and extrapolation results.

Other variations are permissible. Model A, the NPV calculation, may contain a substantial amount of operating data relating to the manufacturing project, but in terms of sensitivity analysis the management might be interested in flexing just one or two key variables. In this context we might consider two inputs sheets – one for the key drivers and the other for the bulk of the unchanging assumptions. Or in the example of company valuation modelling, we would have historic data about the company and assumptions that relate to the forecast period. The historic data is not going to change but the forecast assumptions will be tested, so put them on separate sheets (one tip relating to historic data is to make sure that totals and subtotals are calculated and not simply typed in – this will flag up any errors due to rounding when the historic data was compiled).

Returning to Model C, the consolidation model, I mentioned that I am not particularly keen on linking to other files. The premise here is that each model should be self-contained, and file links are a notorious cause of error – particularly if the linked file is no longer in the location specified by the path in the link formulae – the classic example being when the model and its files are sent as email attachments. These are usually stored in the recipient's Temp directory and the links fail. Remember, if Excel is unable to refresh the links to the precedent files, your model is essentially broken. If file links are to be used, make sure they are isolated from the rest of the model. To manage file links, such as in model C, I suggest that we set up workbook-specific link sheets containing the relevant links, such that each link sheet contains links to only one external workbook. In other models it may just be appropriate to have a single links sheet, which contains all the file link formulae. These values are then fed into the model as required.

Documentation

Key information

Whether we are top-down or bottom-up modellers, once we get started on building a model the temptation is to crack on and get the job done. It is important to make sure that you document your work from the outset. I would recommend a separate documentation sheet, although you could include this information on the audit sheet as described in

Chapter 2. Information we should include would be:

1 Model name.
2 Version number.
3 Filename – use the CELL function to return the full path of the workbook:=CELL ("filename").
4 Date – do not be tempted to use =NOW() functionality as this will only tell you the current date, not the date when the model was last used. The date must be hard-coded (try Ctrl+; as a date shortcut). Put the current date and time in the page footer.
5 Model author – your details, including telephone number and email address, in case anyone needs to get hold of you concerning the model.
6 Model sponsor – on whose behalf are you building the model? Contact details.
7 Model owner – if questions arise, who is empowered to make decisions concerning accounting policies? Who has ultimate responsibility for the verification of the inputs? Contact details.
8 Work completed – a record of main features/elements completed. With dates.
9 Work to follow – a list of priorities for future tasks. With deadlines.
10 Amendments – work previously completed but then revised. With dates.
11 Iteration cycle – how many times has the model been subjected to review/audit by the model sponsor/owner?
12 Audit – what is the overall audit status of the model (see Chapter 2).
13 Instructions to users – description of model purpose, locations of key inputs, key outputs. Description of macros. Instructions for printing.
14 Navigation – Go To macro buttons or hyperlinks.

	A	B	C	D	E	F	G	H	I	J
1										
2										
3		Model name		**NewCo FSA Regulatory Financials Forecast**						
4		Version number		0.5						
5		Filename		C:\FSA\[NewCo FSA Regulatory Financials Forecast.xls]Documentation						
6		Date		01/06/2004						
7		Deadline		30/06/2004						
8		Author		JS, PM to review						
9		Sponsor		N/A						
10		Owner		Finance Director, finance.director@newco.co.uk						
11		Work completed		Model structure						
12				Agreement with FSA financial requirements template						
13										
14		Work to follow		Confirmation of applicable FSA ratios						
15										
16		Amendments		Nil						
17										
18		Iteration cycle		1						
19										
20		Audit status		FALSE						
21										
22		Instructions		Nil						
23										
24		Navigation		Key Inputs		Workings				
25										
26				FSA Report		Financial statments				
27										
28										

Basic workbook documentation, with navigation buttons

File Properties

In my experience the File, Properties dialog box is little used. It is a generic Microsoft Office feature and the five tabs in the dialog box store information which may not be apparent in the workbook itself, or difficult to update manually.

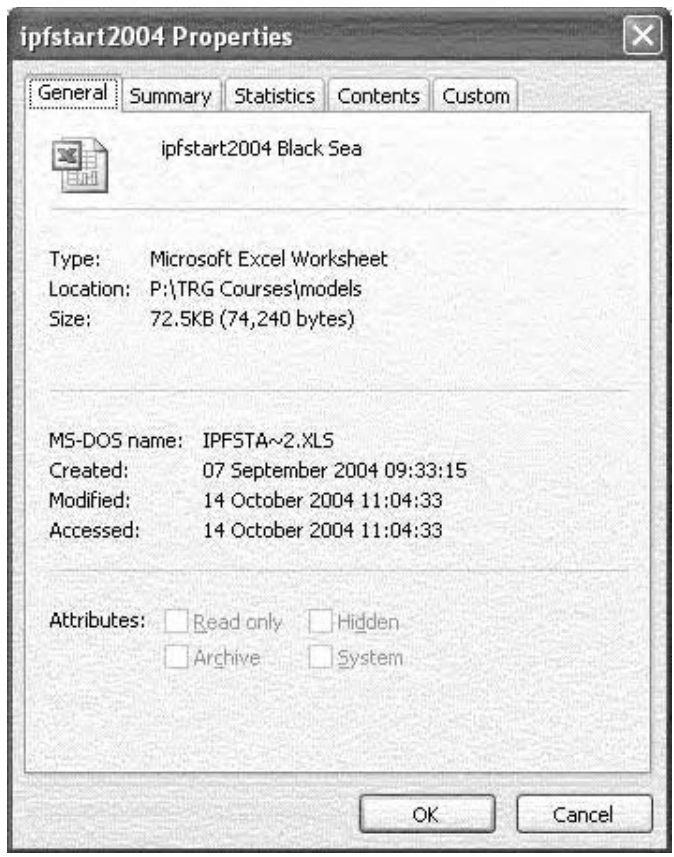
Under Tools, Options, General, Prompt for workbook properties, you can ensure that the Properties dialog box is displayed automatically when a workbook is saved for the first time.

General

This tab provides information about the file name, path, and size, and about when the file was created, modified and accessed.

Summary

This contains information relating to ownership. As far as I can tell, the default Author and Company are derived from the details provided when Excel/Office was installed, but these fields can be edited.



The File, Properties dialog box

Statistics

This duplicates the General tab but also stores information about the use of the workbook. The non-editable field for 'Last saved by' is linked to the name shown under Tools, Options, General, User name.

Contents

This lists all sheets, charts, and range names used in the file. These are all derived from the workbook and cannot be edited.

Custom

Additional headings are provided in which further information can be stored. There is a list of predefined properties or you can define your own.

Reporting

I have suggested that by using the top-down approach to modelling we focus on the outputs first – the tangible product of the largely intangible process. The outputs, or reports, are designed in consultation with the users so that the reports are of genuine value. And we ensure that the outputs are designed from the outset, so that at any stage of the model development we can demonstrate the progress made. We should therefore make sure that we can print our work at a moment's notice. When it comes to reporting some modellers rely on macros, which in some cases is the only way to handle complex reports. But it is helpful to understand basic printing techniques.

Print area

Before printing, go to each sheet in turn and press Ctrl+End. This will select the bottom right hand cell in the worksheet. It should correspond to the intersection of the last column with anything in it, with the last row that contains anything. To fix this:

1 Select the empty columns or rows
2 Delete them: Edit, Delete or Ctrl+−.
3 Save the file, close it and re-open it.
4 Run the Ctrl+End shortcut again to confirm that the end cell is where it should be.

It is worth noting that Ctrl+End relates only to cells in the worksheet. If you have objects such as macro buttons, combo boxes, or embedded graphs, Excel does tend to recognise them and increases the print area accordingly, which you may or may not want, in which case you will have to define the print area manually.

The print area is usually defined automatically by Excel. To specify your own print area you can select the range manually and use File, Print Area, Set Print Area. Range names can speed up this selection process if you have set them up in advance. It would seem that we are restricted to one print area at a time, but we can work around this using non-contiguous selection: Ctrl+click and drag over non-adjacent ranges. Using print preview to see what this does, we find that Excel puts each range on a separate sheet. I would welcome a workaround for this but I have never found a non-macro solution.

We can also specify print areas using the View, Page Break Preview command which has appeared in recent versions of Excel. With this technique, we can define the ranges for printing and view them in isolation from the other information in the workbook. We can redefine the boundaries of the areas by clicking and dragging, and ranges can be added or removed by right-clicking and making the appropriate choices from the short-cut menu.

Whichever technique we use, we are confronted by Excel's inability to assemble print areas from different sheets unless we are content that each print area should be printed on a different page. We could record a simple macro in which the source print areas are copied and pasted (as values perhaps) to a separate sheet and assembled for printing, this sheet then being discarded or retained on completion of the subsequent print command.

Group mode

If you have used the inputs–workings–outputs structure described previously, it is likely that we only need to print the outputs. These should all have the same page layout to give a consistent look-and-feel to the reports. You will know that you can group all your output sheets by Ctrl+ or Shift+ clicking on the sheet tabs, which then sets up group mode. An alternative is to right-click a sheet tab and choose Select All Sheets.

Page setup

Page setup can catch out the unwary, because there are two ways of using it. If you have grouped the sheets together and you use the File, Page Setup command, the settings you specify will apply to all sheets in the group. If, however, you run File, Print Preview and run the Setup command from there, the settings only apply to the current sheet, whether or not it is one of a group. This little-known wrinkle accounts for reams of wasted paper where analysts are convinced that they have the correct settings and yet the changes stubbornly refuse to appear.

Fit to page

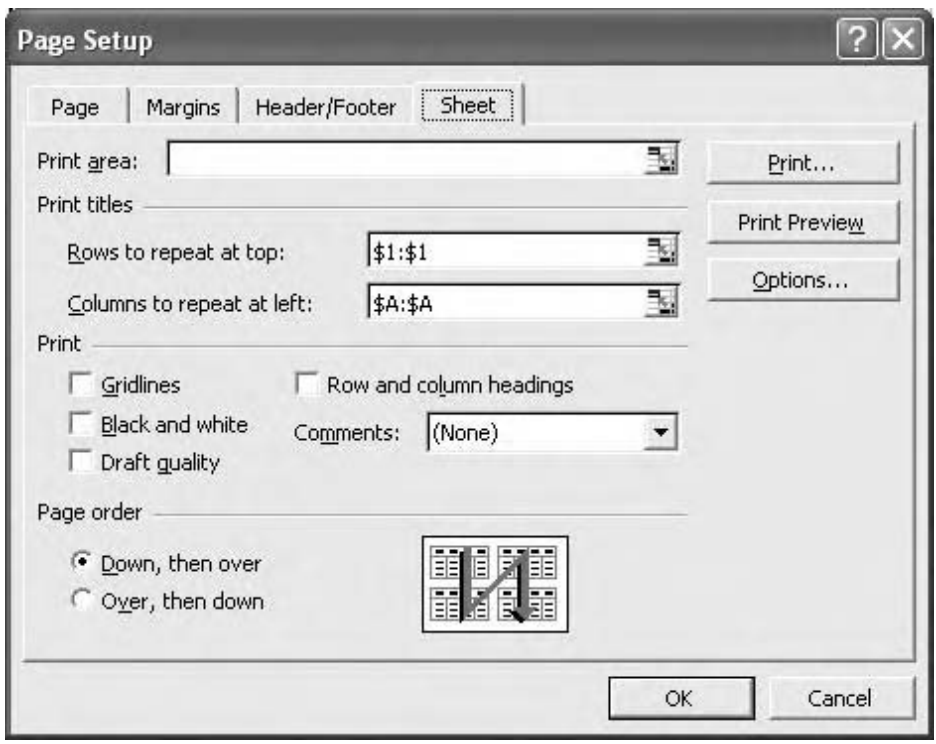
If a standard layout has been adopted for the report sheets, we will find that we have the same number of columns on each sheet so the width is constant. However, each sheet is likely to differ in its length. Page Setup offers the scaling options to increase or decrease by a percentage value, or to fit to a specified number of pages. With the latter, it is worth noting that we need only specify one value; for example, I have adjusted the column widths and layout with the intention of printing on a landscape page, but we do not know the

length. Rather than specifying 1 page wide by 1 tall, we simply enter the width requirement and leave the height blank. The effect of Fit to Page on font size must be appreciated if you are working with a house style.

Headers and footers

Headers and footers should be classed as part of the model's documentation and treated accordingly. Each page of the printout should show the sheet name and page number, and at least one page should have the filename. It should be compulsory to include both the date and the time on the printout – we have seen models approaching financial close that are being adjusted and updated and subsequently printed on an hour-by-hour basis. You must be able to confirm that the printouts on your desk are the most up-to-date available. The footer should also contain such caveats as 'Numbers may not agree due to rounding', Numbers in £000s', or 'Numbers in millions unless stated otherwise' if applicable.

It is not widely known, but headers and footers can be used as a form of security stamp. For some reason Excel "remembers" all custom headers and footers created in a workbook. If you make sure that your details are entered in this way, even if you don't then use the header or footer for printing, they will be recorded for posterity when the file is saved.



Setting print titles

You may be aware that if we attempt to write an & character into a header or footer, for example, XYZ & Partners, Excel will omit the &. To solve this, use XYZ && Partners.

One of the failings of Excel is that it has never had the ability to put a link to a cell in the header or footer, because there are times when we need to include more information than can be entered using the header/footer codes. Back in the worksheet we can use functions such as = CELL("filename") which will return the full path of the spreadsheet, rather than the simple filename returned by the &[file] code used in the header/footer dialog box. We can also use Print Titles to include information from the worksheet, so we could write the CELL function into a cell outside the print area, select the print area, and set the print titles as being the single column and row that intersect at this cell reference.

This can be extended to include multiple column/row intersections, but we cannot include non-adjacent rows or columns. A limitation to print titles is that they are worksheet-specific and cannot be applied to multiple worksheets, although we can set them up on each worksheet individually.

I would not recommend putting the version number of the model into the header or footer because it often gets overlooked when updating a file. The version number should appear on the model's documentation sheet as part of the normal printout.

Reports

We should be able to generate reports at short notice. Once the page setup routines have been followed, printing reports can be as simple as Ctrl or Shift clicking the sheet tabs to group them, and then running the print command (Ctrl+P). If using this method remember to switch off group mode afterwards – click on a sheet tab outside the group, or right-click a sheet tab and choose Ungroup Sheets from the shortcut menu.

Report manager

There is an Excel add-in called the Report Manager which can be used to set up reports. As this book is concerned with in-built Excel functionality it will not be considered further, but you may well wish to explore it yourself.

Custom views

Another way to set up reports is to use custom views. The time spent setting these up is recouped from the ease of use in generating reports later on. It is similar to the process of setting up range names (see Chapter 3) but it includes the print settings. This technique is particularly useful if individual elements of the reports need to be printed, rather than a job lot which may have standard print settings.

1 Clear any existing print area using File, Print Area, Clear Print Area.
2 Select the first range for printing.
3 Use File, Print Area, Set Print Area.
4 Use File, Page Setup to specify the settings required, headers and footers, etc.

5 Use View, Custom Views, Add, and give the view a name.
6 Click OK.

Repeat this for each additional range to include, making sure that the existing print area is cleared each time. This technique is particularly useful if individual elements of the reports need to be printed, rather than a job lot which may have standard print settings. When you need to print, use View, Custom Views, Show and print the selection.

Model development

Let's go!

Drawing on the ideas set out above, and anticipating further suggestions in the following chapters, it is helpful to consider the first steps in setting up a model. Above all else, make sure that the location of the inputs is specified, and that the key results are set out clearly. The amount of effort you put in at this stage should be commensurate with the ultimate purpose of the exercise – the quick and dirty monthly figures spreadsheet does not really require us to labour over the finer points of detail and methodology, whereas the model supporting the business case for opening a new office will benefit from the extra effort spent setting up a clear and robust structure from the outset.

New file

Create a new workbook in Excel, add new sheets as required by right-clicking a sheet tab, choosing Insert, and Worksheet, or pressing Shift+F11. Name the sheets. You will know that you can click and drag the sheet tabs to rearrange them, and you can copy a sheet by Ctrl+click and drag its tab. Decide if an audit sheet is necessary (Chapter 2).

Saving

At this stage, save your work. With a simple model we probably do not need to give too much thought to this, but with a more complex model that will be developed over a period of time it makes sense to consider the grandparent/parent/child system: the first draft of the model is the grandparent; following the next set of modelling activities the workbook is saved with a new name number as the parent; and after the next development saved again with a new name or number as the child. As work progresses, the grandparent is deleted, the parent promoted to grandparent, and the child to parent, with the current work as the child. In this way you will have a rolling series of files showing work-in-progress but with at least two back-ups. Alternatively, if disk space is not a problem, we can use an incremental file saving system, with each version of the model carrying a specific version number (which I would like to see documented in the workbook, for audit purposes: see Chapter 2).

Learn the shortcuts Ctrl+S for File, Save, and F12 for File, Save As. I would recommend that, if you have it, the AutoSave add-in should be disabled (Tools, Autosave – if you cannot see this command, it has already been disabled). You should control the back-up process, not Excel.

Grouping

The general modelling rule is that each sheet should have the same layout ('look and feel'). To apply the same settings to each sheet, group them together by right-clicking on any sheet tab and choosing Select All Sheets. To select groups of sheets, select the first sheet tab, hold down the Shift key, and click on the last sheet tab. To select non-adjacent sheets, hold down Ctrl whilst clicking on the required sheet tabs. Caution: any editing or formatting carried out with group mode on will affect all grouped sheets. You can check the grouping status by looking at the sheet tabs or by reading the Excel title bar – it will have the warning [Group] after the file name. To disable group mode, click on a sheet tab outside the group, or right-click a sheet tab and choose Ungroup Sheets. Note that you will find that some commands appear to be disabled while Group mode is active.

Layout

With group mode enabled, adjust the column widths and enter the column headings that are common to all sheets. Apply any standard borders. You may want to consider providing a 'hard edge' to the model, by hiding all the columns beyond the end of the forecast period or equivalent. Put the active cell in the column which you would like to be the right-hand edge of the spreadsheet. Press Ctrl+Shift+right arrow, followed by Ctrl+Spacebar. These two actions select the surplus columns. Press Ctrl+) or run the Format, Column, Hide command, to hide these columns.

To restore the hidden columns, press Ctrl+A and press Ctrl+Shift+) (or Format, Column, Unhide).

Units and base columns

Identify and label any units columns, and set up a base column if required (see Chapter 3). Apply a fill colour to both.

Number formatting

You may wish to apply number formatting at this stage. I would agree that comma (thousands) format is useful, but I would not recommend any of the custom millions-type formats at this stage (see Chapter 5). Note that if you apply the thousands format, values such as percentages will appear as zeros until formatted appropriately. If we have numbers in the millions we can enter them more efficiently using the exponential format. For example, we can write 10,000,000 as 10E6. Excel displays this as 1.00E+07, and the thousands format makes this more readable (people simply do not like the exponential format). Learn the shortcuts for the common number formats:

Ctrl+Shift+% for percentage, two decimals.
Ctrl+Shift+! for thousands, two decimals.
Ctrl+Shift+$ for your default currency format.
Ctrl+Shift+~ applies the default general number format (i.e. it clears any existing number formatting).

There is no shortcut for changing the number of decimal places.

AutoComplete

Disable the AutoComplete feature. This analyses your text entry whilst you type and if it corresponds to a previous entry in the same column Excel will suggest how to complete your typing for you. I find this immensely irritating and I switch it off using Tools, Options, Edit, and clearing the Enable AutoComplete for cell values check box. There are occasions when it would be helpful to copy from the cell above and in this case I use the little-known shortcut Shift+"

Printing

Now prepare the model for printing, as mentioned in the previous section. We should be able to print our work at a moment's notice. At the very least we should get the page layout setup, with appropriate headers and footers. Run a test print, and show the results to colleagues or the project sponsor to ensure that your layout matches expectations.

Data entry

For quick data entry, select the range to contain the information (row or column) and type in the figures or text. Press Enter to move the active cell down after each entry, or Tab to move the active cell to the right. Hold down Shift with either Enter or Tab to move in the reverse direction. While the range remains selected Excel will cycle you through each cell within that range in turn.

- To fill a range with the same information, select the range first, type the entry, and press Ctrl+Enter.
- To select a range of cells which already contain data or formulae, press Ctrl+*.
- To select cells in a particular direction, press Ctrl+Shift+arrow key.
- To select a row, press Shift+Spacebar.
- To select a column, press Ctrl+Spacebar.
- To select the whole sheet, press Ctrl+A.

Cell comments

As you start work on the model, you may find it helpful to document your work as you go using cell comments. You can easily set these up by using the Insert, Comment command, or the shortcut Shift+F2. To remove cell comments, use Edit, Clear, Comments.

You can change the name which appears by default in the comment by using Tools, Options, General, User name.

Fill Right

As you start writing formulae, you will almost invariably need to copy them across the row. Using the mouse to copy across can be frustrating, particularly when the mouse pointer reaches the right-hand edge of the screen and the spreadsheet starts scrolling rapidly. A more effective technique is to write the formula, select the cells in the row using Shift+right arrow, and then press Ctrl+R, which is the shortcut for Edit, Fill, Right.

Sign conventions

At this stage we should also give thought to our sign conventions – do we show liabilities as negatives or positives? As far as the outputs sheets are concerned, this is determined by reporting conventions which seem to differ from one organisation to the next. However, for workings or calculations purposes, there is a good reason to calculate all liabilities as positives (liabilities here being used in a broad sense, covering all charges, for example, costs, depreciation, interest payable, debt repayments, tax charges, and so on). The reason for showing them as positive numbers is that in our accounting and finance training we tend to learn textbook formulae which express, for example, profits (or earnings) before tax as being revenue *less* costs, *less* depreciation, *less* interest. If costs, depreciation, or interest were calculated as negatives, we would have to add them back, which just would not look right. And worse, an inconsistent approach might result, for example, in positive costs and interest, but negative depreciation. This formula would be messy, and it would take additional time to then check the precedent values to ensure the signs were correct.

Do not confuse the observations here about sign convention on liabilities with the sign seen in cash flows: we may well expect to see a cash account or a company's reserves increasing or decreasing over time, using the appropriate sign to show the direction of the movement.

If we had a reporting convention of showing liabilities as negatives, we can change the sign using =0-, as in

```
=0-InterestPaid
```

This might look a little fussy, but on UK/US keyboards the – and the = keys are adjacent to each other and we see this as a fairly common typing mistake, so I would not recommend simply typing =-InterestPaid, because we could not be sure if this was an error or not. This =0- method of changing sign is also easier to spot.

Navigation

As we start the model build it is important that we can navigate our way around. A sensible layout with appropriate documentation will greatly help, but when checking formulae and references we need to be able to move quickly and efficiently around the workbook. I would expect any competent modeller to be familiar with the following techniques.

Ctrl+PgUp/Ctrl+PgDn

These basic shortcuts are used to move from one sheet to the next.

Alt+PgUp/Alt+PgDn

Not so widely used as shortcuts, but serve to move a screen to the left or screen right.

The scroll tabs and the sheet navigation menu

Right-click scroll tab buttons

Right-clicking anywhere on the scroll tab buttons at the bottom left of the workbook will bring up a shortcut menu which lists all the worksheets and can be used to navigate rapidly across multiple sheets.

Home/Ctrl+Home

The Home key will move the active cell to column A, or to the left hand edge of the worksheet if Window, Freeze Panes has been used. Ctrl+Home takes the active cell to A1, or to the top left cell if the panes have been frozen.

Ctrl+End

This keystroke combination will move the active cell to the bottom-right hand cell of the active sheet. This should correspond to the intersection of the last column and last row of the active area. Sometimes we find that this cell is way beyond the expected end of the sheet, which normally indicates that at some stage we copied something too far across a row (or down a column), and subsequently deleted the material.

The solution to this is to:

1 Select the appropriate rows and/or columns.
2 Delete them.
3 Save the file.

4 Close the file.
5 Re-open the file.

On pressing Ctrl+End again, the active cell should now be in its correct position.

Ctrl+arrow

Press Ctrl+an arrow key to move to the edge of the spreadsheet in that direction. Alternatively press End and the word End appears in the status bar; press any of the arrow keys to be moved to the last cell in that direction. If the row or column is empty, or the active cell is already beyond the last cell, you will find yourself at the outer edges of the spreadsheet. This can be irritating if you frequently need to read across a row, so one workaround is to give the worksheet a 'hard edge'. If, for example, column T marks the right hand edge of your workings area, fill the column with a number or a character. Now when you use Ctrl (or End) + arrow, you should find that the active cell will always stop in column T. A sneaky variation on this is to use the ' apostrophe character – when this is entered in a cell, the cell appears blank, with the (non-printing)' only visible in the formula bar.

F6

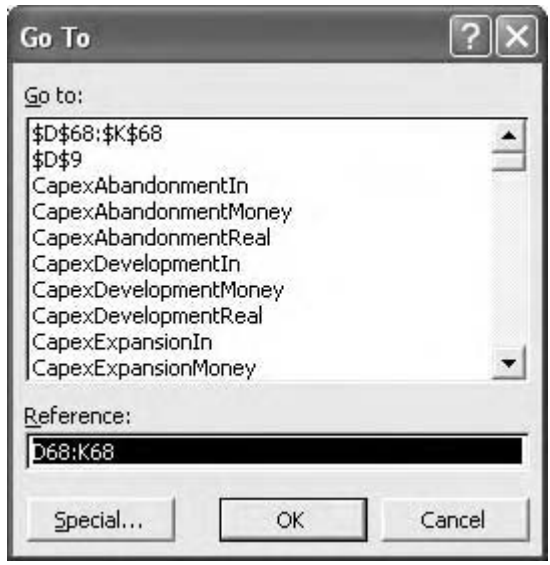
If the window has been split, use F6 to move the active cell from one pane to the next. Use Shift+F6 to reverse.

Go To

You can use either F5 or Ctrl+G to call up the Edit, Go To dialog box. Enter either the cell reference or range name and choose OK (or press Enter). One interesting feature of Go To is

	A	B	C	D	E	F	G
1							
2	Financial year ending			2004	2005	2006	2007
3							
4	Inflation						
5		rate		3%	3%	3%	3%
6		index	1	1.03	1.06	1.09	1.13
7							
8	Price						
9		real terms (2004)		18	18	18	18
10		money terms		18.54	19.10	19.67	20.26
66							
67	Project cash flow						
68		money terms		-41,200,000	2,015,710	2,076,181	25,717,876
69		real terms		-40,000,000	1,900,000	1,900,000	22,850,000

The split window

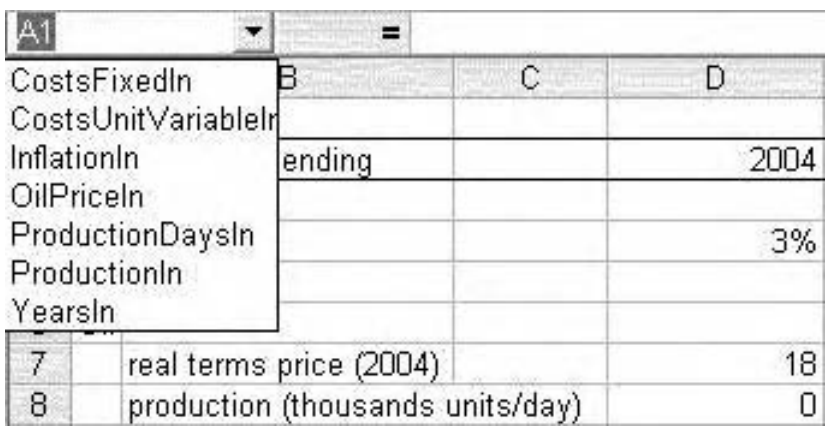


Go to, and Go back

that if you then press F5 again, Excel enters your previous location as the default, as a Go Back function. If I used F5 to move the active cell from E36 to E73, when I press F5 again the destination cell is assumed to be E36. Excel in fact stores up to the last four locations of the active cell, which allows for some useful auditing. Although the dialog box lists all your range names (if you have any), Excel will list the cell references of the ranges in the dialog box.

Name Box

The box on the left-hand edge of the formula bar is the Name Box. It normally shows the cell reference of the active cell, but you can click in the box and type a cell reference



The Name Box

for a Go To functionality. You can also select a range name from the list with the same result.

Note that in this context it is not possible to widen the Name Box, so longer names can be difficult if not impossible to read. We will look at names in more detail in Chapter 3.

Ctrl+[

You will of course be familiar with F2 Edit Cell, and that when used on a formula Excel highlights all precedent cells with different colours (assuming Tools, Options, Edit, Edit Directly in Cell has been activated). With many formulae, however, the references are off-screen and you will need to browse to find them, and in so doing run the risk of losing the active cell. One powerful shortcut is Ctrl+[(open square bracket), which is Select Precedents. This in itself is a shortcut for F5 Edit, Go To, Special, Precedents. The trick with Select Precedents is that the precedent cells are highlighted, and you do not need to scroll around to find the selected cells but instead press Enter repeatedly. You will know that when you have several cells selected the active cell will scroll around within the selection when you press Enter (or Tab). I will often take this a step further and while the precedent cells are still selected, apply a Fill Color. In this way, for example, all the precedent cells for my EBITDA* calculation can be coloured light blue, and I can examine the audit trail at leisure, or even leave it as a marker for other users to follow.

The benefit of Ctrl+[is that you can inspect both the values and the formulae in the precedent cells. Pressing Ctrl+[repeatedly will push you further up the precedent pathway.

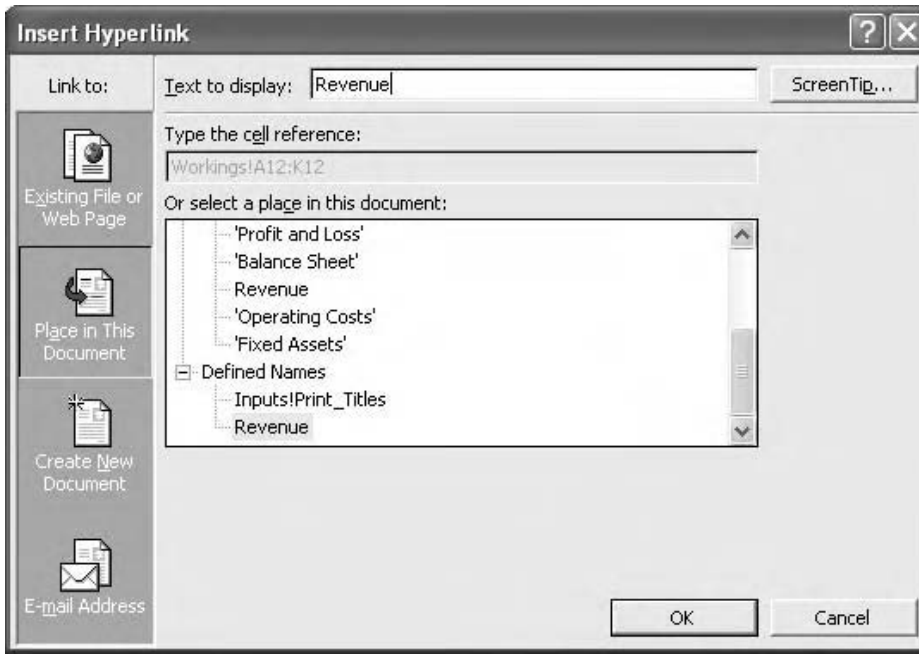
I am sure you can predict that Ctrl+] (close square bracket) will Select Dependents. One thing to note with these two techniques is that they are formula-driven, by which I mean that there must be a formula on the same sheet to push or pull the selection. If you have an inputs sheet, for example, which contains only values, Ctrl+] does not work. Also, if a formula feeds into one formula on the same sheet and another on another sheet, the shortcut will only pick up the dependent on the same sheet. However, with the workings sheet methodology introduced earlier in this chapter, this is not such a problem.

Hyperlinks

Hyperlinks do not seem to be used much in modelling. Used carefully they are a very efficient way of navigating to very specific locations in the workbook.

1 Select the cell to contain the hyperlink and use Insert, Hyperlink (or Ctrl+K).
2 Click on the Place in this Document tab.
3 Type the text to display in the cell (and if you want to, a ScreenTip)
4 You can then either enter a cell reference or a range name for the destination cell(s).
5 Click OK to return to the workbook.

*Earnings Before Interest, Tax, Depreciation, and Amortisation



Inserting hyperlinks

Test the hyperlink by clicking on it. Note that unlike F5 Go To, there is no corresponding 'go back' functionality, unless you then set up further hyperlinks. Hyperlinks can be edited by selecting the cell using the keyboard and pressing Ctrl+K – do not try to click on the cell otherwise you will activate the link!

Excel also has a HYPERLINK function which has the following syntax:

```
=HYPERLINK("[model.xls]'Balance Sheet'!A1","Balance Sheet")
```

When the user clicks on this example the active cell will move to the home cell position on the balance sheet. Note that the filename must be included, and the entire filename+sheet-name+cell reference string is enclosed in quotes. The second part of the formula is the prompt or text which appears in the cell as a hyperlink.

Warning: neither method will update the link references if the sheet or file names change, and as such are very prone to error.

CHAPTER 2

Quality control

Introduction

The previous chapter set out some suggestions concerning model layout and structure. If accepted and implemented, these suggestions take the form of rules, and when using a rule-based methodology we should always be able to check and confirm that such rules are being followed. With the principle of error reduction we want to anticipate potential errors and to identify them as early as possible. This chapter sets out the requirement for us to incorporate quality control into the model development process. If left too late, minor errors have a habit of compounding themselves and they become very difficult to untangle.

If you are reading this book from start to finish, this chapter appears to sit uneasily in the sequence: the previous chapter looked at getting started and setting out the structure of the model, and the following chapters look at the formulae and functions and model use. But here we are concerned about the various checks and tests that we will be applying as we start the number crunching, and we need to appreciate that in some of this we will be a little ahead of ourselves, and the significance of a specific test (e.g. the iteration status) will not become apparent until we cover the related topic in detail later in the book.

The demonstration workbooks for this chapter are in the Chapter 2: Quality Control folder on the CD-ROM.

Taxonomies of error

There are several taxonomies of the types of error which may be found in spreadsheets and financial models.* From my own experience I would suggest that we should be aware of the following categories of error.

Pointing errors

By far and away the commonest error in all the research exercises and seen in general practice is the discrepancy in cell reference. There is a simple rule – never type a cell reference.

*Please refer to Professor Ray Panko's website http://panko.cba.hawaii.edu/ssr/ for the research relating to spreadsheet errors.

To put a reference into a formula, either click on the cell with the mouse, or better, use the arrow keys to select the cell (the keyboard is slower – pointing errors are normally caused when working at speed). Alternatively, consider the use of range names instead of cell references (Chapter 3). It is quite difficult, although not impossible, to put the wrong name into a formula, and it is much easier to spot such errors if they have occurred.

Input errors

You know its 5,000, I know its 5,000, but somehow it ended up in the model as 5 and the numbers are a factor of a thousand out somewhere. Make sure that the numbers on the inputs sheet are expressed in the same units as in the project or transaction documentation. A useful test for this is to make sure that one of your output sheets is a summary of your inputs sheet using references to the actual values used in the dependent calculations. In this example a calculation in the workings multiplied the 5 by 1000, with the outcome that when the input was finally corrected and the 5,000 entered in full, the outputs displayed 5,000,000.

Remember the old expression, Garbage In, Garbage Out.

Omission errors

Naturally the most difficult type of error to spot, and the one that most frequently is noted after the event is an omission error. If the analyst has not been told to include a particular set of costs, for example, who is responsible for the error? This is best avoided by adopting the top-down approach to model building, outlined in the previous chapter. By preparing the model outputs first, and by engaging in an iterative process with the model sponsors, many such omission errors are trapped at the very outset of the modelling process.

Commission errors

These occur when the analyst does something that they are not required to do, such as adding extra detail or including calculations that are not needed but which affect the model's results. An example is that of a part-qualified accountant who amended a tax calculation on the basis that 'this is how it should be done', rather than how the company actually did it. Again, the top-down approach to model design helps prevent this: agreeing the model inputs and outputs in advance reduces the scope for creative modelling.

Alteration errors

A variation of the commission error, where perhaps a flaw or mistake has been identified and corrected by the modeller, but the model sponsors were not told of the update. All such amendments should be logged on the audit sheet.

Calculation errors

A general description for the largest category of error, and many such errors can be hard to track down. A particular problem is the long formula, containing several sets of brackets, lots of sheet names, and wrapping around the formula bar. Not only do such formulae look unpleasant, but research shows that they are rarely checked or tested in detail, and quite often even the analyst who created them is unsure of their exact function. One of the

themes of the formulae and functions chapters is to break formulae into short sections over several cells, which may appear less intellectually rigorous but is far more reliable in the long term. A criticism of the short formula argument is that if we have more formula cells then statistically there is a greater risk of error, but the point is that the formulae are shorter and simpler and inherently less error prone.

Timing errors

Calculations involving the timing of an event, or the duration of a process such as depreciation, are often some of the most difficult to set up reliably. Workarounds include such dodges as simply writing the formula in a fixed number of cells, such that if the asset life, for example, is changed, the calculations do not reflect this unless they are copied to the new cells. This category also includes errors such as using the annual interest rate when calculating interest on a quarterly basis. Chapters 3 and 4 set out a number of reliable techniques which address this type of problem.

Domain errors

Domain knowledge refers to your professional knowledge – finance, accounting, management, or whatever. The researchers reassuringly note that this is the least common type of error – we all know that cash is not profit, that we should depreciate our assets, pay tax, use real discount rates on real cash flows, even if we are not sure how to perform the calculations.

Audit tools

Excel contains a surprising amount of audit functionality without recourse to expensive third party auditing tools. The latest versions of Excel contain some fairly sophisticated audit features, some of which automate the manual procedures described in this section. As with any diagnostic procedure, it is important to be able to understand the significance of the test, and the meaning and implications of the result.

Be aware that some features of the auditing toolbar and several of the audit shortcuts may be disabled if Excel's protection, group, or objects features have been set.

F2 Edit Cell

We know about using F2 to edit the contents of a cell directly in the worksheet, rather than using the Formula Bar. If this does not seem to work, run the Tools, Options command and check that the Edit directly in cell option has been selected on the Edit tab. We can use the Home, End, and Shift and arrow key combinations to correct or amend the formula. If the wrong cell references have been entered, we can use the trick of pressing F2 again, then using the arrow keys to select the correct cells in the worksheet (Point mode, shown on the status bar). Press F2 once more to carry on editing the formula.

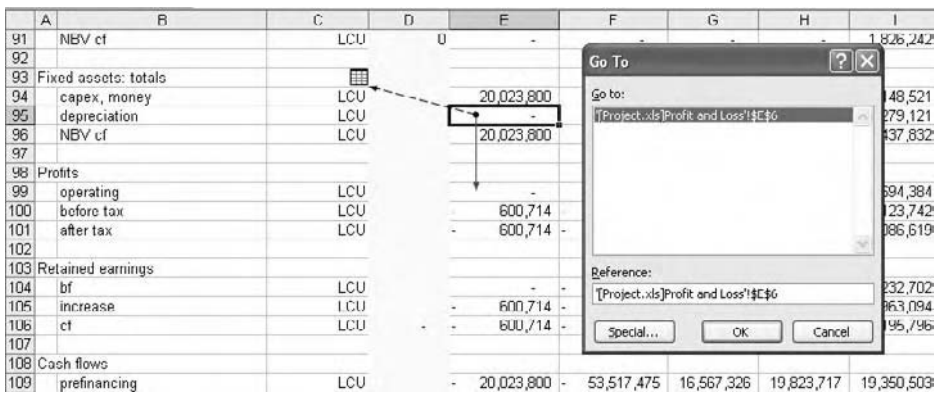
This same technique can be used in some dialog boxes which require range or cell references. Any attempt to use the arrow keys will cause the references to change, which can be annoying if you are simply trying to edit part of a cell address, for example. Press F2 to change from Point to back to Edit mode.

Remember, F2 is your basic diagnostic tool for checking Excel formulae.

Audit Toolbar

You can fire up the auditing toolbar using Tools, Auditing, Show Auditing Toolbar. It contains buttons for tracing precedents and dependents, tracing errors, and for locating invalid entries when using data validation (Chapter 5). I am not particularly keen to see blue arrows drawn all over my workbook, but the toolbar does have some uses. In this example, the revenue calculation feeds into a cash flow calculation further down the workings sheet, as well as to the revenue line on the Cash Flow report. On clicking Trace Dependents, Excel points to both. The workbook icon indicates that the dependent formula is off-sheet, and we can double-click the dashed arrow to see the Go To dialog box which lists the off-sheet references. Generally I find the Ctrl+[(Select Precedents) and Ctrl+] (Select Dependents) shortcuts more useful.

Trace Error is used for auditing cells which contain error values, which in themselves are relatively easy to audit anyway, as we shall see.



Double click on the sheet link arrow to see the Go to dialog box

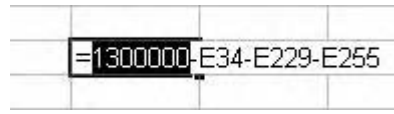
View formulas

We can click on individual cells to inspect their contents on the formula bar. Sometimes, however, we want to get a bigger picture of the type and structure of the formulae used in a workbook, and a neat trick here is to use Tools, Options, View and select the Formulas check box. Alternatively use the keyboard shortcut Ctrl+` (left apostrophe – unfortunately this does not work on most European keyboards). This doubles the width of the worksheet columns and exposes the underlying formulae. This can be very useful to get an overall impression of the workbook, and it is worth noting that this is a printable view. By hiding appropriate columns and adjusting column widths as required we can print out a hard copy of the sheet for future reference. To restore the view, repeat the command sequence/shortcut.

F9

Sometimes we want to get some information about a cell without actually wanting to see it. Press F2 to inspect the formula of interest, and click and drag/select one of the cell references.

Then press F9. The reference converts to the value in the precedent cell. Repeat for each part of the formula, until all references are converted to values. Do not press Enter otherwise the references will be permanently converted to values.



Select the cell reference and press F9

If you use this technique with range names, Excel will treat the name as an array reference and on pressing F9 it will return every value in the array. Not helpful.

The Watch window

This is a new feature in Excel and I have to admit that it is rather useful. Using the new Tools, Formula Auditing, Show Watch Window command, a small window is displayed from which we can monitor a formula and its value.

The Watch window (Excel 2003)

To use it, select a cell containing a formula and choose Add Watch in the Watch window. The Watch window works across sheets, so that the effect of changing an input value or precedent formula can be seen on the dependent formula. I have not checked the limitations of this tool, but it would appear to be robust – I did the usual trick of filling column A with RAND functions and it did not seem to object to watching the 65,536 formulae that were produced.

Error values

Error values signify Excel's inability to understand a formula or function and in themselves can be helpful diagnostic tools. Regardless of the specific layout, we tend to see that formulae cascade through the model, and on encountering an error value we simply work back to locate the first occurrence of the error. If we understand the error value it is usually fairly simple to resolve the problem. I describe some of the common error values and their causes below, but refer to Excel Help for more detailed explanations and for less common causes.

#VALUE!

This error has two common causes:

1 Referencing a cell that contains text. Select the cell with the error and press F2 or Ctrl+[. Inspect the precedent cells and correct as necessary. If the cell looks empty, the problem could be that a user has pressed the Spacebar to clear the contents of the cell, so press the Delete key to make sure.
2 Referencing a range instead of a single cell, for example, =E10+(E12:E20). In this case, the analyst missed out the required SUM. A similar problem arises with range names if the reference to the range name is outside the columns included in the name range, for example, in column D the formula reads =EBITDA, but the range EBITDA is defined as E42:K42 (see Chapter 3 for further information).

#REF!

This error is commonly found when rows or columns are deleted from the model, such that a formula refers to a cell which no longer exists. The unpleasant feature of the #REF! error is that Excel will substitute it into the formula in place of the original cell reference. If several rows or columns have been deleted it can be difficult to rebuild the formula, or to work out what the formula was originally referring to (#REF!-E66-#REF!-E117). If it is possible, undo the deletions and instead select and delete the contents of the unwanted cells (not the rows or columns themselves). Then, with the cells still selected, type some text and press Ctrl+Enter. This should fill the range with text and the dependent formulae now return the #VALUE! error. Locate these formula and make the appropriate amendments whilst the cell references are still available; or at least put in a cell comment to provide instructions on what to do with the formulae after the unwanted rows (or columns) are finally deleted. Do not simply delete the original cell contents as you may end up with formulae that refer to blank cells, a cardinal modelling error.

#NAME?

This error clearly indicates that Excel does not recognise the range name you have entered into a formula. Normally this is easy to fix: on pressing F2 the range names and references in the formula that Excel recognises should appear in colour; if a name remains black it is likely to be the unknown name, although it is possible that it could refer to a range on another sheet. Alternatively, use the F9 technique from the previous chapter: click and drag over each range name and press F9. If Excel recognises the name it displays the values from the range (usually all of them, as an array). If a name evaluates to #NAME? it has not been recognised. The corrections are equally simple. Check the spelling of the name and type the correction, or press F3 Insert, Name, Paste and select the name from the list. If neither of these work, check that the name exists in the first place by using Ctrl+F (Edit, Find), typing in the unknown name (as spelled in the formula), and seeing if Excel can find the original name. If the row is located, run the name command Insert, Name, Create (or Define) as required. Range names are explained in much more detail in Chapter 3.

#DIV/0!

This error often occurs either when data is being deleted from a model, or when formulae are being written in advance of the relevant information being provided. The denominator is missing from a division formula, so once the information or precedent calculation is provided the error will disappear. If the information is genuinely unavailable, use unit values (1) instead, as a temporary workaround. Document your work accordingly.

#NUM!

Not a common error, this usually occurs with the IRR function. Excel uses an iterative technique to calculate the IRR and if it cannot generate an answer within 20 iterations it returns the #NUM! error. This typically happens if all the values in the cash flow have the same sign.

#N/A

This is often generated by the VLOOKUP, HLOOKUP, and MATCH type functions explained in Chapter 4, usually because there is no exact match for the item being looked for. Each of these functions contains an argument to specify if an exact match is required for the lookup item. If the argument has been omitted, Excel assumes that it has been sorted and if there is no exact match the error is produced. If you encounter this problem, refer to Chapter 4 and make sure that the function contains a FALSE or 0 argument.

Debugging errors

Once a formula generates an error value, all dependent formulae do the same, unless there is a circularity or calculation has been set to manual (press F9 to force a recalculation anyway). To locate the source of the error browse down the left hand column of the calculations area until you find the first occurrence of the error. Find and fix.

Break and make

A common modelling problem arises when adding new routines to an existing model. The danger lies in failing to link the new logic to the old formulae in the right places. I like to use what I call the "break-and-make" approach: I identify in advance the formulae which will eventually need updating and I break them deliberately, either by using a nonexistent range name (generating the #NAME? error) or dividing the formula by zero (#DIV/0). I then write the new code. When I am ready, I revisit the broken formula and both correct them and amend them with the appropriate references to the new routine. The error values disappear and I am confident that the model has been thoroughly updated.

Audit sheet

The purpose

The audit sheet is an additional sheet in the model in which we carry out and record our quality checks. Models have different purposes and uses, so the audit sheet can be as simple

or as complex as required. It might contain the key documentation described in Chapter 2. We will consider four types of check which we can perform: structural, arithmetical, financial, and change checks. The results of the checks should be recorded and we should recognise that these checks do not necessarily indicate errors or problems; there may be a perfectly valid reason why the analyst has hidden a column, or used a hard-coded value in a calculation. The purpose of audit is to better understand the model and its operation, and to satisfy ourselves that it does what it is supposed to do. If we are auditing our own work we can make corrections as necessary, but when checking someone else's work we should find out who has the responsibility for making changes.

Layout

To set up an audit sheet in an existing model, simply copy an existing output sheet (try Ctrl+click and drag a sheet tab). Delete any existing sheet content, and any number/text formatting, and name the sheet as Audit. This sheet should conform to the same layout as used in the rest of the workbook.

Each audit check is then listed. At the outset, before the checks are carried out, the result of each test is a fail, and we enter the logical operator FALSE. As there may be a substantial number of checks, it is helpful to have an audit summary, which reads TRUE if all checks have been passed, and FALSE if any one or more has failed. In the example below, the results of each test are recorded in column D. I have entered the following formula in a cell in column E:

=AND(D:D)

The AND function is explained in detail in Chapter 4, and the D:D notation in Chapter 3. The effect of the formula is that Excel scans all the results in column D. If any one (or more) reads FALSE, the function returns FALSE. Only if every single test is TRUE does the function return TRUE. We get a problem if a test returns an error message, because the audit check will simply repeat the error. We will return to this particular problem later (p 86). Also, jumping slightly ahead of ourselves, we can give the cell the name AuditCheck (Insert, Name, Create – range names are discussed in Chapter 3).

AuditCheck ▾ fx =AND(D:D)

	A	B	C	D	E	F	G
1							
2					2005	2006	2007
3							
4		All checks passed?			TRUE	*AuditCheck*	
5							
6	Balance sheet check			TRUE	0	0	0
7							

The AuditCheck cell

On each output sheet we can put in a link to AuditCheck, so that the current audit status is recorded on any printouts that we generate.

It would also be helpful if we could list the number of audit tests, and to count those that currently evaluate to FALSE. In a cell below AuditCheck, write the following formula:

=COUNTIF(D:D,FALSE)

This returns the number of tests that read FALSE.
In the cell below, enter the following formula:

=COUNTA(D:D)

This returns the total number of audit tests.

	A	B	C	D	E	F	G
1							
2					2005	2006	2007
3							
4		All checks passed?			FALSE	*AuditCheck*	
5					1	error out of	
6					28	checks	
7							
8	Balance sheet check			TRUE	0	0	0
9							

Counting errors

Structural checks

Structural checks are manual checks, in that we must carry out particular command sequences each time we wish to run a particular audit test. The results must then be recorded on the audit sheet.

Location of inputs

At all times you should be able to identify the input values in the workbook. Although some users suggest using colour I have recommended that inputs should be located only on the inputs sheet. What if they are not?

1 Select the columns containing the calculations.
2 Press F5 (Edit, Go To), and click the Special button.
3 Choose the Constants option. You can then deselect Text, Logicals, and Errors.
4 Choose OK.

Excel now highlights all cells containing input values. I suggest that we apply a fill colour at this point, to prevent us losing the selection and having to repeat the Go To sequence. When running audit checks I use the pink fill colour because it is particularly unpleasant and I never use pink for anything else. Now browse the worksheet to find any coloured cells. Make any corrections or notes and then put in a line on the audit sheet to record that the test has been carried out successfully.

Hardcoded values

These are values which have been written directly into calculations (=E117*1.175) and can be harder to locate. It would be logical to expect the Edit, Go To, Special, Formulas, Numbers command to select formulae which contain numbers. But it doesn't.

An alternative is available, if you feel bold enough. We recognise that all the numbers in the workings and outputs are ultimately derived from the inputs. If there are no inputs, there should be no results.

1 Back up (save) the file.
2 Select all the inputs and delete them (assuming you can locate the inputs in the first place, if not, use the F5 Edit, Go To, Special command to select them).
3 Check the results and workings. If any numbers are visible, they must be hardcoded. Do not rely on this step, because your inputs currently have a value of zero. Dependent formulae, particularly multiplications, will mask any hardcoded values. Also you may have a rash of #DIV/0! errors, which further serve to cover up any values.
4 With the original input cells selected, remove any number formatting (Format, Cells, General, or Ctrl+Shift+~).
5 Type a 1 and press Ctrl+Enter. All input cells should have a value of 1.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		Financial year ending	Units	Base	2005	2006	2007	2008	2009	2010	2011	2012
3												
4		Inflation										
5		rate			100%	100%	100%	100%	100%	100%	100%	100%
6		index		1.00	2.00	4.00	8.00	16.00	32.00	64.00	128.00	256.00
7												
8		Production (units)			1	1	1	1	1	1	1	1
9												
10		Price										
11		real, 2004	unit		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
12		money	unit		2.00	4.00	8.00	16.00	32.00	64.00	128.00	256.00
13												
14		Revenue			2	4	8	16	32	64	128	256

Using predictive values

Now inspect your workings and outputs. You are looking for unusual number sequences. Growth or inflation, for example, are now compounding at 100% and should be generating the exponential sequence 1, 2, 4, 8, 16, 32, . . . You should be able to recognise this in its various permutations in the workings. Anything with a different sequence must be driven by different values. Check and locate any hardcoded numbers. Make any corrections, and then put in a line on the audit sheet to record that the test has been carried out successfully.

References to blank cells

It is a cardinal rule in modelling that formulae must never refer to blank cells, and to do so is to invite disaster. This problem often arises when people decide to delete routines from their models, without realising that there are dependent formulae. In this case I always recommend that we could delete the rows containing the redundant formulae in their entirety, as dependent formulae will then return the #REF! error and can be easily located (the break-and-make technique mentioned previously). In reality I would run a Trace Dependents check before deleting anything.

So how do we find references to blank cells?

1 Select the columns containing the calculations.
2 Press F5 (Edit, Go To) and click the Special button.
3 Choose the Precedents option (note that the Ctrl+[shortcut does not work for this technique).

4 Choose OK. Excel should have selected every formula on the sheet.
5 Immediately press F5, and click the Special button again.
6 This time, choose the Blanks option.
7 Click OK.
8 Run the F5 Edit, Go To, Special command again and select Dependents to trace the formulae which refer to these blank cells.

Hopefully Excel will tell you that no cells were found. If it does not, apply a fill colour and search for the selected cells. When you find them, it may be apparent as to which formula is reading that cell, if it is not, select the blank cell and press Ctrl+] (Select Dependents). This should then reveal the formula with the reference to the blank cell. You should decide what to do about the problem, such as substituting in zero or unit values, or rewriting the formula. Put a line on the audit sheet to record your findings.

You can also do this test in reverse:

1 Select the whole sheet.
2 Use F5 Edit, Go To, Special and choose Blanks. This selects all blank cells in the worksheet.
3 Press Ctrl+] (Trace Dependents).
4 Apply a background colour.
5 Any selected or coloured cells will refer to a blank cell.

Left-to-right consistency

An important modelling rule relating to our calculations is that there should only be one formula on each row; that is, the formula in the first cell of the row must be the same as that in the last cell in the row. There are instances where it might be considered appropriate to have two or more formulae, for example, in some projects we might have a development or construction phase, followed by an operational phase. There are differences in the accounting treatments used in each phase, in that, for example, assets are not depreciated until they are put into use, or that interest is capitalised during development and expensed during operations. It is not appropriate to have different calculations on the same row, because the transition from one phase to the next is indicated by input information such as the start of production or the generation of revenues. If the project slips or is brought forward, the user simply changes the inputs sheet. If the left-to-right consistency rule has not been followed, any such alteration to inputs will require the user to then locate and change the dependent formulae on the workings, and this should be avoided at all costs. If the formula is not updated, the model is wrong. The technicalities of this important issue are considered in more detail in Chapters 3 and 4, but for the moment we are considering audit checks, and so we need to be able to confirm that the left-to-right consistency rule has been followed. This uses another of the Edit, Go To, Special commands.

1 Select the columns containing the calculations. Make sure that the active cell is in the left hand column.
2 Press F5 Edit, Go To, and click the Special button.
3 Choose the Row differences option.
4 Choose OK.

Excel now compares every cell on every row with its neighbours, and highlights any that differ from the first cell in the row. Again, I would normally recommend using a fill colour at this stage, so that you can browse through the worksheet at leisure. There are two common sources of error which we can trap at this stage: first, a formula has been corrected or updated, but has not then been copied across the row; and second, such a correction has been made, but not copied right across the row – this typically happens with mouse users who use the AutoFill click-and-drag technique to copy formulae across the row. These can be easily corrected.

If we locate a genuine difference in formulae in one row which is based on an assumption about the timing of an event, we need then to consider what steps we then need to take – are we able to correct the formula or write a revised routine ourselves, or does the problem need referring back to the model's author or owner for clarification? Make sure the audit sheet is updated accordingly, and the model documentation is updated to reflect your findings and actions, if any.

3-D calculations

Three-dimensional (3-D) calculations are calculations which refer to cells on different sheets, for example,

```
='Balance Sheet'!F27-'Balance Sheet'!E27-'Profit and Loss'!E24
```

I would suggest that this is not the best way to write formulae, not least because even this little formula looks rather more complex than it actually is, simply because each of the references is prefixed with a sheet name. The difficulties of auditing such formulae are notorious – even with F5 Go To/Go Back techniques, it can be time consuming to locate each reference. We also remember that the operational researchers point out that the use of multiple sheets increases the risk of error in itself; as we check each reference on each sheet, we need to verify that the reference itself is correct in the context of that sheet. It induces a certain amount of gloom when reviewing an apparently straightforward model, in which virtually every formula takes up two or more lines on the formula bar (my colleagues refer to this as 'spaghetti modelling'). I would suggest that we work with two-dimensional formulae – calculations that are based on references on the same sheet (this is a key part of the inputs – workings – outputs methodology). And I know that you are going to point out that you know where everything is in your model, and that you don't have a problem with 3-D calculations, but I would suggest that your colleagues are not so familiar with your work, so the less clutter the better.

We can easily locate 3-D calculations because they all have a unique feature – the exclamation mark. All we need to do is Edit, Find (Ctrl+F) using ! as the search string. Unlike the Go To, Special commands we need to repeat the Find command after locating each 3-D calculation. Having confirmed the absence of 3-D calculations, put a line on the Audit sheet to record the result.

In Chapter 1, I recommended that input values are linked through to the workings sheet, for example,

```
=Inputs!E5
```

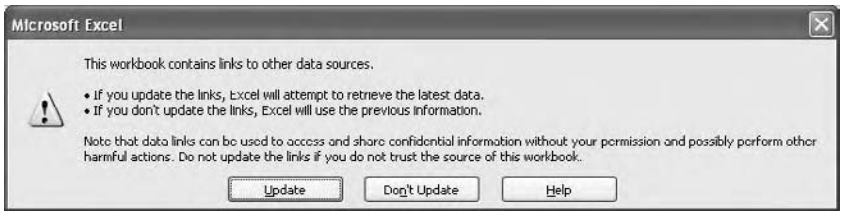
I would suggest, pedantically, that this is an example of a 3-D *formula*, in contrast to

```
='Balance Sheet'!F27-'Balance Sheet'!E27-'Profit and Loss'!E24,
```

which is a 3-D *calculation*. The rule is that we avoid the latter, but 3-D links are acceptable.

File links

Generally, file links should be avoided at all costs. As mentioned in chapter 1, your model is unusable if Excel cannot establish or refresh links to precedent files. I recommended that if file links are to be used, the link formulae are entered on a single sheet for easy reference and management. Excel will normally warn you if there are external links in your workbook when you attempt to open the file.



The file links warning

The Edit, Links command simply lists the files which are linked to your workbook, but it does not tell you where the link formulae are. However, as with 3-D calculations they are easy to locate because the unique feature of such formulae is the use of the [square brackets] around the filename. Use Edit, Find (Ctrl+F) and enter either the [or] as the search string. Amend as required, and update the Audit sheet with an entry to confirm the absence of file links.

I have had people contacting me in some desperation who, having correctly eliminated all such formulae from the entire workbook, still find Excel asking if they wish to update links upon opening the file. The trick here is to check if the model contains range names, and if so, if any of them are links to other files. This can be tested by moving the active cell to a blank part of the workbook and then:

1 Use Insert, Name, Paste, and clicking on Paste List (see Chapter 3). Excel produces a list of all range names in the model.
2 Press Ctrl+* to select this list (Select Current Region).
3 Use the Ctrl+F Edit, Find command to search for the [or], as before.
4 Note the name containing the file link reference, and use Insert, Names, Define.
5 Locate the errant name in the list and click the Delete button.
6 Repeat 3–5 as required.

To confirm that all file links have now been removed, check the Edit, Links command. If it is greyed out, then there are no file links in the workbook. Having done this, we then need to locate any formulae which contain the now deleted range name, using the audit check to locate errors, below.

Errors

In any model there is a possibility that some cells contain error messages, and that these messages are not causing any output problems, perhaps because they occur in some calculations used for a particular scenario. On first inspection all seems to be well, but following further work the errors suddenly manifest themselves. We can use another Edit, Go To, Special command:

1 Select the columns containing the calculations.
2 Press F5 Edit, Go To, and click the Special button.

3 Choose the Formulas option, and uncheck the Numbers, Text, and Logicals boxes.
4 Choose OK.

Apply fill colour to the selection, and browse through the model. Locate and repair formulae, as required. Record your results on the audit sheet.

Hidden columns and rows

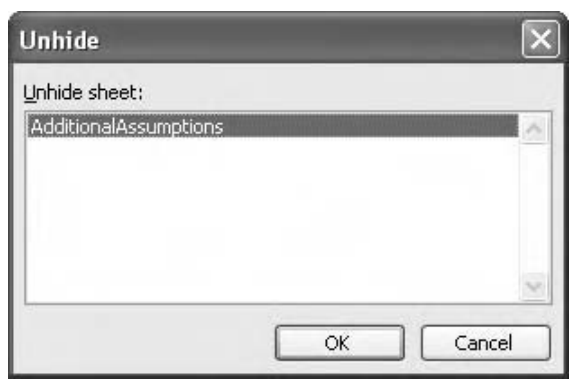
It is quite common to use the Group and Outline techniques (Chapter 5) to make large models easier to handle. In this case, rows and columns are hidden in a structured and systematic way and can be expanded or collapsed as required. At other times columns and rows may be hidden for aesthetic purposes or to simplify printing, and later on we will look at issues concerning quarterly and annual modelling which will require us to hide columns. But there are instances when the analyst has concealed key drivers or formulae and we should be able to locate and explain such hidden detail. The audit check is simple:

1 Press F5 Edit, Go To, and click the Special button.
2 Choose Visible cells only.
3 Apply a fill colour to the cells selected by Excel.
4 Select the whole sheet (Ctrl+A).
5 Use the Format, Column, Unhide/Format, Row, Unhide commands.

Any hidden rows or columns are now clearly exposed. Inspect their contents and if appropriate hide the rows or columns again. Record your results on the audit sheet.

Hidden sheets

Sheets are often hidden to prevent users from accessing calculations or key inputs. Use the Format, Sheet, Unhide command to see the list of hidden sheets. Identify why they have been hidden, and hide again if necessary.



A hidden sheet

If the Visual Basic property of xlSheetVeryHidden has been used, it will not be possible to unhide the sheet without the appropriate access to the macro or VB code.

Merged cells

When centring a heading over a range of cells Excel now creates merged cells. These are a little dangerous in that the grid or matrix structure of the spreadsheet is now compromised. The merged cell is referenced by the cell at the top left of the range, and the contents of the cell can be referred to in formulae. On setting up a merged cell, any existing data in the source cells is lost other than that in the top-left cell. Formulae which refer to other cells included in the merge will return zero values regardless of the content of the merged cell and the audit techniques of F2 and Ctrl+[point to the original locations of the cells even though they no longer exist. There is no simple audit check for merged cells; the only method of which I am aware is to start in column A and press Ctrl+Spacebar to select the column. I then move the active cell and repeat this action in column B, and so on. If when using this technique Excel suddenly selects two or more columns, this is symptomatic of a merged cell. They are not easy to spot, and it may be helpful to apply a fill colour and then to move down through the column pressing Shift+Spacebar to select the row. Once you have found the merged cell, decide whether it is worth unmerging – if it is a title or heading you may be able to accept it. Unmerge using Format, Cells, Alignment, Merge Cells, or click the Merge and Center button. Always press Ctrl+] (Trace Dependents) to make sure that there are no formulae linking to the cell. Record your findings on the audit sheet.

Array formulae

Array formulae are discussed in Chapter 3. They are a particular type of calculation which because of their advanced nature tend to be written only by competent analysts and as such tend not to be a major source of error. They come in two flavours: single cell array calculations, and multiple cell array calculations. They can be difficult to understand and as such their presence in a model should be recorded on the audit sheet. Having looked at how we can find 3-D formulae and file link formulae, it is tempting to assume that we could simply run Ctrl+F Edit, Find, using the unique feature of the array formula which is the curly bracket { or } as the search string. Unfortunately, Excel cannot identify that these brackets exist in the formula, so we have to resort to a cumbersome method similar to that used for locating merged cells. If we select a column in the sheet and press Ctrl+Plus to insert a column, Excel will complain that we are attempting to change part of an array. This technique will locate data tables, which are of course one of the commonest forms of the array calculation. Once an array has been located, press Ctrl+/ to select all the cells in the array.

Excel doesn't like changes to arrays or data tables

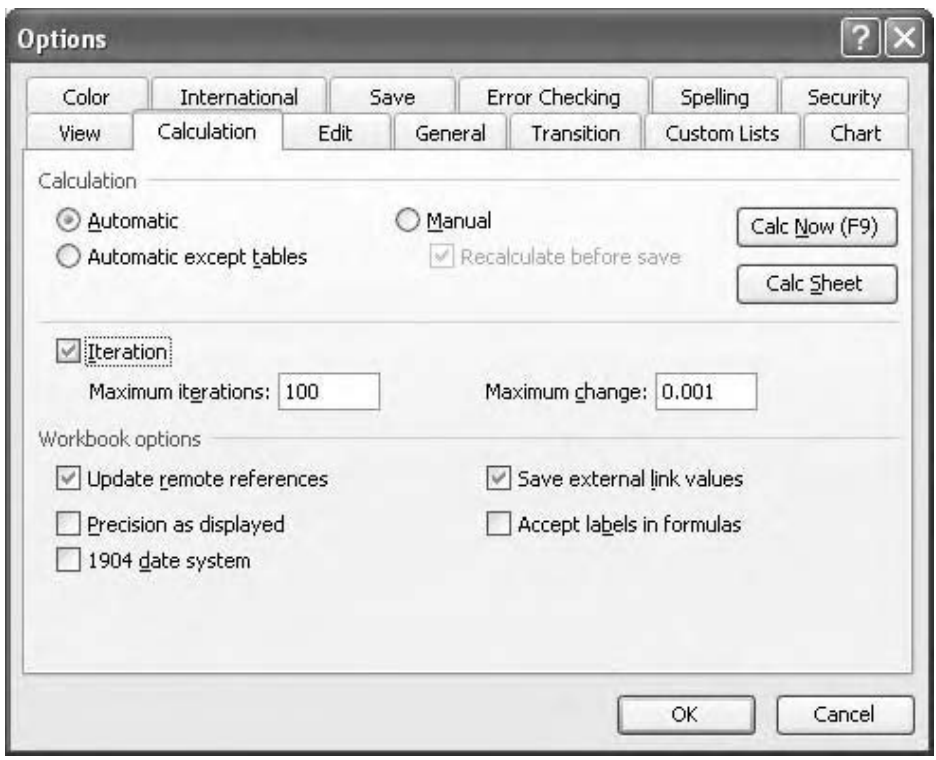
If this action is repeated over the width of the workings area and no such error messages are seen, then there are no multiple cell array formulae.

The curly brackets also disappear if we use Ctrl+` (Tools, Options, View, Formulas), or if we attempt to use techniques such as replacing the = with ^, or prefixing the = with an apostrophe. This means that it is not possible to easily locate single cell array formulae without using a macro.

Remember to record your findings on the audit sheet.

Iteration status

Although not a structural check as such, I include the iteration status check at this point. Iteration is a powerful feature and is explained in much more detail in Chapter 3, but for the moment we should note that it is used to solve circular calculations. The problem with this is that the majority of circularities are accidental and iteration will happily calculate both deliberate circular code and simple slips. We should always check that either iteration is off and that the model is non-circular, or that iteration is on and that the circular code is controlled with a switch (Chapter 3). Use Tools, Options and click on the Calculate tab, and note the Iteration check box.



The iteration command

One of the reasons for running this check in particular is that Excel has a habit of saving the iteration status as a file attribute. When the workbook is re-opened, Excel will switch on iteration without any other warning, and so it is not safe to assume that the iteration status is always off by default.

During model build the iteration should always be off, as it is during the development phase that circular errors are most likely to arise.

Record the iteration status on the audit sheet.

Arithmetical checks

The accountants teach us that if we can calculate the same thing using different methods and get the same result then we can be confident about our work. As the model is being developed, the values generated by our calculations will change, and so these audit checks must be dynamic. The arithmetical checks are designed to check that things add up. Even a simple model can merit a quick cross-check. If the inputs–workings–outputs structure has been used, with or without variations as described in the previous chapter, it is likely that the reports contain totals and subtotals which duplicate calculations in the workings. We can turn this to our advantage by comparing the results. If a discrepancy is found, it will usually be due to one of three common errors:

1 a calculation has been updated to reflect a new element on the workings sheet, but the relevant output sheet has not been updated to show this new element; or
2 a calculation has been updated to reflect a new element on the workings sheet *but not then copied across the row*, and the output has been updated;
3 the output has been updated but with no changes made to the workings sheet.

For example, the project cash flow on the workings sheet should have the same values as the project cash flow summary formula on the Cash Flow report. On the audit sheet, we link the two lines and subtract one from the other. The result should be zero. Put in a logical sum on the audit sheet as the audit check.

4	Project cash flow check										
5	on workings		-20023800	-53517475	16567326	19823717	19350503	15069046	19606749	20664031	17122170
6	on Cash Flow		-20023800	-53517475	16567326	19823717	19350503	15069046	19606749	20664031	17122170
7	check	=SUM(E7:N7)=0	0	0	0	0	0	0	0	0	0

A simple audit check

Other such arithmetical checks might include the financing cash flow, profits before and after tax, increases or moves in cash balances and retained earnings.

Another form of arithmetical check is to test that the assumptions reported in the outputs match the assumptions on the inputs sheet. This serves to ensure that the same units are used throughout – we once reviewed a model of a hydroelectric dam project, in which the energy output was specified as kilowatt/hours. Somewhere in the model it was arbitrarily changed to megawatt/hours, an order of magnitude of difference. This had meant a change in the tariff structure, and a number of problems then arose.

Link the input values from the inputs sheet, and the output values from the reports, and set up a logical sum as the audit check.

Financial checks

Principle

The arithmetical checks simply prove that things add up. If they do not, the underlying problem is usually fairly easy to identify, using the navigation shortcuts from Chapter 1. Financial checks test the underlying financial and accounting policies used in the model, and can be much more difficult to remedy if errors are found. The researchers point out that as financial professionals, finance is described as our 'domain knowledge' and as such the evidence is that there are markedly fewer errors in this area: it is unlikely that we would use an incorrect depreciation treatment, or an out-of-date tax calculation. Whether they are modelled correctly is another story, and thus the need for financial checks. These should include verification that the appropriate accounting and financial rules have been applied – the notes to a company's audited accounts provide a conceptual guide. We also need to recognise that individuals and organisations may have different definitions of the metrics being used in the model: is your understanding of 'cash flow available for debt service' the same as mine? How many different ways do you know how to calculate Return on Capital Employed/Return on Investment? Whether you choose to provide de initions here in the audit section or to set them out in the reports is a decision you need to make soon.

Balance sheet

The first, classic, financial check is that of the balance sheet. A lot of modelling is based on cash flow analysis and balance sheets are not created routinely; some analysts never use them. However, it is worth noting that a lot of people involved in financial modelling hold financial rather than accounting qualifications, and so are not too familiar with the concept of double entry accounting and the basic accounting equation that

ASSETS = LIABILITIES + EQUITY

For example, any tax we incur is paid from our cash flow and also deducted from our profits. Interest payments reduce our cash, and (in most jurisdictions) also reduce our tax liability. Accounting for one part of the transaction and not the other is quickly highlighted if even the simplest of balance sheets is included in the outputs, whether or not it is formally required. Alternatively, set up an abbreviated balance sheet on the audit sheet. We can then test that movements in the cash flow report are reflected in the cash balances, and that movements in the profit and loss or income statement are mirrored in the retained earnings on the balance sheet.

We need to set up the appropriate lines on the audit sheet. If we simply link through the balance sheet imbalance line to this sheet, we will simply see a line of zeros which in a large model simply disappear off the right hand edge of the screen. In the base column we could put in a sum function and test to see that the product of the row is equal to zero.

```
=SUM(E20:P20)=0
```

But the accountants teach us about a type of error called the 'compensating error'; simply put, if we have an imbalance of +10 in one period and −10 in another, the product is zero and would not be detected arithmetically. To avoid this, we can nest each of the balance sheet links in the ABS function:

```
=ABS('Balance Sheet'!E36)
```

The ABS function returns the absolute value, that is, without the sign. This should allow the SUM function to flag up any errors.

But we might also have very minor rounding errors floating around. If the imbalance line reads 0.0022 in a particular period the logical SUM check returns FALSE. On inspection, however, it might be difficult to locate the rounding error, particularly if the audit sheet has been formatted. We could agree a particular level of tolerance, say, 0.01, and enter this into a cell at the top of the audit sheet (and perhaps give it a range name such as Tolerance). We then return to the SUM function and update it:

```
=SUM(E20:P20)<Tolerance
```

A variation of this is to use the ROUND function in which we specify the number of decimal places we wish to include:

```
=ROUND(SUM(E20:P20),2)=0
```

In this example, we are rounding the formula to two decimal places.

Cash Flow

The second of the classic financial checks is the reconciliation of cash flow movements with the cash balance on the balance sheet. My own preference is that the net cash increase and the cash carried forward (closing or ending balance) are linked through to the audit sheet as two identifiable lines, rather than running the test as a single, 3-D calculation. The test is that this period's cash carried forward balance less the last period's cash carried forward balance is the same as the net cash increase (decrease) from the cash flow statement. Put in an appropriate logical sum in the audit column.

Profit and Loss

The third classic financial check is the reconciliation of movements on the Profit and Loss report with changes in the retained earnings on the Balance Sheet. This should prove that the retained earnings carried forward balance for this period, less the retained earnings balance for the last period, should be the same as the increase (decrease) in retained earnings from the profit and loss/income statement.

Ratios

One of the common reasons we build models is to carry out ratio analysis on cash flows for a variety of purposes. I would suggest that such analysis can be used during model development as what I would describe as a rationality check. If you have some experience of working with ratios, and you have an understanding of the entity you are modelling, you may have a gut feeling for what sort of values you might expect to see. If these expectations are set out formally, you and your colleagues are able to monitor them as the model is developed. The liquidity ratios (current assets, acid test), efficiency ratios (stock turnover, debtors and creditors, etc.) and profitability ratios (return on capital employed, gross profit, net profit, etc.) are very helpful. As this list is fairly extensive I will defer to your own professional knowledge and experience to select the ratios of most relevance to your own

requirements. On the audit sheet, list the ratios to be tested, along with your expected threshold values. As with the profit and loss and cash flow checks, link through the source information on a line by line basis, and write the appropriate calculations. These could either be as one-offs, for example, an overall return on capital employed, or on a periodic basis copied across the row. In either case we are concerned that the ratio passes or fails the test, rather than its numerical outcome, so the formula should be a logical test. The audit column cell should then return TRUE if the ratio test is satisfied in all periods, and FALSE if any one (or more) fail. This is a simple AND function, as above.

It is important to document the method for calculating the ratio, as for example, return on capital employed has many flavours. We should also document our assumptions about the threshold values, remembering that they will depend on the circumstances of the model and the company, industry or country. From your experience you will know that not even the textbooks agree on the expected values of particular ratios, but even if you are not that familiar with using ratios I would recommend using them as an audit check, as they can rapidly point out flaws in assumptions or calculations that might be otherwise difficult to spot. Just to give one example, in setting up the receivables (debtors) and payables (creditors) inputs in a model, the values for the debtor days and creditor days were tranposed – instead of expecting the company's customers to pay within 30 days, and the company to pay its bills within 90 days, the figures were reversed. The subsequent working capital results did not merit attention, nor was the effect noticed on the cash flow. However, the current assets ratio (current assets/current liabilities) was in excess of 4 to 1, where we would expect this type of company to have a ratio to the order of 2.5 to 1 or less. If this ratio had been set up as an audit check early on, this error would have been detected and resolved at the time, rather than at the late stage of development that it actually reached.

IRR

The internal rate of return is directly related to the net present value of an investment and its cash flows. Ignoring interest and inflation, if we invest 1,000 today, and receive four annual payments of 250, the net present value is zero and the IRR is 0%. We can use this as an audit check if we recognise different types of elements in the model as being cash flows. As a simple example, we could consider calculating the IRR of a debt. The debt cash flow is made up of the amount drawndown, the repayments, and the interest charged by the bank. From this it may be obvious that the IRR of a debt cash flow must be equal to the interest rate, although this does depend on the interest calculation being used and that the interest rate remains constant. If, for audit purposes, we charge interest on the opening balance and calculate the debt cash flow as the debt drawdown less the repayments less the

4	**Debt**									
5	interest rate			5%	5%	5%	5%	5%	5%	5%
6	interest on opening balance			0	1,000	800	600	400	200	0
7										
8	bf			0	20,000	16,000	12,000	8,000	4,000	0
9	drawdown/(repayment)			20,000	-4,000	-4,000	-4,000	-4,000	-4,000	0
10	cf		0	20,000	16,000	12,000	8,000	4,000	0	0
11										
12	**Cash flow**	IRR	5.00%	20,000	-5,000	-4,800	-4,600	-4,400	-4,200	0

A simple IRR test

interest, the IRR equals the interest rate. We could also recognise that the NPV of the debt cash flow, discounted at the interest rate, should be zero.

Change checks

When working on a model over an extended period of time, with periods of perhaps days or even weeks between modelling activity, it can be difficult to develop a familiarity with the numbers being generated, or a sense of progress. The change check is not an audit check proper, but acts more as a continuity check. Noting the observations about the ratio and IRR tests above, we could identify a number of key results and record their values in the audit sheet – IRR, NPV, debt service cover ratio, whatever. Even though the results are not meaningful, because the model is not yet complete, they can remind us of our previous work and show the effects of recent and current work.

1 Put the appropriate headings on the audit sheet.
2 Write the appropriate key result calculations on the workings sheet, and put links to these formulae on the audit sheet.
3 At the end of the modelling session, copy these results and Edit, Paste Special, Values into the adjacent cells. Put the date above the pasted cells (use Ctrl+; to insert the current date).
4 At the end of the next session, copy the updated results and either paste values on top of the previous or adjacent to them, and date accordingly.

The benefit of this technique is that at the end of each bout of modelling activity we can monitor the effect of the work on these key results and check that the changes tie in with our expectations or understanding of the effects we would expect to see.

Model comparison

Sometimes we have the situation where there are several copies of a model (for example, it has been emailed to colleagues). With the tinkering that can then take place, it can be difficult to determine if each model is exactly the same. Why not add up the workings, or an output sheet? A simple SUM can be tucked away tidily at the bottom of a sheet, or on the audit sheet. The idea can be extended by putting in a SUM at the bottom of each column, which might help identify where the discrepancies are creeping in. This technique is not completely reliable, but can be helpful.

Model map

It can be quite helpful to form an overall picture of the model and we can use a basic model mapping technique to create a graphical view of the model and its components.

1 Make a copy of the worksheet to be mapped – Ctrl+click and drag the sheet tab.
2 Select the entire sheet – Ctrl+A.

3 Adjust the standard column width to 2 units. If the worksheet is particularly long or wide, try changing the Zoom to 50% or less.
4 Still with the whole worksheet selected, press F5 Edit, Go To and click the Special button.
5 Choose Constants, and clear the Text, Logicals, and Errors check boxes. Choose OK. Excel has now selected all cells which contain numbers. Apply a fill colour. If Excel is not able to find any cells, make a note of this result.
6 Repeat Step 4, and then choose Constants, this time clearing the Numbers, Logicals, and Errors check boxes. Choose OK. Apply a fill colour to all selected cells, which in this case contain text.
7 Repeat Step 4 for Constants, Logicals, and Constants, Errors, using a different colour each time.
8 Then repeat Step 4 for Formulas, and each of Numbers, Text, Logicals, and Errors.
9 Now select just the columns in the forecast period (if the model has one). Use F5 Edit, Go To, Special, and Row differences. Apply a fill colour to any cells found.
10 Finally, select the columns of the forecast period and run F5 Edit, Go To, Special and select Precedents. Directly on completion of this command, use F5 Edit, Go To, Special, and choose Blanks. Apply a fill colour.

In the few moments it has taken to carry out these steps, we now have a full map of the worksheet, in which we can visually identify key exceptions to the modelling rules set out in this chapter and elsewhere, in particular the location of inputs and hardcoded values in formulae; breaches of the left-to-right consistency rule, and references to empty cells. By clicking on any of the coloured cells, regardless of its size, we can still read its contents on the formula bar.*

Alternatively, the map can confirm that the worksheet layout conforms to good practice.

If you have reduced the zoom level to below 40%, you will notice an interesting feature in that Excel will display range names in the worksheet. With our practice of naming single rows (see Chapter 3), this is not that useful because the display is so small, but if names have been used to describe blocks of cells then it can be quite helpful.

* We could use conditional formatting for the model map, using ISTEXT and ISNUMBER functions, although we would not be able to differentiate between values and formulae. See Chapter 5.

CHAPTER 3

Mainly formulae

Introduction

The purpose of this chapter is not to show you how to write formulae in the first place, but to consider ways in which you can set them out more clearly and which conform to the principle of error reduction. One of the concerns we have about writing calculations is what I call the 'intellectual challenge' approach: the attempt to solve modelling problems in the least amount of space. The results are formulae of bewildering length and complexity, that require considerable time and effort to understand, and are difficult to audit and review. We have all done it – we have spent some time trying to solve a problem and in so doing we have created a monster of a formula, which takes up four lines on the formula bar and has six closing brackets. And then we look back at the formula a week or so later and amaze ourselves that we were so clever, because we are not quite sure how (or why) we did it.

There are a number of articles and web sites whose authors push hard for this approach. In almost every case I would urge caution, because long, complex calculations are daunting to look at, and the operational research suggests that the longer the formula, the less chance of it being either read correctly or understood. If the reviewer or user is unable to understand a formula, we are failing to give them the confidence they need to rely on the results of the model. We can still tackle complex problems, but I suggest that the task is broken into steps, using as much of the worksheet as required. It may appear more time-consuming, but the logic is simply expressed and should be easy to follow. We have 36,536 rows in a workbook – an awful lot of real estate into which to break down and spread out the calculations.

This chapter explores ways in which we can make formulae simpler to understand and the next considers how functions can be used in this context, although I have included the use of a handful of functions in this chapter to help illustrate the use of particular techniques. Conforming to the overall purpose of the book, the intention is to encourage you to reflect on alternative practice in the light of these suggestions. I am confident that you would be able to generate alternative solutions to the problems and issues that follow. Even if you reject my solutions, you will have greater confidence in your own modelling abilities.

The demonstration workbooks for this chapter are located in Chapter 3: Mainly Formulae folder.

Range names

The big debate

The use of range names is probably one of the most contentious issues in financial modelling. For some they are just plain common sense, but for others they represent the dark side of modelling. In this section, we will look at range names in some detail, and in doing so explore the arguments for and against.

Geographical precision

Many modellers seem to prefer the geographic precision of the cell reference – there is no dispute over the location of Inputs!D4. Indeed there isn't – but what exactly is Inputs!D4? Most of the time we are more interested in *what* the cell contains, rather than *where* it is located. And in Chapter 1 we covered a number of simple navigation techniques which would allow us to visit the cell if required. I have encountered modellers who claim that they can read a formula and intuitively understand the elements referred to by cell reference. I might suggest that intuition is not the most reliable of modelling skills.

Speed

Some modellers argue that it is much faster to write formulae using cell references. This is undoubtedly true, for anyone reasonably proficient with the keyboard and mouse. But the number one error in financial models is that of the pointing error – inserting the wrong cell reference. We certainly should not type cell references, but should point to the cell using either the mouse or keyboard. It is all too easy to click on the wrong cell, and it can be quite tricky to spot this mistake as it happens, particularly if the user is inexperienced or under time pressure. But although not impossible, it is quite difficult to use the wrong range name, and often such a mistake is immediately apparent on inspection of the formula.

The Lotus 1-2-3 legacy

From my own experience I have identified a number of reasons why some people are so resistant to names. Lotus 1-2-3 had range names but they did not work in quite the same way as they do in Excel. For example, the Lotus modeller could write @SUM(Sales) but not

```
+Sales-Costs
```

Lotus names could only be used as a range reference, in functions such as SUM, COUNT, AVG, etc. Having learned this limitation, the Lotus modeller is often not aware of the enhanced functionality of Excel names. I have mentioned before that a lot of people cut their financial modelling teeth with 1-2-3 and by now have progressed to senior levels of the organisation, and I have heard of many cases where the junior (and not-so-junior) members of the team are discouraged from using techniques such as names because of management antipathy.

Poor teaching

I have also found that the subject of range names is often poorly explained. Many people are shown how to set up range names using the text at the start of the row. For example, the text 'total sales' is used to name the adjacent row.

4	London		
5	total sales		
6			
7	Paris		
8	total sales		
9			
10	New York		
11	total sales		

Row headings do not make sensible range names

First, Excel does not care much for the space between the words and substitutes in the underscore character total_sales, which looks scruffy. But second, and more importantly from a modelling perspective, we might have several 'total sales' lines, for example, one for each business unit or region. There is no differentiation between each one, and Excel will only allow one definition, so range names do not appear to offer any value. At this point most analysts immediately dismiss range names altogether.

Range labels

In passing it is worth being aware that Excel possesses a rogue feature called 'Range Labels', in which the text to the left or above a range can be used in formulae as a kind of range name on the fly. This is not worth exploring further. You may wish to confirm that this lightweight feature has been disabled by checking that Tools, Options, Calculation, Accept labels in formulas is off (the default since Excel 2000).

The rule of thumb

There is a trade-off between development time and model usage, and I would agree that certainly in a small model (e.g. the monthly management accounts) the use of range names is superfluous. My simple rule of thumb is that if you cannot see the item being referred to on screen, then it should be named, and the name must exist in the worksheet.

The principle

The underlying principle of using range names is that formulae are readable. Consider the following example:

```
Earnings before tax      =Revenue-CostsOperating-InterestNet
```

(we will look at name conventions later).

If you are familiar with the concept of the earnings (or profits) before tax it should not take too long to spot that I have left out depreciation.
Now consider this formula:

```
Earnings before tax      =E16-E87-E114
```

What's missing from this? In order to discover the missing reference, we need to inspect the other references first. Not too time consuming using the F2, Ctrl+[and auditing tools from Chapter 2, but it detracts from an early understanding of the formula.

Name conventions

Before going too much further with this exposition of range names, it is worth recognising that even in a relatively simple model you could end up with a couple of hundred range names. We should give some thought to how we will organise the names in order to make them manageable, and to ensure that they have real meaning. I normally suggest the following naming convention:

CategoryDescriptionSubDescription . . .

For example,

CostsFixedReal
CostsLabourHour
CostsVariableMoney
CostsOperating

All of the above names are in the overall category of costs. Within this I have provided appropriate descriptions to differentiate one element from the other, and the name should be as long as it needs to be to allow for this. I have also written the names as compound words, with no spaces, with each component identified with an upper case letter. As noted above, Excel will substitute the underscore _ character into any spaces, which works but looks untidy. Some people suggest that a full stop can be used in this context, for example, Costs.Variable.Unit (similar to the command syntax in Visual Basic for Applications, in Chapter 7). Excel does not allow the use of arithmetical operators in range names. Numbers cannot be used for names unless you prefix them with an underscore, but this is not recommended.

Avoid using names that could confuse or mislead, and use common sense if abbreviating or using acronyms. EBITDA is recognisably earnings before interest, tax, depreciation, and amortisation, but would you recognise PCTCT as profits chargeable to corporation tax? What about TVA or MwSt*? Do not use range names that look like Excel functions, for example, SUM or NPV. This does not confuse Excel but may cause problems for your users. Avoid names such as Q1 and Q2 because these are also valid cell references. In fact Excel will not allow you to do this, and will add a trailing underscore _ character to the name without telling you (Q1_ and Q2_), which can lead to trouble if you have not spotted it.

Range names must be unique. If you attempt to create a name that already exists on the same sheet, Excel will prompt you and offer the choice between going ahead with the new name and eliminating the earlier definition, or to stop and set up an alternative name for the current range. Think this one through if it should happen: any existing formulae containing the original name will automatically refer to the new definition. If you create a duplicate name on a different sheet, however, Excel will not provide any warning and will create something known as a local or sheet level name, which we will look at shortly.

---

*Taxe sur la Valeur Ajoutée and MehrwertSteuer respectively – French and German sales tax.

The key point is to make sure your range names are meaningful. We use them so that we can write readable formulae in such a way that other users can understand our work. A logical and consistent approach will enhance this.

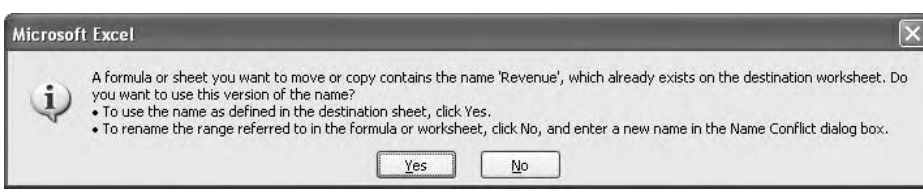
Benefits

There are several further benefits of using range names. As a model is developed, it is likely that blocks of code are moved around, with rows being inserted or deleted as required. Cell reference formulae will still work, of course, but the references will be different, for example, you may have recognised that E45 is your Revenue line, so when it next appears as E63 you may have to spend a moment checking that this is correct. The range name Revenue, however, will remain unchanged regardless of where it is located.

I am often asked, fairly aggressively, "How am I supposed to remember all the names?" There is a simple answer – you aren't. And I normally respond, "Do you remember all your cell references?". You can always use the F3 Paste Name dialog box to refresh your memory.

Re-usable code

Additionally, range names can be re-used. The earnings before tax formula referred to above is the same in all of my models. The elements to which it refers, however, will be in different locations in each new model that I set up. But if I use range names I can simply copy and paste the formula from one workbook to the next. If the names in the formula do not exist in the new workbook Excel will by default treat the names as absolute references and refer to the corresponding cells in the new workbook, until such time as the names are redefined. If the appropriate range names already exist in the new workbook, Excel will offer a choice between using the name as defined in the new workbook or using the name in the original file. The option exists to change the names in the source formula before pasting into the new workbook (Name Conflict). Given that over time, colleagues and users become familiar with your name conventions, new model development can become quite efficient as you re-use code from previous models.



Name conflicts – make sure the pasted name refers to the correct definition

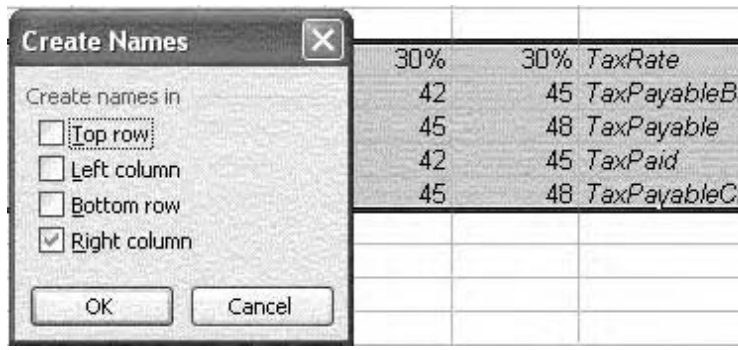
The certainty of names

Range names offer considerable advantages in heavy duty modelling, large complex models perhaps with file links, and with macros. Range names provide an element of certainty that cell references lack: if you need to prepare a monthly income statement derived from

14 business units you probably do not want to waste time hunting around the source files for the detail you require, when you know you need to pick up revenues, costs, and so on. When running command sequences to pick up data from other files, the consistent use of names in the source files can allow for very effective data capture, where the use of cell referencing may be much less reliable – the insertion or deletion of a single row can wreck the process. VBA macros which are written using name ranges will be more robust, and of course the VBA code can be re-used in new models (see Chapter 7).

Creating names

It is good practice to document your names in the workbook and this is easily achieved using the Create Names command. We have already considered the problem in using the row headings as potential names, in that the text down the left-hand side of the sheet is descriptive but also can be repetitive. Instead, put the names on the right-hand side of the range. In this way names are unique, descriptive, and immediately adjacent to the range to which they refer. Select the name and the cells and run the Insert, Name, Create command. This produces a simple dialog box:



Names are created from text located at the edges of a range

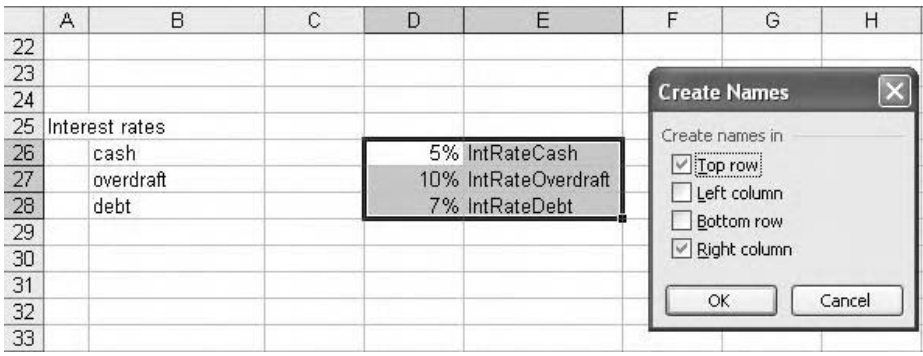
Excel examines your selection and identifies that the names are in the right-hand column (using the simple assumption that this is the only text in the selection; values or formulae cannot be used as names). Click OK. On page 22, we saw the Go to functionality of the Name Box and this can be used to confirm that the name has been created and to confirm the range. Note that the Name Box cannot be widened, so it can be difficult to locate longer range names – in which case use F5. A further tip is to click and drag over a range of cells, and then inspect the Name Box. If Excel recognises the selected range, it will show the name in the box. Selecting an individual cell within a named range will result in the cell reference only appearing in the box.

A shortcut for Insert, Name, Create is Ctrl+Shift+F3.

It is possible to name individual cells as well as rows or columns. The name must be adjacent to the cell, and the usual Insert, Name, Create command used. An individually named cell is effectively an absolute reference.

Insert, Name, Create is not infallible: if you have selected multiple rows and names and there are several blank rows in the selection, Excel may fail to identify the position of

the names and the dialog box is blank. You can help by selecting the appropriate check box. If you run the command and yet the name does not appear to have been created, check that you have selected both the row and the name. If you have a range which contains more rows than columns, Excel can get confused and offer to create names in both the top row and the right column. If you do not spot this, you will discover that the outcome is that Excel creates only the name that occurs in both the top row and the right column, and the range thus named is in the first column. This is typically noticed when the name is used in a formula and returns the #VALUE! error, because one name is attempting to reference more than one cell (see below).



Excel sometimes gets confused

An alternative to using names on the right-hand edge of the range is to set up an extra column on the left-hand side and put the names at the start of each row. This can be useful when your forecast period is very wide. Some users suggest hiding this column once it has been set up, but I prefer to keep it visible so that I can inspect names and ranges in the workbook.

I have referred to range names describing rows in both this and subsequent sections, but the same methods can be used in the context of columns.

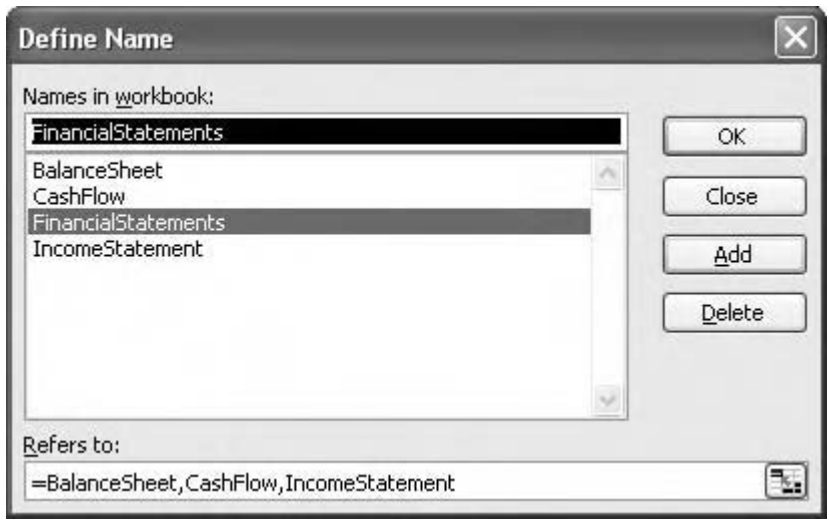
Interestingly, the Undo command does not work for Create Names, but it does work for Define Name.

Defining names

The Insert, Name, Define (or Ctrl+F3) command allows additional functionality. The Define Name dialog provides information about the range names already set up and their corresponding cell references. It also allows for names to be deleted. But because good practice recommends that names should be documented in the worksheet itself, range names should not be set up using this command, the point being that the defined name only exists in the Name Box. You can use it to set up names that you might not want to appear in the sheet, for example, 'PrintArea' or 'BalanceSheet'.

You can also assemble non-adjacent ranges using range names, which can be useful for printing and other purposes. For example, I could assemble the ranges BalanceSheet,

IncomeStatement, and CashFlow, into a single range called FinancialStatements, provided that they are all on the same sheet. To do this, set up the individual names first using Define Names. Then return to the Define Name dialog box and type in the new name. In the Refers to section, type an equals sign followed by the names separated by commas. Important – do not press Enter as Excel tends to ignore what you have just done. Instead, click Add. Tip – if you type the names in lower case you should see them convert to upper case as appropriate.



Assembling non-adjacent ranges into a single range

Combination names can be a little difficult to use. If you press F5, you will not see them listed, but if you type the name Excel will happily highlight the source ranges as required. You cannot perform calculations on combination names, as this will normally generate the #VALUE! error. But you can print them: use File, Page Setup and select the Sheet tab. In the Print area box, type in the combination name (or use F3 Paste Name). If you next select Print Preview, you may be disappointed – if the source names are not physically adjacent to each other in the workbook, Excel will put a page break between them.

There are further tricks using Define Name to follow.

Using names

If we consider the Earnings before tax formula mentioned previously,

=Revenue-CostsOperating-DepreciationTotal-InterestNet

we need to understand what the names mean in this context. I have often encountered very strong criticism from the cell reference modellers who demand to know which Revenue is being referred to in the formula. There is a very simple answer: it is the cell at the intersection of the Revenue row with this column. If the model has been set up so that each column represents one year, then the revenue figure referred to is the revenue figure for that

year. By virtue of the use of the range name, it cannot be the revenue for any other year. To understand this further, type the following formula into a cell: =3:3

This formula is a relative reference to row 3 – by omitting the column reference Excel assumes that this formula therefore refers to row 3 in *this* column, whatever this column actually is (and this is a very efficient way of writing code). A range name is a more meaningful way of setting up such references. A formula such as =Revenue is read as the instruction to look up 'this column' to its intersection with the row called Revenue and to return the value from that cell. This range/column intersection is as unique as its cell reference. If it is not on the current sheet Excel will then look in the same column on each sheet until it finds the correct cell. This should emphasise the key modelling layout rule, that each column has the same function on each sheet. If you cannot guarantee the same layout and structure on each sheet in the workbook, you should avoid the use of range names from the outset.

If the formula is in a cell which does not have a corresponding name in the same column, the usual result is the #VALUE! error. Editing the cell will reveal that the formula is valid (the name is in uppercase and in colour); but Ctrl+[(Go To Precedents) should reveal the problem.

All of this is fine, assuming that every formula you will write should refer to something in the same column. If, for any reason, you need to refer to something in a different column (e.g. a previous balance) you will need to use cell references.

Having set up your range names, there are a number of ways of using them. First, you can type the names directly into your formulae. Tip: if you have created range names using the compound word with initial capitals convention (e.g. CostsOperating), try typing the formula in lower case:

=revenue-costsoperating-depreciationtotal-interestnet

If Excel recognises the names, you will find that it will capitalise the names automatically:

=Revenue-CostsOperating-DepreciationTotal-InterestNet

If you get a #NAME! error, edit the cell and look for the name that has remained in lower case.

=Revenue-CostsOperating-deptotal-InterestNet

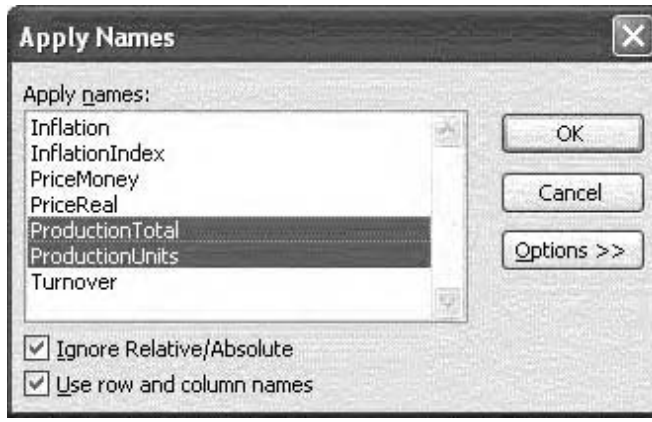
Paste Name

Another way to use names in formulae is to use Insert, Name, Paste and selecting it from the list in the dialog box (the shortcut is F3). As the list of names can be quite long, you can type the first letter of the name you are looking for to scroll more rapidly through the list (if you are particularly fast on the keyboard, you can type up to the first three letters).

Applying names

As mentioned earlier, it is undoubtedly quicker to write some formulae using cell references, and it would be laborious indeed to then rewrite the formula using the appropriate names. Instead, we can use the Insert, Name, Apply command. Excel will scan through the worksheet and identify any cell references that correspond to named ranges, and convert the reference to the name.

When using this command, we often find that Excel has automatically selected one or more names in the dialog box – this is not because Excel is intelligent enough to work out which names need to be applied, but simply because it remembers the last names that were created. Most of the time we need to cherry-pick the relevant names from the list.

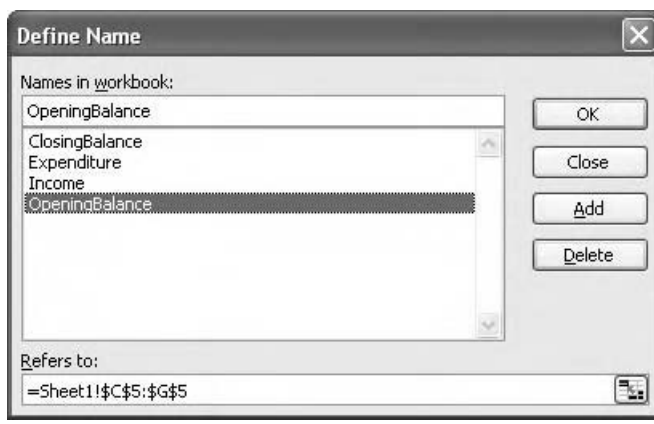


Applying names to existing cell reference formulae

If, after running this command, the references have not been converted to names, check that the reference is to a cell in the same column (names work on the principle that the reference is to the intersection of the current column with the named row), or that the reference is not to another sheet, for example, =Inputs!E5 will not be converted to InflationIn even if that is the correct name to substitute, because of the sheet name in the formula. In this case it will have to be updated manually.

Deleting names

Good practice suggests that you should delete misspelled or redundant range names at the earliest opportunity. Insert, Name, Define (Ctrl+F3) is the only way to delete range names.



Using the Define Name command to delete mis-spelled or redundant range names

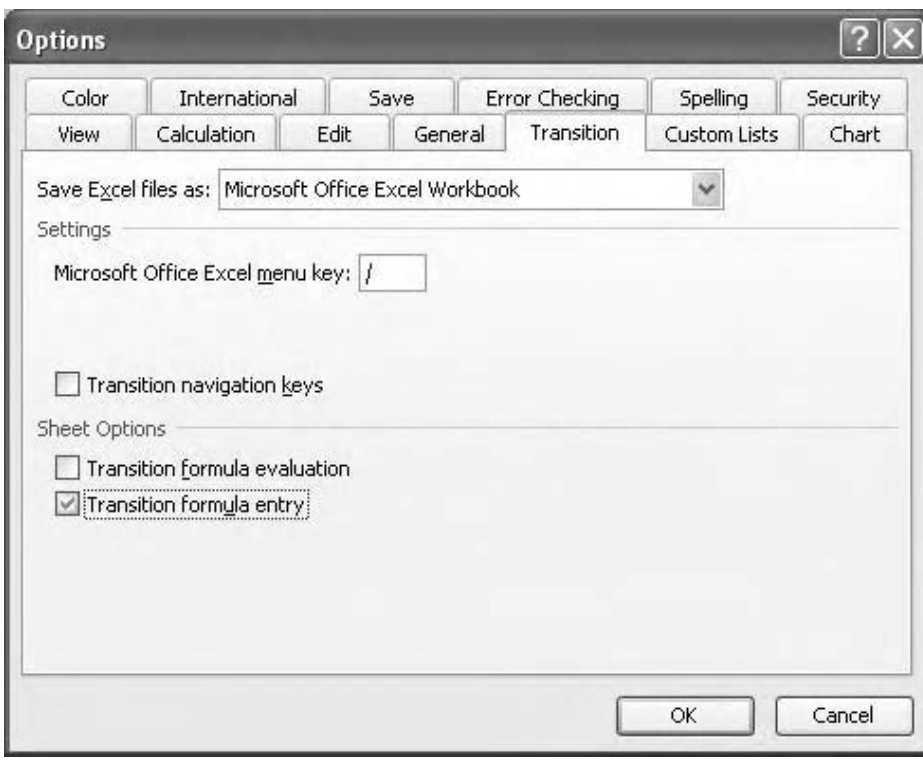
If the deleted name has already been used in a formula, the #NAME? error value appears. Use the audit tricks from Chapter 2 to locate and fix the problem, or use Edit, Replace (Ctrl+H) to substitute a new or corrected name for the deleted name.

Excel only allows names to be deleted individually from the list, and so can prove rather time consuming if there are a lot of them. The usual Ctrl- and Shift-clicking techniques don't work in this dialog box.

Removing names

A few years ago I developed a course in collaboration with a client, and the analyst assigned to the project was vehemently opposed to range names. Each time I sent him the latest iteration of the training model, it would be returned with the range names removed and the formulae replaced with cell references. Although it is a simple operation to convert references to names, Excel offers no functionality to revert names to references. In this case, the analyst was manually rewriting the formulae, a time-consuming and perhaps redundant exercise.

There is one slightly messy way in which we can solve this problem. Although Lotus 1-2-3 used range names, it did not have the same ability to identify the intersection of the column containing the range reference with the named row. Excel was deliberately designed to contain functionality that would help users make the transition from 1-2-3, and so we can take advantage of this to force Excel to behave like its competitor. Assuming



A cumbersome way of removing names from formulae

that we have a model containing formulae using range names, we can use Tools, Options, click on the Transition tab, and choose Transition formula entry. Excel now thinks that it is 1-2-3.

Identify a cell containing a range name formula and press F2. No difference. Press Enter, and then edit the cell again. The range names have now been converted to cell references for the whole of the named range, although you will notice that the formula bar reveals that the names are still there. Repeat as required for each formula. When complete, repeat the Tools, Options, Transition sequence and deselect the Transition formula entry box. Both the formula bar and F2 show that the names have now been removed. The formulae can then be tidied up.

Assuming that we have followed the left-to-right consistency rule, we need only carry out this operation on the left-hand formulae and then copy across when complete. You will now need to use the Define Name command to delete the unwanted/redundant names.

200	Amount subject to tax		-	-	-	1,108,104
201						
202	Tax					
203	rate		30%	30%	30%	30%
204	payable bf		-	-	-	-
205	incurred	0	=E200:N200*E203:N203		-	332,431
206	paid		-	-	-	-
207	payable cf	0	-	-	-	332,431

The formulae resulting from the 1-2-3 remove names trick

Extending ranges

A mild criticism of named ranges is that if the range is extended (extra columns or rows are added), the name may or may not recognise this. It is relatively easy to work around this. If the new columns are inserted from the edge of the working area, then the named ranges do not change. If the new columns are inserted inside the working area, then the range names will expand accordingly. Tip: whether you are using range names or cell references, take care when inserting new columns. Any formula that referred to the previous column will now skip the new columns. Copy the formulae from the last working column to the new edge of the workbook.

5						new columns		
6		0.0	8.5	14.6	21.1		=F9	OpeningBalance
7		16.6	13.2	13.7	17.3		10.6	Income
8		8.2	7.1	7.2	7.6		6.0	Expenditure
9	0	8.5	14.6	21.1	30.8		35.4	ClosingBalance

A formula problem from inserting new columns

Moving named ranges

During model development it is not uncommon to restructure sections of the model. If named ranges are to be moved, we should make sure that the entire range is selected, including the name. If we then Ctrl+X Cut and Ctrl+V Paste, the cells and the name will move to the new location. If only part of the range is moved, the range name will refer to the original location.

The same applies if we copy and paste ranges and appropriate caution should exercised when writing dependent formulae, to ensure that they are not referencing the original ranges.

Names and functions

Range names can cause problems if used with some Excel functions and it is appropriate to consider this now, in advance of Chapter 4, although you may wish to refer to that chapter in this context. What I call range functions (MAX, MIN, COUNT, SUM, AVERAGE, etc.) are designed to operate on a range of cells, so if a range name is used as an argument, the function will act on that range. For example, =SUM(Revenue) will, quite rightly, return the total of the line called Revenue, regardless of which column contains the formula. But if I need to return, say, the smaller of this year's available cash compared to my annual debt repayment, the following formula would give me unexpected results:

=MIN(CashAvailable,DebtRepayment)

The problem is that as an array type function, MIN finds the smallest value in the entire CashAvailable range, compares it to the smallest value in the DebtRepayment row, and returns the smaller of the two. As I copy the formula across the row, I get the same result in each cell. If I press F2 to edit the formula, I notice that Excel highlights both rows, rather than the individual cells I expected to see.

In order for us to use these functions and benefit from range names, we need to provide some extra detail. If I wrote = CashAvailable Excel would have no problem, because it would return the CashAvailable value from this column. If I rewrite the formula slightly, =0+CashAvailable, this calculation works as well. But if I now substitute the calculations into the original formula, I discover that it now works:

=MIN(0+CashAvailable,0+DebtRepayment)

F2 now reveals that Excel has gone back to the column intersections as expected. The reason for this is that Excel does not read formulae and functions from left to right, it applies the rules of arithmetic priority and in this example it evaluates the contents of the brackets first. So it carries out the two calculations as they are written, and it then passes the results to the MIN to process. However, adding zero to a range name does not look too clever, so Excel shows a glimmer of intelligence in allowing us a final trick – omit the zero. The final formula is:

=MIN(+CashAvailable,+DebtRepayment)

Do not omit the comma (argument separator) – we are not adding the two values together. Use the +range name technique for any array type functions. The clue that this amendment is required is that usually (but not always) on writing the formula and copying across the row you get the same result in each cell.

Additional name functionality

The previous sections introduced the key name concepts and techniques, sufficient for most routine modelling purposes. The following sections describe additional techniques which are less commonely used.

Intersection formulae

It goes without saying that cell C1 is the intersection between column C and row 1 and provides a unique reference. If we have row and column names we can repeat the trick with range names. If we had a column named London and a row called Sales, I could write a reference to the London Sales cell in the following way:

=London Sales

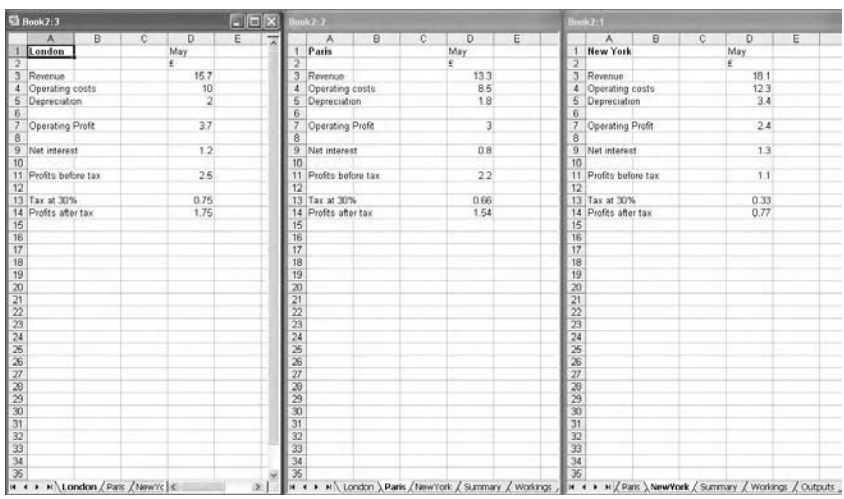
In case you have not spotted it, there is a space or 'intersection operator' between the two names. As this intersection is as unique as a cell reference, there is no requirement to write the formula in the same row or column as the named ranges. If it turns out that there is no intersection between the row and the column, Excel will return the #NULL! error. I include this short paragraph on intersection formulae out of completeness; I do not find it particularly useful, and indeed I have found it rather unstable.

Three-dimensional names

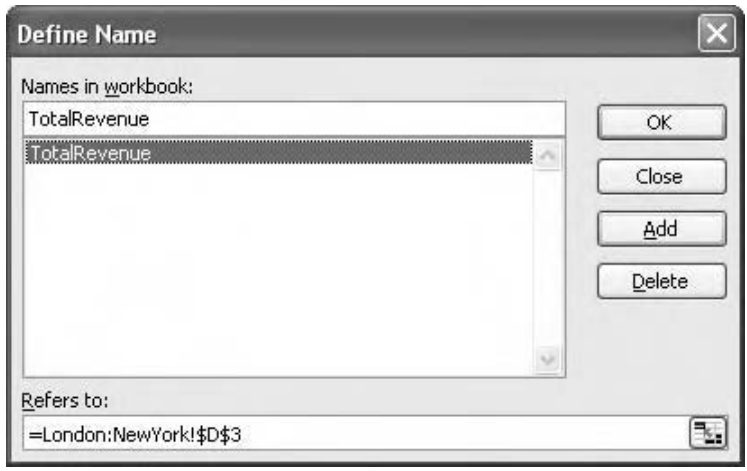
It is possible to set up range names which refer to elements on different sheets, which would be useful for consolidations, but in practice little else. In this example we have three inputs sheets which have the exactly the same layout but represent the profit and loss statements for three business units.

We can create a summary sheet for the totals.

1 Make a copy of one of the inputs sheets to act as the summary sheet (Ctrl+click and drag the sheet tab).
2 Delete the numbers from this sheet.
3 Use Insert, Name, Define (or Ctrl+F3) and type in the range name.
4 In the Refers to box, delete the existing suggestion.



Setting up 3D names



Defining the 3D name

5 Click on the first sheet tab in the range, hold down Shift and click on the last sheet tab. Then click on the cell to be named.
6 Rather than repeating steps 3–5, click Add after defining the name. Type in the next name, and simply change the cell reference in the Refers to box. Only choose OK when all names have been defined.

Test that the 3-D names work by using the SUM function on the summary sheet.

I am not particularly enthusiastic about 3-D names because the standard audit/go to functionality does not work: neither Ctrl+[or F5 will show the precedent cells. Also, if they are to be used at all they should be used for a single dimension – as with naming rows as one row high and several columns across, and columns as one column wide and several rows in length, the 3-D range should be one cell drilling down through each sheet. If multiple cells are named Excel will make no attempt to use the column matching principle from earlier on; and a SUM function will return the value of the sum of all cells in the range, regardless of which column the SUM is copied to. Likewise the reference must be to the same cell on each sheet – we cannot have a 3-D name which refers to E16 on one sheet and E20 on the next. Finally, 3-D names must be defined, they cannot be created, and therefore are unlikely to be documented in the spreadsheet.

Sheet level or local names

If you set up a range name using Create or Define, the name is described as a global name. Provided you are in the correct column, you can write a formula on any sheet referring to a name and Excel will return the appropriate value. As we also noted earlier, if on creating a name, Excel spots that the name already exists on the same sheet, it will warn you. If, however, the duplicate name is on a different sheet, you will receive no warning at all. When you come to use the name in a formula, you may be baffled to find that the name does not return the value you expect. Ctrl+[Go To Precedents also flags up that something has gone wrong. This is because you have set up a local or sheet level name. Let us assume, in the first instance, that this was a genuine error.

Identifying a sheet level or local name

If you have a look in the Define Name dialog box (Ctrl+F3), you will find that the duplicate name has the sheet name showing in the list.

If this is the case, you can simply use the Delete button in the Define Names dialog box to remove the name. Sometimes it is not so simple. If the current sheet has the local name, you will not be able to see the global name in the Define Names list because it is masked by the sheet level name. If you are on a different sheet, you will not be able to see the local name. To get out of this muddle, go to the sheet which contains that local name and delete it. Then delete the global name. Then make any corrections required.

However, you may wish to set up sheet level names deliberately. If we have a workbook in which each sheet represents a separate business unit, it may be appropriate to have the same layout and the same names on each sheet (e.g. Revenue and Costs). This can be done quite simply by setting up one sheet first with all the headings, formulae and names required. Then copy the sheet (Ctrl+click and drag the sheet tab) as many times as is required. Because the same names occur on each sheet, Excel sets up sheet level names for the duplicates. The formulae use the sheet level names, and if you wish to write further formula you can do so.

The original sheet contains the global names, and should be deleted if not required. The now-redundant global range names should be deleted using Define Name. To then summarise the figures from each sheet onto a main report, write the appropriate formulae but this time using the sheet name prefix for each name, for example:

=South!Sales+North!Sales+East!Sales+West!Sales

Problems can arise if you mix and match global and local names. For example, if I fail to delete the source sheet in the exercise above, in addition to the local names South!Sales, North!Sales, etc., I still have the original global Sales. If I attempt to include it in a formula using it's own sheet name, for example:

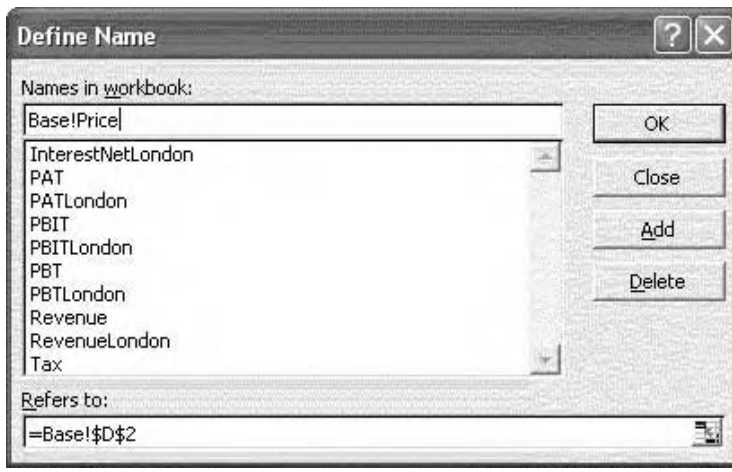
=Original!Sales+South!Sales+ · · ·

Excel pulls an odd little trick by substituting the filename for the sheet name:

=consolidatedsales.xls!Sales+South!Sales+ · · ·

Not very helpful.

To create a sheet level name without copying names from other sheets, we use Insert, Name, Define (Ctrl+F3), and prefix the name with the sheet name, for example, Base!Price – note the use of the ! exclamation mark. For documentation purposes we should then type the name into the worksheet.



Defining a sheet level or local name

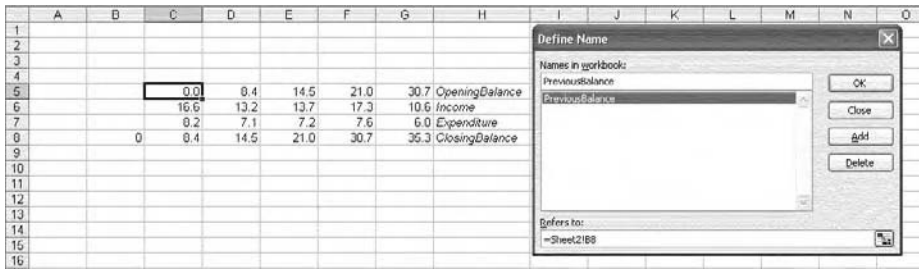
Relative names

I pointed out previously that we can be very comfortable using range names once we understand that they can only refer to the intersection of the named range with the current column: if your formula in column E reads Revenue, then we know with absolute certainty that the Revenue figure is also that for column E – it can not be anything else. Hopefully, as far as your modelling goes, that will be the case.

But you may encounter relative names. An example of this might be where I set up a routine that begins with an opening balance, which is by my definition would be a link to the previous closing balance. Because the previous balance is in the previous column, a normal range name would not be allowed. However, we could set up a range called PreviousBalance using Define Name. Select a cell in the opening balance row, and press Ctrl+F3. Type in the name PreviousBalance, then click in the Refers to box. In the worksheet, click on the actual previous balance cell and note the absolute reference that then appears in the dialog box. The trick here is to press F4 (Absolute) three times until the $ signs disappear. Then click OK.

Now rewrite the opening balance formula using the new name. It should work – if you press F2 Excel should highlight the previous balance. Now try using F5 Go To and choose PreviousBalance as the destination. What happens? You should find that it is always relative to the current cell, whether or not it is actually the previous balance at all. In this example, PreviousBalance is always 3 cells below and one the left of the active cell.

If your colleagues or users are new to range names they are unlikely to thank you for using relative names and in any case I suggest that relative names are best avoided.



PreviousBalance is always three cells below and one to the left of the current cell

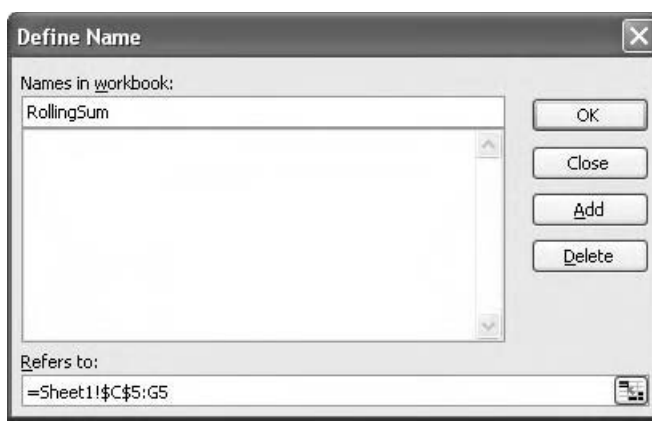
Mixed absolute and relative names

Accumulated depreciation is an example of a rolling total type of calculation, where in this case we need to add up all the depreciation previously charged up to and including the current period. You will probably know that the cell reference version of the formula is:

=SUM($E$80:E80)

As the formula is copied across the row, the $E$80 reference is fixed, and the relative reference changes. To achieve the same effect with range names, try the following:

1 Name the row as required using Create Names (Ctrl+Shift+F3).
2 Write the formula required in the appropriate row. Copy across the row and ignore the result.
3 Move to the first formula in this row.
4 Use Define Name (Ctrl+F3) and select the name.
5 In the Refers to box, amend the reference so that it only refers to the first cell in the range, then delete the $ signs (F4) from the second reference.
6 Click the Add button (not OK).



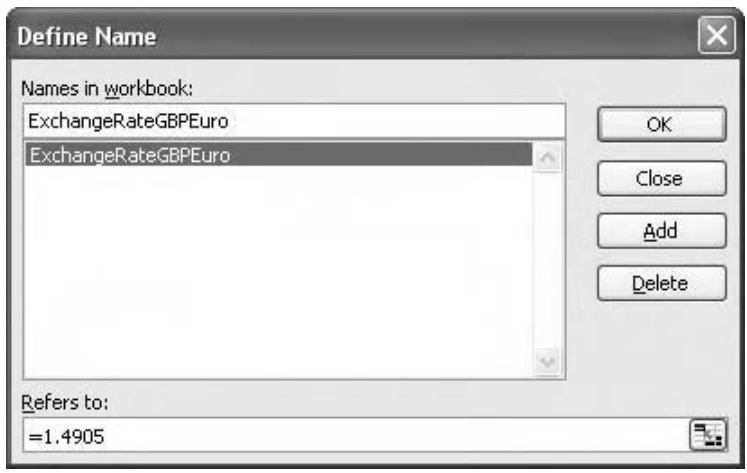
Mixed reference range name

Test that it works.

This technique is not very flexible and may result in unexpected circularities. If you copy the formula to another row, Excel will include the cells between the formula and the original named range. The use of F5 Go To will highlight the potential problems of this technique.

Naming values and formulae

Take, for example, this afternoon's exchange rate between sterling and the euro. I should write it into my model, appropriately named. Instead, I shall use Define Names. I shall use a plausible name, such as ExchangeRateGBPEuro, but instead of referring to a specific cell or range, in the Refers to section of the dialog box I type in the value. I can now amend my currency calculations by multiplying by my new exchange rate, but nobody can see it in the workbook.



Naming a value

Even worse, using F5, ExchangeRateGBPEuro returns an invalid reference. Where is the exchange rate? The only way to check is to inspect the Define Name dialog, or to use the F9 trick from Chapter 2. It may be that you can think of a moral reason for hiding key values such as this, but good practice suggests that all values used in the model should be both easily accessible and properly documented.

We can slide further into the ethical abyss by naming formulae. In this case, you can use Define Names to set up names for formulae which are entered into the Refers to section of the dialog box. It is not quite as straightforward as it could be – a formula such called GrossProfit which is made up of Sales-Costs will return the same value if copied across the row, because Excel only uses the left hand cell from each of Sales and Costs. However, if you write the named formula using cell references it should work.

Dynamic range names

Returning to the issue of having named ranges grow or shrink as data is added or deleted, it is possible to solve this using expanding or dynamic range names. This needs some

careful thought, but is worth exploring. It may be helpful to refer to the section on the OFFSET and Lookup functions in Chapter 4.

Let us create a chart based on a series of sales figures. We would like to add the sales figures for each week as they become available, but we do not want to have to keep updating the chart. Because of the open-ended nature of the exercise, it would be sensible to put the range name in a column to the left of the figures.

	A	B	C	D	E	F	G	H	I	J	K	L	M
31													
32				Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
33													
34	Sales		*Sales*	7.60	6.19	5.90	6.35	9.49	5.54	0.92	9.44	...	...
35													

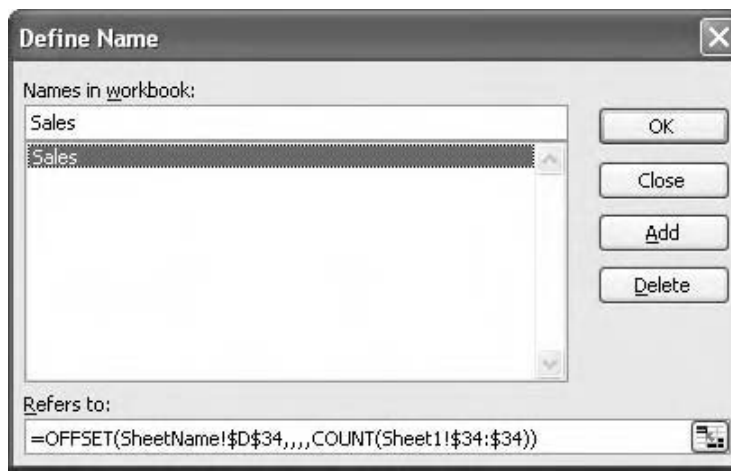
Range name in column C

What we would like to do is to automatically increase this range as the figures are updated.

1 We can use the COUNT function to return the number of values in the row (COUNTA would include the text in the headings).
2 We can use the OFFSET function to define a range that starts in the first cell in the row and ends *n* cells further on, where *n* is the value returned by COUNT. If COUNT increases, the OFFSET reads further across the row.
3 We combine the two functions in the Define Name dialog box.
The syntax is:

=OFFSET(SheetName!$D$34,,,,COUNT($34:$34)) (explanation below)

(note the four commas).
4 Click Add.



A dynamic range name using OFFSET

5 Set up a chart, based on the existing range. Use the Chart, Source Data commands to specify both the Data Range and the Series values are derived from the named range – remember to include the sheet reference otherwise Excel complains continuously.
6 Now add new data and test to see that the range has updated.

The COUNT formula is used to count the number of values in row 34 (in this example). We do not know how many values this will eventually contain, and this referencing looks at the entire row. Make sure the referencing is absolute ($34:$34).

The OFFSET function uses $D$34 as its starting point and reads off the number of cells in the row identified by the COUNT. As further cells are added, the COUNT increases and the range expands. Note the four commas – these are placeholders for arguments that are not relevant here but are explained further in Chapter 4. In simple terms, the COUNT returns the argument for the width of the range.

The dynamic name functionality is useful in the context of the sales chart exercise used here, and in topics such as lookup tables, where it is important that if new data is added to the table, its definition must be updated.

Dynamic names are useful when data is being added. If you start deleting data, specifically from the left hand edge of the range, the COUNT value falls and the OFFSET does not read across the full range. You can solve this by changing the start point cell reference, for example, $D$34 could be changed to $H$34 to exclude the first four weeks. Make sure that the redundant values are then deleted.

Dynamic range names are of interest to the modeller but they bring in the ambiguity inherent in the use of OFFSET. Although F5 Goto is effective, F2, Ctrl+[and the auditing tools are a little vague in identifying the cells in use, because of the full row referencing used in the COUNT function.

Listing names

It can be useful to provide your users with a list of the range names you have set up in the model. Select a cell on your inputs or documentation sheet, and use the Insert, Name, Paste (F3) command and click the Paste List button. Excel lists all your range names in alphabetical order, along with their cell references. There are a couple of useful tricks you can pull here: a quick scan down the list will show you any names that refer to deleted ranges, as they will refer to #REF!. You can use Insert, Name, Define to then delete any of these redundant names. But we can also use the list to answer two important questions: do any names refer to blank references, and are there any ranges which have multiple names?

If you look at the list of names and corresponding references, you might ask yourself how has Excel actually provided the cell references. It displays what is apparently a formula (e.g.) =Workings!$E$4:$K$4. It is possible to do this manually, if you prefix the formula with an apostrophe. But if you look carefully you will see that Excel has not done this. If you now edit the cell and press Enter, you will either return a #VALUE! error or a value, depending on the column. If you then use the Ctrl+[Go To Precedents shortcut, you will highlight the source range. (Note that you must edit the cell first; this method does not work whilst Excel is displaying the cell references).

Empty ranges

To use this method to locate empty ranges, try the following sequence:

1 Paste the name list onto the workings or calculation sheet.
2 F2 Edit each of the cell references, then
3 Select all the references.
4 Press Ctrl+[Go To Precedents.
5 Press F5 Go To Special, and select Blanks.
6 Using the Fill Color palette to apply a background colour.
7 Browse the sheet for any coloured cells.
8 Select any cells that you find, and check the name that appears in the Name Box (assuming the original name text has been deleted from the sheet already).
9 Use Insert, Name, Define to delete the name from the list. (You may wish to Ctrl+F Edit, Find or Ctrl +] the name first, in case it has been used in a formula.)
10 Repeat as required.
11 Record the results on the audit sheet, if necessary.

Multiple names

We can use the name list to locate multiple names used for a range. This typically occurs when a name has been misspelled and then corrected, without having then deleted the misspelled name. However it can also occur during the early stages of a model's development, before a naming convention has been agreed, and the names already in use are then replaced with new names. This can lead to a lot of confusion.

1 F3 Paste List onto any sheet.
2 DO NOT edit the cell references.
3 The names are listed alphabetically, so we use the Data, Sort command to sort the references into ascending order.
4 In the column adjacent to the references, write a logical formula to test if the range reference is the same as the one in the cell below, for example:

```
=E10=E11
```

5 Copy the formula down the column. Unique references will return FALSE.
6 You can write a function to count the number of TRUE values:

```
=COUNTIF(list of references, TRUE)
```

Obviously if this returns zero then no further work is required.
7 Locate all TRUE values and identify the problem names; correct using Insert, Names, Define.
8 Record the results on the audit sheet, if necessary.

External name references

Paste List also allows you to identify names that refer to other files. We noted in Chapter 2 that this is not best practice but in some cases is unavoidable. The name list will include

	PBT	=Workings!$E$100:$N$100	FALSE	1
	PAT	=Workings!$E$101:$N$101	FALSE	
	RetainedEarningsBf	=Workings!$E$104:$N$104	FALSE	
	RetainedEarningsIncrease	=Workings!$E$105:$N$105	FALSE	
	RetainedEarningsCf	=Workings!$E$106:$N$106	FALSE	
	CashFlowPreFinancing	=Workings!$E$109:$N$109	FALSE	
	CashFlowFinancing	=Workings!$E$110:$N$110	FALSE	
	NetCashBf	=Workings!$E$113:$N$113	FALSE	
	NetCashIncrease	=Workings!$E$114:$N$114	FALSE	
	NetCashCf	=Workings!$E$115:$N$115	FALSE	
	CashBf	=Workings!$E$122:$N$122	FALSE	
	CashCf	=Workings!$E$123:$N$123	FALSE	
	OverdraftBf	=Workings!$E$130:$N$130	FALSE	
	OverdraftCf	=Workings!$E$131:$N$131	FALSE	
	TaxableProfit	=Workings!$E$192:$N$192	FALSE	
	TaxLossBf	=Workings!$E$195:$N$195	FALSE	
	TaxLossIncurred	=Workings!$E$196:$N$196	FALSE	
	TaxLossUsed	=Workings!$E$197:$N$197	FALSE	
	TaxLossCf	=Workings!$E$198:$N$198	FALSE	
	PCTCT	**=Workings!$E$200:$N$200**	**TRUE**	
	TaxAmountSubjectToTax	**=Workings!$E$200:$N$200**	**FALSE**	
	TaxRate	=Workings!$E$203:$N$203	FALSE	
	TaxPayableBf	=Workings!$E$204:$N$204	FALSE	
	TaxIncurred	=Workings!$E$205:$N$205	FALSE	
	TaxPaid	=Workings!$E$206:$N$206	FALSE	
	TaxPayableCf	=Workings!$E$207:$N$207	FALSE	

Using Paste List to locate multiple name for a range

the references to other files as file link formula. These can be easily located if you Ctrl+F Edit Find and search for the square brackets [or], back slash/or for the string '.xls'. Alternatively we can sort the list of names by the cell reference column and browse through the addresses.

It should be noted that Paste List will not list sheet level names that have been defined on other sheets. Nor does it differentiate between sheet level and global names.

	CostsOperating	=Summary!$D$4
	Depreciation	=Summary!$D$5
	InterestNet	=Summary!$D$9
	LondonRevenue	='C:\spreadsheets\London.xls'!Revenue
	PAT	=Summary!$D$14
	PBIT	=Summary!$D$7
	PBT	=Summary!$D$11
	Revenue	=Summary!$D$3
	Tax	=Summary!$D$13

An external range name

BODMAS

Brackets, order (power), division, multiplication, addition, and subtraction: the principle of arithmetic priority. One of the ideas promoted in this and the following chapters is to keep formulae short – the longer the formula, the more chance we have of running into a Bodmas slip. You know the example: = 1+2*3 gives the result as 7. =(1+2)*3 is 9. Generally the use of brackets around each 'clause' is helpful, but in more complex formulae I often recommend breaking the formula into a number of smaller calculations. It may seem time-consuming and an insult to the intelligence, but I've seen too many megaformulae in my time which contain simple Bodmas errors.

Timing

Apart from simple calculator-type models, which contain sequences of single calculations which are time-independent, most models will have some period of time under analysis, either as historical data or forecast assumptions. The first rule that applies is that the time periods must be consistent across each sheet (the same column has the same function on each sheet), and we should use the same time periods across each sheet. This is simple in concept, but in practice it can be difficult to apply, particularly in relation to producing the option for quarterly or annual reports. Another common problem is that the timing or duration of certain events is uncertain, with difficulties in writing dependent formulae. In the following section we will consider various techniques which help us work around some of the inherent difficulties in writing robust, time-dependent calculations.

Left-to-right consistency

One of the fundamental rules of models that involve more than one period is that each row should have only one formula, and this was briefly introduced, along with its corresponding audit check, in Chapter 2. Consider a discounted cash flow scenario in which we are considering investing in a new production facility. The investment will probably take place in the third and fourth quarters of this financial year. The accounting rule is that the new asset will not be depreciated until it is put into use, in the first quarter of the next financial year. We need not consider the depreciation treatment in detail, but we need to recognise that for the first two quarters of the analysis there should be no depreciation, and that it should kick in after that. It might be tempting to leave the first two cells in the depreciation line blank; after all, there is no need for a calculation. But what happens if we decide to bring the project forward – on the inputs side the management will type the investment figures in the appropriate cells, but they are then required to locate and update the depreciation formula. The same problems arise if we defer the expenditure – the onset of depreciation should also be delayed. We should avoid users having to update formulae, however well-intentioned.

The modelling rule is that the formula for depreciation must be the same all the way across the forecast period. We need the same formula for the period before depreciation starts, for the duration of the life of the asset, and the same formula must handle the period

after the asset has been fully depreciated, with no user intervention required. We will explore techniques to solve this problem in the sections which follow.

Base column

To adhere to the rule of left-to-right consistency, we need to recognise that some formulae would almost by definition need to be different, in particular the first cell of a row. For example, consider a simple inflation compounding formula that takes the previous value and multiplies it by the current inflation rate:

```
=E5*(1+InflationRate)
```

This works fine when copied across the row, but only if there is a value of 1 in E5 to start off the logic. However, we cannot type this into E5 as this is in the forecast period and breaches the rule about the hardcoded values in the workings area. Furthermore, perhaps E5 itself should be an escalation value, for example, 1.03. This causes a further problem, because the value of 1.03 is dependent on inflation having a value of 3% in the time period represented by column E. If this value changes, the hardcoded 1.03 will not reflect this.

This problem is simply solved using a base column. The purpose of this additional column at the left-hand edge of the forecast area is that it handles such initialisation values and other numbers that may be required for formulae but without breaching the left-to-right consistency principle, nor that of having hardcoded values in the calculations area. The values in the base column are not inputs as such and are not variables, in that their value will not change. They are simply there to make the formulae work.

	A	B	C	D	E	F	G	H
1								
2								
3	Inflation							
4		rate			3%	3%	InflationRate	
5		index		1	=D5*(1+InflationRate)			

Column D acts as the base column

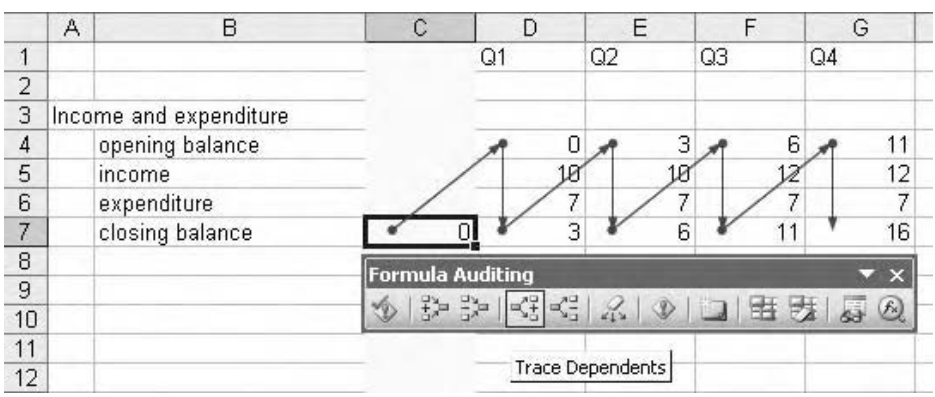
Another way of using the base column is to recognise that it falls outside the forecast area as such, and can be described as the 'now' or 'current' column. In this way it can handle time-independent information, such as the depreciation life of an asset, which will not vary over the forecast period. Also, it can be used to store genuine current information, such as the actual current value of an asset, which then feeds into the forecast calculations. In some types of modelling, for example, company valuation, we may need multiple base columns, which in this case would handle one or more years of historic data.

	A	B	C	D	E	F	G	H
1				2001	2002	2003	2004	2005
2								
3	Forecast inflation							
4		rate					3%	3%
5		index				1	1.03	1.06
6								
7	Growth assumption						4%	6%
8								
9	Forecast (real)				Historic data			
10		revenue		20	21	23	23.92	25.36
11		costs		15	15	16	16.64	17.64
12		operating profit		5	6	7	7.28	7.72
13								
14	Forecast (nominal)							
15		revenue					24.64	26.90
16		costs					17.14	18.71
17		operating profit					7.50	8.19

Extending the base column idea to handle three years' of historic data. Column F now acts as the original base column for calculation purposes

Corkscrew

This is a very simple technique which will be used in material in the following chapters. It allows us to pass information from one period to the next. It can be illustrated using the example of a simple bank account. At the start of the period we have an opening balance, we then deposit funds into the account, and we make withdrawals. At the end of the period we have a closing balance, which becomes the opening balance for the next period. We call it a corkscrew because if we run a Trace Dependents audit check on the code, we see a zig-zag as the balances pass forward.



Using the Formula Auditing Trace Dependents tool to illustrate the corkscrew

When we set up this routine we note that without exception, the opening balance formula is a simple cell reference to the previous balance – it is never a calculation, and it never contains a range name (why not? See page 56 above). In the first period, this means that it is looking outside the forecast period proper, which is why we use the base column. However, it is a cardinal modelling rule that formulae must never reference blank cells, so we initialise with a value of zero for the closing balance in the base column.

In the context of the depreciation problem mentioned above, the corkscrew offers a mechanism where we can identify the cash flows as they occur and roll them up until such time as the depreciation formula should kick in.

	A	B	C	D	E	F	G
1				Q1	Q2	Q3	Q4
2							
3	Fixed assets						
4		bf		0	10	20	25
5		capex		10	10	12	12
6		depreciation		0	0	7	7
7		cf	0	10	20	25	30

The opening balance in D4 is a cell reference link to the zero in the base column C7

You may see different ways of describing the opening and closing balances. The closing balance is sometimes referred to as the ending balance. In English accounting, we often use the expressions 'Brought forward' (b/f or bf) and 'Carried forward' (c/f or cf). As always, be consistent with your own terminology, to avoid confusion later on.

An interesting feature of the corkscrew is that the closing balance is a balance sheet item. Depending on how the corkscrew is set up, it may also include the profit and loss and cash flow items.

	A	B	C	D	E	F	G	H	I	J	K
1				Q1	Q2	Q3	Q4				
2											
3	Fixed assets										
4		bf		0	10	20	25				
5		capex		10	10	12	12		▸ goes to Cash Flow		
6		depreciation		0	0	7	7		▸ goes to Profit and Loss		
7		cf	0	10	20	25	30	⟶	▸ goes to Balance Sheet		
8											

The corkscrew can handle information that is reported on different output sheets

I have often found that corkscrews can offer a simple solution to apparently complex problems, in which analysts have found themselves bogged down in technical problems trying to identify items that have occurred previously instead of thinking about bringing the item forward. Some countries report accumulated depreciation on their balance sheets, which can be solved as a continuous sum of all previous depreciation, or more simply as a basic three line corkscrew.

In the three line corkscrew we can perform the following calculations:

=closing balance−opening balance=increase, or
=opening balance+increase=closing balance

	A	B	C	D	E	F	G
1				Q1	Q2	Q3	Q4
2							
3	Fixed assets						
4		bf		0	8	6	4
5		capex		10	0	0	0
6		depreciation		2	2	2	2
7		cf	0	8	6	4	2
8							
9							
10	Accumulated depreciation						
11		bf		0	2	4	6
12		depreciation		2	2	2	2
13		cf	0	2	4	6	8

Accumulated depreciation corkscrew. The annual depreciation from row 6 is linked to row 12, which adds in the accumulated depreciation from previous years (row 11)

In the four line corkscrew we can calculate:

=opening balance
=additions to the corkscrew
=deductions from the corkscrew
=opening balance+additions−deductions=closing balance

Masks

The technique of masking is simply that of enabling or disabling a particular calculation or routine based on a zero or unit value. If the formula is multiplied by zero, it evaluates to zero, if it is multiplied by 1, we get the expected result. This is another method for tackling the issue of left-to-right consistency. The mask can be either input, in which case the string of 0s and 1s is entered manually on the inputs sheet, or calculated using logical functionality. This is explored in much more detail in Chapter 4, but it is worth looking at the basics here. A simple example should help – I would like to pay a dividend to my shareholders, but under the terms of a loan covenant I am not permitted to do so until the loan has been paid off. We will assume that the bank are willing to vary the duration of the loan subject to our financial circumstances, so we cannot simply type in the dividend formula from the point at which the loan is repaid (and this would breach the left-to-right consistency principle). I have a loan corkscrew in my model which shows the loan being drawdown and then there are a series of repayments until the loan is finally paid off. Whilst the repayments are underway, therefore, I cannot pay a dividend. All we need here is an extra line for the mask.

Rather than writing an IF routine, we simply write the test condition:

=LoanRepayment=0

When you enter this, Excel returns FALSE or TRUE. FALSE has a value of zero and TRUE a value of 1. If we now multiply the dividend calculation by the loan repayment mask,

	A	B	C	D	E	F	G	H	I	J	K	L	M
1					Year 1				Year 2				
2				Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4		
3													
4	Loan												
5		bf		0	8	6	4	2	0	0	0	LoanBf	
6		drawdown		10	0	0	0	0	0	0	0	LoanDrawdown	
7		repayment		2	2	2	2	2	0	0	0	LoanRepayment	
8		cf	0	8	6	4	2	0	0	0	0	LoanCf	
9													
10	Dividend mask			FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	DividendMask	
11				=LoanRepayment=0									
12													
13	Dividend			0	0	0	0	0	2	2	2	Dividend	
14				=DividendCalculation*DividendMask									
15													

Loan repayment corkscrew and dividend mask

dividends are only shown for those years in which there is no loan repayment. Do not worry if you can see problems with this, for example, an interest-only loan, or the possibility of paying a dividend provided we exceed a particular interest or repayment cover ratio, because this is fully explored in Chapter 4.

Flags

Flags are a type of mask but tend to indicate single events, such as the start of operations or the expected date of the completion of a transaction. They use the same TRUE/FALSE functionality, and may be calculated or input.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2				2005	2006	2007	2008	2009	2010	2011	2012	Year	
3													
4	Start year												
5		year	2007	StartYear									
6		flag		FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	StartYearFlag	
7				=StartYear=Year									
8													

The StartYear flag – is the year in row 3 the same as the year shown in C5?

Switches

Switches and masks are similar, but we tend to think of masks and flags as time-dependent, extending across the forecast period with the results being generated for each period. Switches, on the other hand, tend to be single cells and are therefore time-independent. They may also have more functionality than simple true/false outcomes, and a common example of a switch would be a cell used to drive the scenario to be used by the model (see Chapter 6 for a more detailed explanation).

Changing time periods

If we agree with the rule of left-to-right consistency, we are faced with a number of problems where it would seem that this is going to be a very difficult rule to comply with. In this section we will consider a number of techniques to solve this problem.

Quarterly to annual

This is one of the thorniest of all modelling issues: how do we set up a model so that we can report either quarterly or annual results (or monthly to quarterly, etc.)? A basic modelling rule is that the model is set up using the smallest time units required – if there is a requirement to report monthly then the whole model is set up at this level. This in itself is problematic, because with some types of analysis there may be more months in the forecast period than columns in Excel. The usual solutions tend to be a bit messy: the annual total formulae are interposed amongst the quarterly figures, and by judicial use of hidden columns we can laboriously convert from one time period to the other.

		E	F	G	H	I	J	K	L	M	N	O	P	Q
1		Year 1					Year 2					Year 3		
2		Q1	Q2	Q3	Q4	Total	Q1	Q2	Q3	Q4	Total	Q1	Q2	Q3
3														
4	Profits													
5	operating	17.95	17.20	12.49	11.30	**58.94**	10.06	16.30	15.10	11.42	**52.88**	12.11	18.05	17.28
6	before tax	5.21	7.33	7.33	5.94	**25.81**	8.58	7.84	9.16	6.89	**32.46**	5.35	7.04	9.66
7	after tax	1.88	4.26	2.99	1.97	**11.10**	3.05	2.12	1.97	1.12	**8.26**	2.93	1.26	1.00
8														

Wrong solution: profits calculated quarterly, with annual totals in columns I and N

The immediate problem with this type of solution is first it requires a lot of effort (which usually ends up being macro-driven) and second, that it drives a coach and horses through the principle of left-to-right consistency. If any formula needs to be amended or updated, it is no longer a question of simply copying across the row, because this will then overwrite the annual totals. Let us look at a couple of solutions – the principle is the same whether we need to convert from quarterly to annual or from monthly to quarterly, or any other permutation.

Rolling total

One solution is to separate the quarterly and annual calculations into two rows. The quarterly row works as before, and we set up a corkscrew and mask for the annuals.

1 Set up a four line corkscrew with the headings Brought forward, Additions, Retirements, and Carried forward.
2 The brought forward (or opening balance) is a link to the previous closing balance.
3 The additions is a link to the quarterly figure.
4 We will skip the retirements for a moment, and set up the carried forward (closing or ending balance) as the brought forward plus the additions, less the retirements.

		E	F	G	H	I	J	K	L	M	N	O	P
1		Year 1				Year 2				Year 3			
2		Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
3													
4	Profits												
5	operating	17.95	17.20	12.49	11.30	10.06	16.30	15.10	11.42	12.11	18.05	17.28	18.50
6	before tax	5.21	7.33	7.33	5.94	8.58	7.84	9.16	6.89	5.35	7.04	9.66	10.70
7	after tax	1.88	4.26	2.99	1.97	3.05	2.12	1.97	1.12	2.93	1.26	1.00	2.63
8													
9	Operating profit												
10	bf	0	17.95	35.15	47.64	58.94	69.00	85.30	100.40	111.82	123.93	141.98	159.26
11	additions	17.95	17.20	12.49	11.30	10.06	16.30	15.10	11.42	12.11	18.05	17.28	18.50
12	retirements												
13	cf	0 17.95	35.15	47.64	58.94	69.00	85.30	100.40	111.82	123.93	141.98	159.26	177.76
14													

Rolling total #1: basic cumulative corkscrew but with no retirements

5 At this stage, the corkscrew accumulates the quarterly figures over time. We want to drop out the total in the fourth quarter of each year, so we need to set up a mask that identifies the fourth quarter (remember that Q4 is also a valid cell reference, so we enclose it in quotes in the following example).

=Quarter="Q4"

6 The retirements formula adds the opening balance to the additions and multiplies by the mask:

=(Bf+Additions)*QuarterMask

7 The retirements line now shows the annual figures.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1						Year 1				Year 2				Year 3			
2					Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Quarter
3																	
4	Profits																
5		operating			17.95	17.20	12.49	11.30	10.06	16.30	15.10	11.42	12.11	18.05	17.28	18.50	
6		before tax			6.21	7.33	7.33	5.94	8.68	7.84	9.16	6.89	6.36	7.04	9.66	10.70	
7		after tax			1.88	4.26	2.99	1.97	3.05	2.12	1.97	1.12	2.93	1.26	1.00	2.63	
8																	
9	Quarter mask				FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	
10																	
11	Operating profit																
12		bf			0	17.95	35.15	47.64	-	10.06	26.35	41.46	-	12.11	30.16	47.44	Bf
13		additions			17.95	17.20	12.49	11.30	10.06	16.30	15.10	11.42	12.11	18.05	17.28	18.50	Additions
14		retirements			-	-	-	58.94	-	-	-	52.88	-	-	-	65.94	Retirements
15		cf		0	17.95	35.15	47.64	-	10.06	26.35	41.46	-	12.11	30.16	47.44	-	Cf
16																	

Rolling total #2: insertion of the quarter mask and the retirements formula

If we use range names for this, we might end up with the following: OperatingProfitQuarterly and OperatingProfitAnnual. For reporting purposes, we could either have a single version of each report which then contains a reference to, for example, the quarterly value. We can then use Ctrl+H Edit, Replace and change all …Quarterly references to …Annual, and then hide the unwanted columns. Or we could set up two copies of each report, one of which is the quarterly report, the second of which is the annual, with the columns hidden in advance.

	A	B	C	D	H	L	P	Q
1					Year 1	Year 2	Year 3	
2					Q4	Q4	Q4	Quarter
3								
4	Profits							
5		operating			11.30	11.42	18.50	
6		before tax			5.94	6.89	10.70	
7		after tax			1.97	1.12	2.63	
8								
9	Quarter mask				TRUE	TRUE	TRUE	
10								
11	Operating profit							
12		bf			47.64	41.46	47.44	Bf
13		additions			11.30	11.42	18.50	Additions
14		retirements			58.94	52.88	65.94	Retirements
15		cf		0	-	-	-	Cf
16								
17	**Profit and loss**				**Year 1**	**Year 2**	**Year 3**	
18								
19		**operating profit**			**58.94**	**52.88**	**65.94**	
20								

Hiding the columns to produce an annual report

The benefit of this solution is that we have preserved full left-to-right consistency. The downside is that we will probably have to build an awful lot of corkscrews. With some models we are required to produce quarterly reports for the development phase and annual reports thereafter. This technique easily lends itself to this requirement – and is flexible if the requirement changes.

Relative totals

This is a simpler variation of the rolling total method and can be set up using cell references or relative range names. The concept is similar, in that we add up the values from the current and previous three quarters. Every fourth total then represents the total for that year. To do this, your first quarter of the first year must be in column D or further to the right, to avoid it referring to cells beyond the left hand edge of the worksheet.

1 Write a SUM function that adds up the four cells above and to the left.
2 Copy the formula across the row. Test to confirm that the Q4 totals represent the total for the year.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1					Year 1				Year 2				Year 3				
2					Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Quarter
3																	
4	Profits																
5		operating			17.95	17.20	12.49	11.30	10.06	16.30	15.10	11.42	12.11	18.05	17.28	18.60	
6		before tax			5.21	7.33	7.33	5.94	8.58	7.84	9.16	6.89	5.35	7.04	9.66	10.70	
7		after tax			1.88	4.26	2.99	1.97	3.05	2.12	1.97	1.12	2.93	1.26	1.00	2.63	
8																	
20																	
21	Operating profits				17.95	35.15	47.64	=SUM(E5:H5)		50.15	52.76	52.88	54.93	56.69	58.86	65.94	
22																	

The relative sum technique is used in row 21

Use the same column hiding and reporting techniques described previously. Use the relative naming procedure described on page 62 if you wish to use range names. This method is simpler than the corkscrew and far less time consuming, and if you are confident about your model layout it is very robust. However, the corkscrew retirements produced zero results in quarters 1, 2, and 3, so an accidental link to one of these would be rather obvious. The relative total method has values for all quarters and it would be more difficult to spot a mistake. Of course, there is no reason why you couldn't use the mask in this technique as well.

We should give some thought to the ways in which we convert quarterly figures into their annual equivalents. For some figures a total is sufficient, but for others we may need to return an average, for example, with interest or inflation rates. Again the corkscrew offers a simple solution for this. I leave it to you to extend these techniques to consider converting monthly to quarterly to semi-annual to annual.

Circularities and iteration

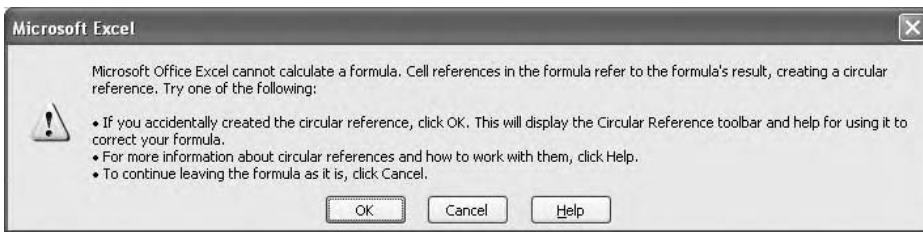
A circular formula is one which directly or indirectly refers to itself. Most of the time they are produced in error, and Excel wastes no time in filling your screen with dialog boxes, Help windows, toolbars and blue audit lines. Quite often it is a simple matter to locate and

rectify a circularity, but in some cases it can be rather difficult, especially if the predecessor trail extends over multiple sheets. However, sometimes we are faced with calculations that are inherently circular. In this section, we will review some techniques for locating the accidental circularity, and then consider how to solve the deliberate or intentional circularity.

You should recognise that a circular model is fundamentally broken. Somewhere you have a calculation that is no longer recalculating, and dependent formulae similarly fail to recalculate. At this point, your model has the functionality of a table in Microsoft Word.

Debugging circularities

We have all had occasions when we have written a trivial formula and suddenly Excel fires up the circular warning. If you are lucky, the blue audit lines which then fill your workbook are meaningful and you can locate the source of the problem. But sometimes the problem is rather more intractable – it is hard to believe but some users continue working on the model despite the warning. Unfortunately, Excel has never had something as useful as a #CIRC! error, which would make life so much simpler. Instead we have to examine the model carefully.

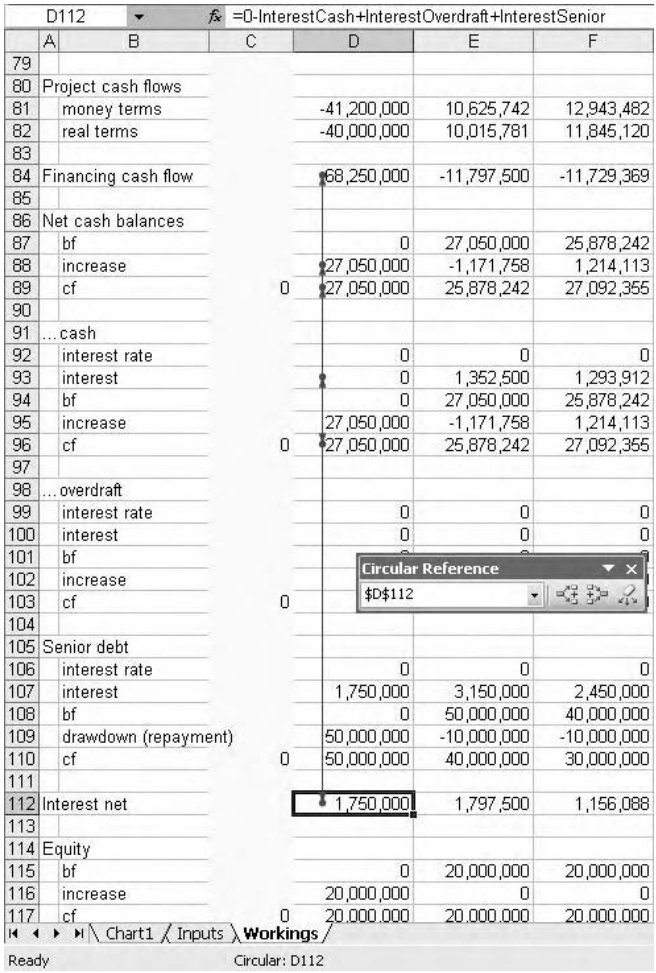


The circular warning. Do not ignore

Having dismissed the circular warning dialog box and closed down the so-called Help window and the circular reference toolbar, note the Circular warning on the status bar. If there is a cell reference next to it, one or more formulae on the active sheet are not recalculating due to the circularity. They may or may not be in the circular path. The cell reference next to the warning will refer to a non-calculating cell (which could be a calculation in the circular path, or a dependent formula that refers to a cell in the circular path). If there is no cell reference then the active sheet is clear. Note that the circular warning remains even if you are working in another workbook – never ignore this warning, because something, somewhere, is broken.

Once you have identified which sheets are involved in the circularity, you can then examine the formulae. There are two things to note here: because Excel is unable to calculate, the formula which gave rise to the circularity will evaluate to zero. Also, formulae in the circular path are not recalculated, so changes to predecessors or to inputs will have no effect on these cells. One neat little trick is to select a cell, press F2 and then press Enter. If the cell is in the circular path, the value originally shown in the cell is lost and a zero appears. Continue with this and eventually you will be able to differentiate between non-circular calculated cells and the circular non-calculated cells. Do not use this technique for a large circularity.

There are two basic approaches to tracking down circularities, both of which are somewhat risky, so the first step is always to back up the file first.



	A	B	C	D	E	F
79						
80		Project cash flows				
81		money terms		-41,200,000	10,625,742	12,943,482
82		real terms		-40,000,000	10,015,781	11,845,120
83						
84		Financing cash flow		68,250,000	-11,797,500	-11,729,369
85						
86		Net cash balances				
87		bf		0	27,050,000	25,878,242
88		increase		27,050,000	-1,171,758	1,214,113
89		cf	0	27,050,000	25,878,242	27,092,355
90						
91		...cash				
92		interest rate		0	0	0
93		interest		0	1,352,500	1,293,912
94		bf		0	27,050,000	25,878,242
95		increase		27,050,000	-1,171,758	1,214,113
96		cf	0	27,050,000	25,878,242	27,092,355
97						
98		...overdraft				
99		interest rate		0	0	0
100		interest		0	0	0
101		bf				
102		increase				
103		cf	0			
104						
105		Senior debt				
106		interest rate		0	0	0
107		interest		1,750,000	3,150,000	2,450,000
108		bf		0	50,000,000	40,000,000
109		drawdown (repayment)		50,000,000	-10,000,000	-10,000,000
110		cf	0	50,000,000	40,000,000	30,000,000
111						
112		Interest net		1,750,000	1,797,500	1,156,088
113						
114		Equity				
115		bf		0	20,000,000	20,000,000
116		increase		20,000,000	0	0
117		cf	0	20,000,000	20,000,000	20,000,000

Excel attempts to determine the circular path. Note the circular warning on the status bar

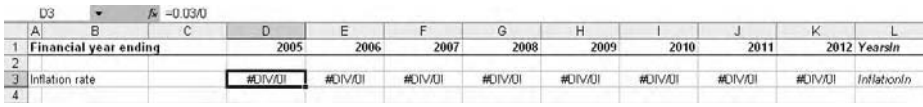
Trial and error

The first method is an unsophisticated trial-and-error approach. If you are dealing with a forecast-type model, you will have many rows of calculations, and hopefully you will be observing the left-to-right consistency rule established earlier in this chapter, whereby the formula in the first cell of each row is simply copied across the row. This means that you do not need to examine the formulae other than in the first column, so delete all the additional columns. Once you have done this, start deleting individual routines, pressing Ctrl+Z (Undo) after each deletion. At some point the circularity will disappear, returning on Undo. Once this happens, delete individual elements within each routine, looking for the same effect. If you are lucky you may be able to locate the circularity quite quickly. Once you have found it, make a note of the problem and its solution, close down the workbook. You should then be able to make your correction in the original file.

This technique is not wholly reliable, in that some circularities are caused by problems in the now-deleted forecast period, for example, MAX/MIN, and some of the lookup type functions. If the circularity disappears when you delete the columns other than the first one, you may have a problem involving calculations across columns, so Ctrl+Z Undo and delete all columns after the first two.

Error tracking

A more reliable solution is to introduce an error message into the model. As noted above, there is no #CIRC! error, so we will have to do this manually. Again, back up the workbook first. Locate an input that is at the head of the longest dependency trail which you think runs through the circularity, for example, inflation or price. It is up to you which error you introduce, but I would normally divide the input by zero, which gives the #DIV/0! message (and I am assuming that there are no such errors already in the model). Alternatively, if you have a strong suspicion concerning the location of the circularity, introduce the error closer to the suspected code.



The original input value has been divided by zero

Inspection of the dependent formulae should now reveal that the error has cascaded through the model – press F9 (Recalc) a couple of times to force the issue. You will see that the error does not penetrate into the circular code, which should still contain numbers. This is a problem, in that the error message is in the clean parts of the model, and the numbers are in the broken part. It would be more helpful if the position was reversed.

The error has cascaded through the workings but has not appeared in row 84.

Go to Tools, Options, Recalculation and switch on Iteration (this is explored in more detail shortly). Press F9 (Recalc) a couple of times to force the error message into the circular path. Next, repeat the command sequence and switch Iteration off. You should see the circular warning reappear, and at this stage the error message should permeate the whole model. Locate the original input cell and remove the division formula. The error message disappears throughout the clean sections of the model and remains trapped within the circularity. You could use conditional formatting to assign a colour to the error cells (see Chapter 5), or a background colour – use F5 Edit, Go To, Special, Formulas, Errors to select the cells.

Selecting cells with error values

Now that the circular path has been isolated, you can work your way through the code in an attempt to locate the source of the problem. Note that F2 (Edit) will cause the cells to revert to zero. If you switch to manual recalculation (Tools, Options, Recalculation, Manual) the numbers will remain. Initially you should concentrate on excluding those routines which you firmly believe are not actually causing the circularity. You could also try variations of the error routine – once you have narrowed down your investigation, try replacing the ' = ' in a precedent formula with a text character such as '^'. The formula is now a text string, and dependent formula now have the #VALUE! error, which can be helpful to differentiate from the surrounding #DIV/0 errors. As text, it cannot be in the circular path, and at some point you will find that the circularity disappears. Restore the formulae as required, make a note of the cause of the circularity, and update the original workbook. Iteration is explained in more detail in the next section

You should always bear in mind that there could be more than one circularity in a model, and the use of different error values can be useful in this context. It can also be helpful to sketch out the components of a model in order to determine the relationships between them and the audit path.

Handling circular code

We are occasionally faced with a problem that involves an inherent circularity which cannot be avoided. We will explore this issue by using an interest calculation as the example but please note this is simply for the purposes of illustration and I am making no suggestion that this is necessarily the correct way to do it.

Let us consider an interest-earning cash account.

D4 =AVERAGE(D5,D6)*D3

	A	B	C	D	E
1					
2	Cash				
3		interest rate		5%	
4		interest		7.5	
5		bf		100	
6		cf		200	

A simple interest calculation

We have the opening and closing balance, and we earn interest at 5%. Using the average balance method, this gives us 7.5 of interest. The effect of this would be to increase the total cash balance carried forward, so we could add the 7.5 of interest to the existing end balance. But we are using the average balance method to calculate interest, in which case we must repeat the calculation, which increases the amount of interest, which then increases closing balance, so we calculate the interest again, and so on. There is a circularity involving the carried forward balance and the interest amount.

If we put this into the model, Excel complains about the circularity and the model locks up. So we go to Tools, Options, Calculation, and switch on the Iteration command. We then find that the interest and the closing balance have been calculated to some detail: 7.692308. We also discover that Excel will happily recalculate the formulae if the inputs change and all dependent formulae now work, so the model is apparently in good working order.

Take a look at the status bar. The original circular warning has been replaced by the Calculate prompt. This normally appears when you have set recalculation to manual, and Excel prompts you to press F9 to recalculate. But we aren't using manual recalculation, and pressing F9 has no effect and indeed the Calculate prompt remains. This is normally the only clue that you will have that someone is using iteration. But what's the problem? The model is working.

Now try the following formula:

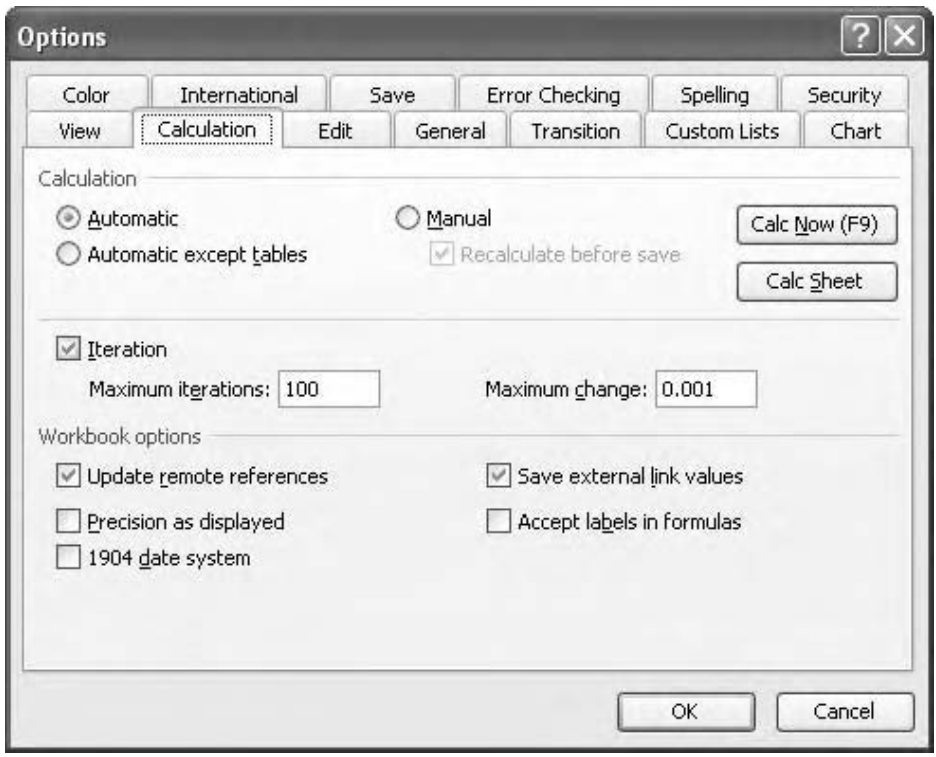
D11 = =SUM(D9:D11)

	A	B	C	D	E
1					
2	Cash				
3		interest rate		5%	
4		interest		7.692308	
5		bf		100	
6		cf		207.6923	
7					
8					
9				1	
10				2	
11				300	
12					

Iteration is on. Where does the 300 come from?

You will have spotted that this formula is circular, but Excel did not complain when it was entered. You may want to consider why the sum of 1 and 2 should equal 300. Before we look at that, just press F9 a couple of times.

If we go back to the Tools, Options, Calculation dialog box, we note that Iteration is part of the Excel recalculation engine. The first point is that we cannot simply iterate part of the workbook, it has to be all or nothing. The second point is that iteration is controlled by two constraints. Excel will either iterate the formula 100 times, or until there is a maximum change of 0.001 (both of these are defaults: the maximum number of iterations is 32,767, and the maximum change is up to 15 decimals). The interest on the cash account was constrained by the maximum change setting, whereas the simple sum was governed by the maximum iterations (1 + 2, repeated 100 times). The interest value will not change until its precedents are changed, but the simple sum will reiterate 100 times every single time the model recalculates.



The iteration command and constraints

The interest routine is described as a converging circularity (i.e. the results are governed by the maximum change constraint), the simple sum is a diverging circularity (the results are governed by the maximum iterations). It is this latter type that is the most dangerous.

The key issue is that after switching iteration on, Excel will iterate all subsequent circular formulae without comment or warning. As we know, the majority of circularities are

errors, so we must be very cautious about the use of iteration. Note that we can't simply turn it off because Excel will only complain about 'a circularity'. If we have other or multiple inadvertent circularities there is nothing to warn us: my simple sum is a deliberately obvious example of what might go wrong; we have seen real life errors ticking over at much lower levels, but with the result that the model gave a different answer each time it was run.

Most investment banks specifically instruct their analysts not to use iteration because of this inherent problem of differentiating deliberate from accidental circularities. These poor souls are left wrestling with the algebra in an effort to work around the problem. With a little planning, however, it is possible to use iteration and to preserve career prospects.

Let us look again at the interest routine: it is a good calculation if the iteration is on, but I would prefer it to make itself non-circular if iteration is off. We could consider the use of an IF test here: in Excel-speak,

IF(Iteration=ON,AVERAGE(Opening Balance, Closing Balance)*IntRate,0)

If iteration is on, therefore, run the circular interest calculation, if iteration is off, return zero (non-circular).

The problem here is the test condition because we cannot write Iteration=ON. There is no direct way of testing the iteration status from a formula. But we can do it indirectly, if the user provides the information about iteration. In a cell at the top of the worksheet, enter the prompt 'Is iteration on?' In the adjacent cell, enter the appropriate response (you may find it helpful to name the cell, Switch). Now update the interest calculation to read:

=IF(Switch,AVERAGE(+Bf,+Cf)*IntRate,0)

(note the plus signs prefixing the range names – refer to page 60 for explanation).

D6 fx =IF(Switch,AVERAGE(+Bf,+Cf)*IntRate,0)

	A	B	C	D	E	F	G
1							
2		Is iteration on?	TRUE	Switch			
3							
4	Cash						
5		interest rate		5%	IntRate		
6		interest		7.5			
7		bf		100	Bf		
8		cf		200	Cf		
9							

Implementing the Switch mechanism

Note that you do not need to specify the condition that Switch=TRUE, because the content of the Switch cell is either true or false and the IF condition must evaluate to either true or false.

We can now test the mechanism. Switch on the iteration and do not forget to update the Switch cell to TRUE. The interest should be calculated. Set the Switch to FALSE and switch off the iteration. The interest reads zero and the circularity has disappeared.

One observation is that when the switch is false, the interest (or whatever is being calculated) reads zero. One workaround is to change the switch formula as follows:

=IF(Switch,AVERAGE(+Bf,+Cf)*Rate,Bf*IntRate) or
=IF(Switch,AVERAGE(+Bf,+Cf),Bf)*IntRate

In both cases, the false outcome is to multiply the opening balance by the rate which generates a number. This is useful if you are developing code in which you need to see the effect of interest (or whatever); the zero value previously used is not very helpful. My own preference is for the original formula: if a user notes that there is no interest being shown on the Profit and Loss statement, for example, it is a significant omission and would be caught. The Bf*IntRate version of the formula results in incorrect numbers and may not be picked up by the users. This does not quite fit in with the Principle of Error Reduction, in that users may believe that the temporary interest value is the correct value.

Some people have suggested that the false outcome could be a text flag, for example, 'Iteration is Off'. This is an option, but any dependent formulae will generate the #VALUE! error, which is probably not worth the trouble.

We can now write circular code that is controlled by the switch. At any stage, we can switch iteration off to test for unintentional circularities. Any and all code that involves iteration should have the switch control built in. As it is most likely that accidental circularities are introduced during the development of the model, it would make sense to leave iteration off until such time as the circular components are under test. Once the model is complete and is being used for sensitivity analysis (see Chapter 6), you can leave the iteration on. A question that may arise, in using the inputs/workings/outputs methodology described previously, is where should the iteration prompt and switch cell be located? The TRUE/FALSE response is indeed an input, and a variable as such, but I would keep them on the workings sheet because the switch drives the circular code. Once the model is ready for the users, iteration is switched on, and the users should not really have any reason to switch it off.

It is very good practice to document the use of iteration so that other users can be confident that other circularities are not being masked. You should note that it is also possible to lie – the switch could read FALSE and yet iteration is still on. For this reason I always recommend that on seeing any sort of switch mechanism you should immediately check the iteration status in the Tools, Options dialog box. This should form part of the audit routine and the outcome should be recorded on the audit sheet.

The iteration off/on routine lends itself to macro automation (see Chapter 7). I prefer not to make it too easy to set up iteration, because users may not understand the full implications of using it and may end up in trouble. If a user does not understand the prompt 'Is iteration on?' it is unlikely that they would go much further with it. You could use the data validation tricks in Chapter 5 to ensure that the user does not input 'Yes' in response to the prompt!

A fringe benefit of using the switch mechanism is that if error values appear in iterated circular code they are almost impossible to resolve, as the error is trapped in the circular loop regardless of any correction. Setting the switch to FALSE and immediately back to TRUE has the effect of purging the error.

Array formulae

I find it hard to enthuse about array formulae. It is true that they can perform some spectacularly complex calculations, but in routine use they are of little value and generally fall foul of the principle of error reduction. They are difficult to edit, difficult to understand, and, if referring to large arrays, they can slow down the recalculation of a model. We should also recognise that few analysts, in practice, would claim to be familiar with array formulae, so you may find colleagues unwilling or unable to help you if you get stuck.

According to Excel Help, an array formula 'can perform multiple calculations and then return either a single result or multiple results'. In practice this means that we can, for example, multiply one block of cells by another block of cells. The formula is written as:

	A	B	C	D	E	F	G
1							
2							
3		This column	times	this column		makes	
4		15.04		10.99		{=B4:B17*D4:D17}	
5		13.82		11.89		164.35	
6		17.32		14.20		245.91	
7		16.27		15.74		255.97	
8		10.98		16.11		176.95	
9		12.42		14.50		179.99	
10		19.74		10.26		202.63	
11		10.17		12.39		125.91	
12		19.44		13.97		271.53	
13		17.43		11.51		200.54	
14		11.75		12.98		152.50	
15		14.84		14.77		219.18	
16		18.64		11.36		211.82	
17		10.94		12.99		142.09	
18							

A trivial example of an array formula

We press Ctrl+Shift+Enter to enter the formula. The commonest errors are failing to select the full range of cells to contain the array formula, and having either too few or too many cells in the selection. The next commonest error is then editing the formula and forgetting the Ctrl+Shift+Enter trick, so that Excel complains that it cannot be part of an array – press Esc to continue.

Excel complains if you try to edit array formulae

Another example is to perform an array calculation with the results in one cell. We can use the audit sheet created in Chapter 2 as an example of this. You may recall that we set up an AuditCheck cell, which returned TRUE if all audit tests had been passed, and FALSE if even one had failed. However, if one of the tests returned an error message, the audit check displays the error. We can easily amend the formula to include an ISERROR check as in the following:

```
=NOT(ISERROR(AND(D:D)))
```

The NOT converts the result to FALSE if the formula does return an error.

We already have a code that identifies how many tests have failed overall, and so it would be helpful if we could return the total number of audit tests that were giving an error message. SUMIF and, more correctly, COUNTIF immediately come to mind, with ISERROR as the logical test. ISERROR does not differentiate existing FALSE values from errors. The array method will do this as a single calculation:

```
=SUM(IF(ISERROR(AuditTests),1,0))
```

Remember to press Ctrl+Shift+Enter, and test to prove that it works.

Note that I have substituted Audit Tests as a range name for D:D, as I have found that using the D:D column referencing technique does not seem to work, although cell referencing and range names seem to function. Again I would remind you that if you are experimenting with this or similar single cell array formulae, you must use Ctrl+Shift+Enter.

In general I would suggest that in routine modelling you should find sufficient functionality in the masking routines and the logic and lookup functions in the next Chapter 4. I would refer you to the specialist Excel formula reference guides for further insight into array formulae. The general modelling rule is to keep formulae short and understandable, and with 65,536 rows you have a lot of space to use.

Coercion

Sometimes the results of our calculations are in formats or layouts that we do not wish to use, and it would be helpful to change the appearance not through formatting but by forcing Excel to change the result. A simple example is that of the TRUE and FALSE operators generated by logical tests. We know that they have numeric values of 1 and 0 respectively and we can happily multiply against these values when using masks and other techniques. However, if we attempt to SUM a row containing TRUE and FALSE values, the result is zero, because Excel does not accept the logical operators as values. In order for this to work, we need to convert (or coerce) the operators by the simple operation of adding zero. For example,

```
=DebtRepayment>0
```

reads TRUE for periods in which repayments take place, and FALSE if they do not. If we need to return the number of loan repayments, a sum of this mask will not work. If we coerce the formula

```
=(DebtRepayment>0)+0
```

we now see a line of 1 and 0 values. The sum of this row will now be valid.

CHAPTER 4

Mainly functions

Introduction

There seems to be a distinct learning curve associated with Excel functions. Most people begin gently with the likes of SUM and COUNT and AVERAGE, and soon graduate to IF and VLOOKUP, and perhaps SUMIF and COUNTIF. And that, for many, is as far as they get, so once they start developing financial models they are overly reliant on this limited group of functions. The purpose of this chapter is to provide you with the twelve or so most useful functions for the general financial modeller. As with Chapter 3, we will look at some examples in the context of techniques combining formulae with functions.

The demonstration workbooks for this chapter are in the Chapter 4: Mainly Functions folder on the CD-ROM.

We can use the Shift + F3 or Ctrl + A shortcuts for function help. Remember that when writing a formula or function we can use F2 to change from Edit to Point mode and back again, and we can use this trick to specify ranges in dialog boxes.

In my years of teaching Excel functions I have found that many people find Excel function Help difficult to use, particularly with its explanations of function arguments and syntax. I take the liberty here of rephrasing some of these elements using expressions that my students have found more understandable.

Logical

Good old IF

The first group of functions to consider are the logical functions: those that depend on one or more conditions being satisfied. Although simple in principle, they are prone to both overuse and abuse, and the IF function in particular is very prone to error. Experience from model auditing has shown that some modellers have a limited understanding of the use of logic and tend to generate unnecessarily complex formulae. You've seen them – calculations which take up four lines in the Formula Bar and have six closing brackets. This approach is unfortunately reinforced by the intellectual satisfaction which some modellers obtain through writing such code.

The basic format of the IF function is:

=IF(test condition, outcome if TRUE, outcome if FALSE)

You know that the basic test condition operators are

=	equality (equals to)	<>	inequality (not equal to)
<	less than	<=	less than or equal to
>	greater than	>=	greater than or equal to

The tests and the outcomes can use values, text (in "quotes") cell references, formulae, and functions.

Let us consider some common examples:

=IF(E10>0,E10,0) If the content of E10 is greater than zero, show that value, otherwise show a zero.

=IF(E10>0,E10*E5, " ") If the content of E10 is greater than zero, multiply E10 by E5, otherwise show nothing. Caution: the lazy " " technique now returns a text string, although the cell will look empty. Any dependent formulae will now return the #VALUE! error. A better solution is to return a zero and use appropriate formatting (Chapter 5).

=IF(E10=0,E10,E10*E5) If the content of E10 is equal to zero, show zero, otherwise multiply E10 by E5. Caution: E10 must be *exactly* zero for this to work.

Now let us extend this using an example that requires us to split positive (cash) and negative (overdraft) values from a cash flow (net cash):

Cash	=IF(NetCash>0,NetCash,0)
Overdraft	=IF(NetCash<0,0,NetCash)

I hope you can spot the mistake. The cash formula tests to see if net cash is positive and shows the value if it is. The overdraft formula tests to see if net cash is negative and shows a *zero* if it is. I correctly changed the direction of the test from > to <, but inadvertently switched the true/false outcomes as well. The second formula is in fact a cash formula.

A MAX/MIN solution

Let's consider an alternative solution for this problem. Instead of using an IF, let us try MAX and MIN:

Cash	=MAX(+NetCash,0)	or	=MAX(0,+NetCash)
Overdraft	=MIN(+NetCash,0)	or	=MIN(0,+NetCash)
(or alternatively,	=MAX(0-NetCash,0)		

The MAX function works by returning the largest value from the arguments in the brackets. In this case, it compares the NetCash value to zero. The MIN returns the smallest value. MAX does not return positive numbers in itself, but in this case the comparator is zero so it returns any value greater than that. I have written each formula in two ways to show that the order of the arguments doesn't matter, unlike the IF. MAX and MIN can evaluate up to 30 arguments.

The purpose of the + in front of the range name is to prevent Excel from evaluating the largest value in the whole NetCash range (page 60).

Although I have only used trivial examples so far, we need to recognise that IFs tend to cause problems because people do not pay enough attention to the detail. If one formula seems to work, it is assumed that the next one works as well. The real problems arise when we extend the IF with further conditions.

```
=IF(E10>0,E10,IF(E10>100,E10*E5,0))
```

This contrived example introduces the nested IF but also fails miserably. If E10 is greater than zero, show that value, otherwise if E10 is greater than 100, multiply by E5, otherwise show zero. The problem is that any value of E10 that is positive will satisfy the first IF condition, so that if E10 is 1 or 1000 the function will return that value. The multiplication by E5 will never happen. The formula should read:

```
=IF(E10>100,E10*E5,IF(E10>0,E10,0))
```

Using AND and OR

At this stage we will look at techniques for building more complex conditions. If we need to specify that two conditions must be satisfied (e.g. E10>0 and E5>10) we can write an AND function:

```
=IF(AND(E10>0,E5>10),E10*E5,0)
```

The AND function returns TRUE if both arguments evaluate to TRUE. If either or both return FALSE, the AND returns FALSE. AND will allow up to 30 individual tests to be evaluated.

If we require either argument to be true, rather than both, we can use the OR function:

```
=IF(OR(E10>0,E5>10),E10*E5,0)
```

In this case, OR will return TRUE if either test is TRUE. Only if both tests are FALSE will it return FALSE. As with AND, up to 30 tests can be evaluated. Occasionally the product of the AND/OR test may be opposite to the result you actually need. Imagine that should both E10 be greater than zero and E5 greater than 10, the FALSE outcome of the IF test should be followed. This can be done using the NOT function:

```
=IF(NOT(AND(E10>0,E5>10),E10*E5,0)
```

Let's apply the principle of error reduction.

Starting again

I have not really gone into too much detail with these last examples because in my opinion we are moving into dangerous waters and I am going to suggest some alternative ways

of composing this logic that should be more readable and therefore less prone to error. Let us go back to the original IF test:

=IF(E10>0,E10,0)

Try writing just the test condition itself into a cell:

=E10>0

Excel will evaluate the formula and return TRUE or FALSE. These are referred to as the logical operators. If you rewrite the formula:

=(E10>0)+0

Excel will return a 1 or a 0; 1 being TRUE, 0 is FALSE. The addition of the zero forces Excel to return the value of the logical operator; some people prefer the values to the operators.

The logical mask

This functionality offers an alternative approach to building logic into models; instead of constructing long, cumbersome IF formulae we can break the underlying logic into individual steps, with each step carrying out a single logic test. This links back to the concept of the mask, introduced on page 75.

Breaking down the =IF(E10>0,E10,0) formula into steps, we have

Mask	=E10>0
Calculation	=Mask*E10

This is a very simple illustration, and you are probably thinking that it seems redundant to split a one cell formula into two cells. But now let us extend it with the same multiple conditions used above:

=IF(AND(E10>0,E5>10),E10*E5,0) can be expressed as:

Line 1	=E10>0
Line 2	=E5>10
Mask	=AND(Line 1, Line 2)
Calculation	=E10*E5*Mask

And similarly,

=IF(OR(E10>0,E5>10),E10*E5,0) can be expressed as:

Line 1	=E10>0
Line 2	=E5>10
Mask	=OR(Line 1, Line 2)
Calculation	=E10*E5*Mask

Finally,

=IF(NOT(AND(E10>0,E5>10),E10*E5,0) is:

Line 1 =E10>0
Line 2 =E5>10
Line 3 =AND(Line 1, Line 2)
Mask =NOT(Line 3)
Calculation =E10*E5*Mask

The mask technique can be very useful for identifying events. For example, the accounting rules relating to depreciation generally specify that assets are depreciated when they are put into use, which may be some time after they were purchased. If we can find a driver that indicates that the asset is in use, we can build a simple mask to control the depreciation formula.

Mask =Production>0
Depreciation =(depreciation formula)*Mask

	A	B	C	D	E	F	G	H	I	J
1				2005	2006	2007	2008	2009		
2		Production								
3		units		0	10,000	10,000	10,000	10,000	Production	
4										
5		**Mask**		**FALSE**	**TRUE**	**TRUE**	**TRUE**	**TRUE**	**Mask**	
6										
7		Fixed assets								
8		asset life	4	years						
9		bf		0	100000	75000	50000	25000		
10		capex		100,000	0	0	0	0		
11		**depreciation**		**0**	**25000**	**25000**	**25000**	**25000**		
12		cf	0	100,000	75,000	50,000	25,000	0		
13										

This approach can be too simplistic. If the driver is not directly associated with the asset, we could end up with the mask returning the wrong value and suspending depreciation, which is not acceptable. The test of Production>0 only returns information about the current level of production, rather than the more important fact that the event of production starting (and therefore depreciation) has taken place. Let us change the routine and add an additional line:

Production is happening =Production>0
Production has started =OR(previous cell, +ProductionHappening)

Previous cell is a reference to the cell immediately to the left of the formula cell, and this new line needs a FALSE in the base column to start it off. The formula looks at the previous result and compares it to the current production level. The OR returns TRUE if either cell contains a TRUE. What makes this interesting is that when the new ProductionStarted line flips over to TRUE, it *cannot* revert back to FALSE. Ever. If production is halted, the formula still returns TRUE because the previous cell reads TRUE. If we make the depreciation formula read off the second line, we find that depreciation will now continue regardless of production.

D7 =OR(C7,+ProductionHappening)

	A	B	C	D	E	F	G	H	I	J
1				2005	2006	2007	2008	2009		
2	Production									
3		units		0	10,000	10,000	10,000	10,000	Production	
4										
5	Production									
6		is happening		FALSE	TRUE	TRUE	TRUE	TRUE	ProductionHappening	
7		has started	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	ProductionStarted	
8										
9	Fixed assets									
10		asset life	4	years						
11		bf		0	100000	75000	50000	25000		
12		capex		100,000	0	0	0	0		
13		**depreciation**		**0**	**25000**	**25000**	**25000**	**25000**		
14		cf	0	100,000	75,000	50,000	25,000	0		
15										

We can also write this routine without the OR. The formula can be written as:

=previous cell+ProductionStarted>0

This produces the same result.

Even with these trivial examples, we are trying to set out our work in a way that can be easily followed. Each line contains only one action, be it a logic test, a mask, or a calculation. One criticism is that apparently simple logic is now spread over several lines, but I think the ease with which we can revisit and understand what can be complex logic sequences more than repays the extra effort in setting up these routines.

Putting it into practice

An example should help. Let us consider commercial bank loan. We will imagine that we can negotiate a repayment holiday, which means we can drawdown the funds and simply pay the interest until the repayment period starts. At this stage we are not sure how long the repayment period lasts, nor are we sure about the duration of the loan.

The modelling problem is in two parts: we must identify a trigger to commence repayments and a trigger to stop repayments. Repayments therefore only take place in the time between the end of the repayment holiday and the completion of the repayment. The principle of left-to-right consistency demands that we can only have one formula for debt repayment.

To begin with, we will borrow 1,000,000 to be repaid over 10 years, with a 2-year repayment holiday at the start. We may want to defer the drawdown to the second year, and the duration of the loan and the repayment holiday are variables. Assuming equal repayments each year, the annual repayment will be:

=SUM(Amount)/(Duration – RepaymentHoliday)

that is, 8 annual payments of 125,000.

We can set this up in the model:

AnnualRepay... =SUM(Amount)/(Duration-RepaymentHoliday)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		10	*Duration*									
6		repayment holiday		2	*RepaymentHoliday*									
7		annual repayment		125000	*AnnualRepayment*									
8														
9		bf		0										
10		drawdown		1000000	0	0	0	0	0	0	0	0	0	
11		repayment												
12		cf	0											
13														

We can set up a corkscrew to handle the debt:

L11 =AnnualRepayment

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		10	*Duration*									
6		repayment holiday		2	*RepaymentHoliday*									
7		annual repayment		125000	*AnnualRepayment*									
8														
9		bf		0	875,000	750,000	625,000	500,000	375,000	250,000	125,000	0	-125,000	*Bf*
10		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
11		repayment		125,000	125,000	125,000	125,000	125,000	125,000	125,000	125,000	125,000	125,000	*Repayment*
12		cf	0	875,000	750,000	625,000	500,000	375,000	250,000	125,000	0	-125,000	-250,000	*Cf*
13														

Note that simply pulling in the annual repayment figure causes the debt to be overpaid. We need to suspend the repayments during the first 2 years. One solution is to set up a quick mask based on the year number or column number, along the lines of =Number>RepaymentHoliday, and on first pass this would do the trick.

M14 =AnnualRepayment*Mask

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		10	*Duration*									
6		repayment holiday		2	*RepaymentHoliday*									
7		annual repayment		125000	*AnnualRepayment*									
8														
9		number		1	2	3	4	5	6	7	8	9	10	*Number*
10		repayment mask		FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*Mask*
11														
12		bf		0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	*Bf*
13		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
14		repayment		0	0	125,000	125,000	125,000	125,000	125,000	125,000	125,000	125,000	*Repayment*
15		cf	0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	0	*Cf*
16														

But later on I shall consider the effect of deferring the loan drawdown, that is, borrowing the funds in year 2 or year 3. If this happens, and we assume that we are still entitled to the repayment holiday, then this mask will not work. If we consider a number sequence that has a 1 in the year in which the first repayment is due, we can write a formula such as = Previous + 1 where Previous is a cell reference to the preceding cell. If this is copied back to the start of the row, it does not work until you type –2 in the base column. You may recognise this as =0−RepaymentHoliday. At this point, the number sequence (or index) is now driven by the input repayment holiday value, and the first repayment is

due wherever the index reads 1 (although this still does not solve the problem of the deferred drawdown). The mask formula can be written as:

=HolidayIndex>0 and the repayment formula becomes
=AnnualRepayment*Mask

C9 =0-RepaymentHoliday

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		10	*Duration*									
6		repayment holiday		2	*RepaymentHoliday*									
7		annual repayment		125000	*AnnualRepayment*									
8														
9		index	-2	-1	0	1	2	3	4	5	6	7	8	*HolidayIndex*
10		repayment mask		FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*Mask*
11														
12		bf		0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	*Bf*
13		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
14		repayment		0	0	125,000	125,000	125,000	125,000	125,000	125,000	125,000	125,000	*Repayment*
15		cf	0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	0	*Cf*
16														

This now holds back the repayments until the end of the repayment holiday (as the mask changes from FALSE to TRUE), but it does not recognise that the loan will eventually get paid off, or could be paid off early. You should set up another index, starting in the base column with =0−Duration and incrementing by 1 each year. Set up a second mask which reads =DurationIndex<1. If you inspect this row it reads TRUE from the start. We should only make repayments when both lines read TRUE, so set up a repayment mask reading: =AND(+HolidayMask,+DurationMask) and amend the repayment calculation in the corkscrew accordingly.

Note that AND and OR are grouped with the likes of MAX and MIN in that they require range names to be prefixed with a plus sign if they are to read from the same column and not the whole row (page 60).

D13 =AND(+HolidayMask, +DurationMask)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		10	*Duration*									
6		repayment holiday		2	*RepaymentHoliday*									
7		annual repayment		125000	*AnnualRepayment*									
8														
9		holiday index	-2	-1	0	1	2	3	4	5	6	7	8	*HolidayIndex*
10		holiday mask		FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*HolidayMask*
11		duration index	10	9	8	7	6	5	4	3	2	1	0	*DurationIndex*
12		duration mask		TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*DurationMask*
13		repayment mask		FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*Mask*
14		bf		0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	*Bf*
15		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
16		repayment		0	0	125,000	125,000	125,000	125,000	125,000	125,000	125,000	125,000	*Repayment*
17		cf	0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	0	*Cf*
18														

Test the operation of the masks by changing both the repayment holiday and duration values.

I mentioned previously that we might not be sure when we will actually draw down the loan, but whenever this happens, we will still be entitled to the repayment holiday.

This issue can be resolved by changing the existing mask. First, we need to identify when the loan is actually drawndown. Set up a row which tests for this: = Amount > 0. This returns TRUE in the year the drawdown takes place. We can create a flag to show that this event has occurred, and we can then replace the existing duration and holiday indices with a single loan repayment index. On the line below the drawdown test put a zero in the base column and then fill the row with =Previous+Above, where Previous is a cell reference to the previous cell in the row and Above is a cell or range reference to the drawdown test line. This row should read 1 from the point at which the drawdown takes place. In the next row, we can set up the loan index using a neat trick to carry out a rolling sum:

=SUM($start of index:start of index).

The first reference in this formula must be absolute, the second part relative. When you copy this formula across the row, you should find that the numbers accumulate. The value of 1 occurs in the year in which the debt is drawndown.

D11 =SUM($D$10:D10)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3		Business loan												
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		10	*Duration*									
6		repayment holiday		2	*RepaymentHoliday*									
7		annual repayment		125000	*AnnualRepayment*									
8														
9		loan drawndown		TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	*DrawdownMask*
10		flag	0	1	1	1	1	1	1	1	1	1	1	*Flag*
11		index		1	2	3	4	5	6	7	8	9	10	*LoanIndex*
12														
13														
14														
15		repayment mask		FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	*Mask*
16		bf		0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	*Bf*
17		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
18		repayment		0	0	125,000	125,000	125,000	125,000	125,000	125,000	125,000	0	*Repayment*
19		cf	0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	125,000	*Cf*
20														

To complete this routine, we need to know when the repayments should start, assuming that the repayment holiday still applies. We also need to identify when the repayments finish, although to get the logic the right way round we should actually identify the period during which the repayments have not finished.

Repayments due =LoanIndex>Holiday
Repayments not finished =LoanIndex<=Duration

D11 =SUM($D$10:D10)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3		Business loan												
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		10	*Duration*									
6		repayment holiday		2	*RepaymentHoliday*									
7		annual repayment		125000	*AnnualRepayment*									
8														
9		loan drawndown		TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	*DrawdownMask*
10		flag	0	1	1	1	1	1	1	1	1	1	1	*Flag*
11		index		1	2	3	4	5	6	7	8	9	10	*LoanIndex*
12														
13		repayment due		FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*RepaymentDue*
14		repayment not finished		TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*RepaymentUnfinished*
15		repayment mask		FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*Mask*
16		bf		0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	*Bf*
17		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
18		repayment		0	0	125,000	125,000	125,000	125,000	125,000	125,000	125,000	125,000	*Repayment*
19		cf	0	1,000,000	1,000,000	875,000	750,000	625,000	500,000	375,000	250,000	125,000	0	*Cf*
20														

This is a tidier solution than the previous version, in that only one index is required. You may find alternative solutions.

Having worked through this explanation of the use of logic techniques in modelling, I would invite you to consider solving the loan exercise using IF.

Lookup

Good old VLOOKUP and HLOOKUP

The lookup table is a well established financial modelling tool. As an organised structure, it contains reference information that can be retrieved for calculation purposes. It allows me, for example, to quote the current share price of a company, or to set the hourly rate for a consultant, or even to run scenarios (see Chapter 6). In the right circumstances it is robust and reliable, but it doesn't lend itself to all types of modelling. The two basic flavours are VLOOKUP and HLOOKUP – with the former the information is arranged in rows, in the latter it is in columns.

The basic syntax of the VLOOKUP is:

=VLOOKUP(item, lookup table, column, match type).

You can enter an item into a cell, the VLOOKUP looks for it in the first column of the lookup table, reads across to the appropriate column, and returns the corresponding entry.

Item is the cell reference or range name containing the search item. The search item must occur in the first (index) column of the lookup table (in HLOOKUP the search item must occur in the first row).

Lookup table is the range in which the information is arranged. This can be cell references or a range name.

Column is the number of the column which contains the item you wish to return. This is normally a value. The left hand column is column 1.

Match type is FALSE or 0 if an exact match is required. If TRUE or 1, Excel will match the lookup item with an item from the range which is less than or equal to the test item, but with the assumption that the first column has been sorted into ascending order. If this argument is omitted, Excel assumes that the match type is TRUE.

We need one VLOOKUP formula for each item of information we wish to obtain. VLOOKUP is not case sensitive in terms of the search item and its corresponding entry in the index column. Look at the following example:

C5 fx =VLOOKUP(C3,C9:H15,2,0)

	A	B	C	D	E	F	G	H	I
2									
3		*Enter Ticker:*	GHI						
4									
5		*Company name*	GHI Inc.						
6		*Current share price*	11						
7									
8			**Ticker**	**CoName**	**Price**	**Hi**	**Lo**	**P/E**	
9			ABC	ABC Ltd	13	23	7	8%	
10			DEF	DEF plc	13	22	11	6%	
11			GHI	GHI Inc.	11	16	9	7%	
12			JKL	JKL Ltd	16	20	12	7%	
13			MNO	MNO plc	12	21	16	8%	
14			PQR	PQR Inc	19	22	13	10%	
15			STU	STU Group plc	15	24	11	9%	
16									

A well-constructed lookup table can be very reliable. However there are problems and limitations. The index column must not contain duplicates; VLOOKUP will only handle the first match. One of the commonest problems is when new data is appended to the table and the formulae are not updated, although a simple workaround for this is to use a range name for the table and to rename the table after updating. As the formula requires you to specify the column from which you want to return the result, a further structural problem occurs if columns are inserted or deleted. The hardcoded column number can also cause problems if the formula is copied, as can the failure to set absolute references for the lookup table – this latter problem is often missed. If the match type argument is omitted, the formula will return an approximate match rather than an exact match, which can be misleading as there will be no indication of this. Only if the item does not appear in the index column will Excel return the #N/A error. Lastly, it should be obvious but lookup does not work in reverse: with my share portfolio above, I can not find out the company name of my lowest priced stock.

It should be apparent that VLOOKUP requires an organised table structure. Our cash flow models do not have such organisation.

Lookup without using LOOKUPs

In the example below, we can see a typical lumpy cash flow and the resulting cash balances of a project. Being prudent financial planners, we wish to avoid going into overdraft and so it would be helpful to flag up cash shortfalls. However, I do not want to keep scrolling across the screen on the off chance that this scenario might happen; it would be more useful if Excel could warn us of the event. Firstly we will identify the lowest cash flow. This is easily done with a MIN: =MIN(CashFlow). We will call this CashFlowMinimum. Next, we need to identify the year in which this occurs. The row of years is called YearIndex.

CashFlowMin... =MIN(CashFlow)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**Financial year ending**				**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**
2															
3	Cash flows														
4		project			743	416	158	580	702	- 158	721	921	743	416	172
5		minimum cash flow		-158	*CashFlowMinimum*										
6															

Let us solve this problem with a simple mask. We can put in a line that tests if the minimum cash value corresponds to the cash balance value for that year.

=CashFlowMinimum=CashFlow Call this row Mask.

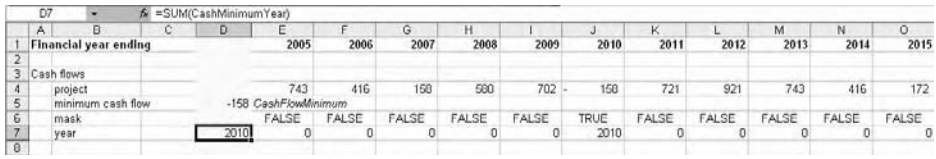
As CashFlowMinimum is the name of a single cell, it is inherently absolute and can be copied across the row. We should see a TRUE value in the year in which this value occurs. Now multiply by the YearIndex

=YearIndex*Mask Call this row CashFlowMinimumYear.

To finish off, put a SUM in the base column:

=SUM(CashFlowMinimumYear).

Change the numbers to prove that this works.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**Financial year ending**				**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**
2															
3	Cash flows														
4		project			743	416	158	580	702	-158	721	921	743	416	172
5		minimum cash flow		-158	*CashFlowMinimum*										
6		mask			FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
7		year		2010	0	0	0	0	0	2010	0	0	0	0	0
8															

MATCH and INDEX

Staying with the problem of looking up information in the financial model, where the values themselves are not organised and could occur anywhere in a given range, we can look at two further functions, MATCH and INDEX, which together provide the lookup functionality we need. To introduce them we will look at them individually.

MATCH

The MATCH function looks for an item in a range and returns its position. It has the following syntax:

=MATCH(item, range, match type)

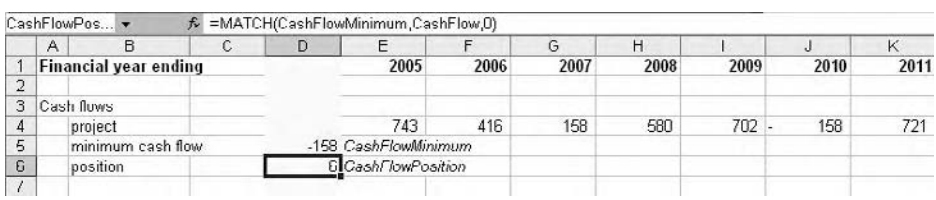
Item is the cell reference or range name of the value we wish to look for.

Range is the row or column where this item can be found.

Match type is FALSE or 0 if an exact match is required. If TRUE or 1, Excel will return an item from the range which is less than or equal to the test item, but with the assumption that the range is in ascending order. If this argument is omitted, Excel assumes that match type is TRUE.

Using the cash flow example above, we can write this formula:

=MATCH(CashFlowMinimum,CashFlow,0) Important – do not miss out the 0. Call it CashFlowPosition.



	A	B	C	D	E	F	G	H	I	J	K
1	**Financial year ending**				**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**
2											
3	Cash flows										
4		project			743	416	158	580	702	-158	721
5		minimum cash flow		-158	*CashFlowMinimum*						
6		position		6	*CashFlowPosition*						
7											

The formula tells us that the lowest cash balance is in position 6. Now we need to find out the corresponding value from the year index row. Although this is not entirely dissimilar to the LOOKUP function, remember that the cash flow line is in no particular order, as it simply reflects the cash flows generated each year.

INDEX

The MATCH function is effectively the first part of a VLOOKUP – it locates an item in a range and returns its position. The second part of VLOOKUP is then to locate the position

in another range and return the corresponding value. This is handled by the INDEX function. In many years of teaching this, I have found that the simplest explanation of INDEX is:

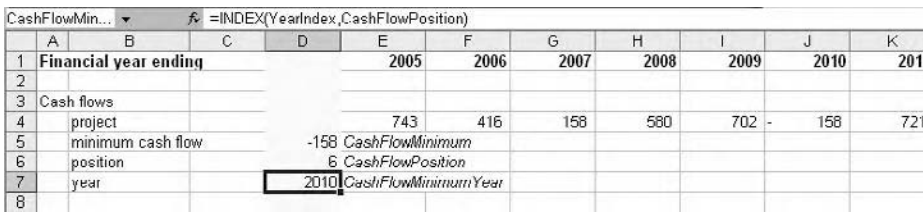
=INDEX(range, position)

Range is the row (or column) you want to examine.

Position is a value corresponding to the location of the value to be returned. Position must be a positive, whole number. Position 1 is always the left hand cell in a row, or the top cell in a column.

Returning to our minimum cash flow example, the MATCH function tells us that the lowest value is in position 6 of the cash balance row. We now use the INDEX to return the corresponding year.

=INDEX(YearIndex,CashFlowPosition)



	A	B	C	D	E	F	G	H	I	J	K
1	**Financial year ending**				**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**
2											
3	Cash flows										
4		project			743	416	158	580	702	- 158	721
5		minimum cash flow		-158	*CashFlowMinimum*						
6		position		6	*CashFlowPosition*						
7		year		2010	*CashFlowMinimumYear*						
8											

If you feel confident you could omit the first step and simply nest the two formulae:

=INDEX(YearIndex,MATCH(CashFlowMinimum,CashFlow,0))

Again, do not forget the zero.

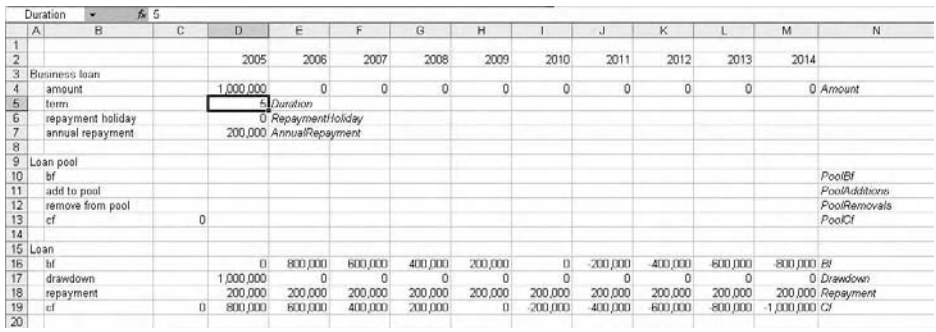
Straightaway there are a couple of caveats to consider. First, we must recognise that the search item is unique. Given that even in a formatted cell Excel can calculate to fifteen decimal places, it may be considered unlikely that two cash balances are exactly the same – but it is not impossible. Second, the MATCH range must have a one-for-one correspondence with the INDEX range – there is no value to the exercise if they are of different sizes or are in a different order.

Does the apparent complexity of the nested INDEX … MATCH offer any benefits over the simple mask method that we considered earlier?

The concept of the pool

INDEX, however, can offer a great deal of functionality in its own right. We can explore this in the context of the loan repayment schedule we looked at in the previous section. For the moment we'll take out the repayment holiday, and delete the loan and repayment

mask elements. We'll borrow the money for five years. Set up new lines as shown, ignoring the negatives:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		5	*Duration*									
6		repayment holiday		0	*RepaymentHoliday*									
7		annual repayment		200,000	*AnnualRepayment*									
8														
9	Loan pool													
10		bf												*PoolBf*
11		add to pool												*PoolAdditions*
12		remove from pool												*PoolRemovals*
13		cf	0											*PoolCf*
14														
15	Loan													
16		bf		0	800,000	600,000	400,000	200,000	0	-200,000	-400,000	-600,000	-800,000	*Bf*
17		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
18		repayment		200,000	200,000	200,000	200,000	200,000	200,000	200,000	200,000	200,000	200,000	*Repayment*
19		cf	0	800,000	600,000	400,000	200,000	0	-200,000	-400,000	-600,000	-800,000	-1,000,000	*Cf*
20														

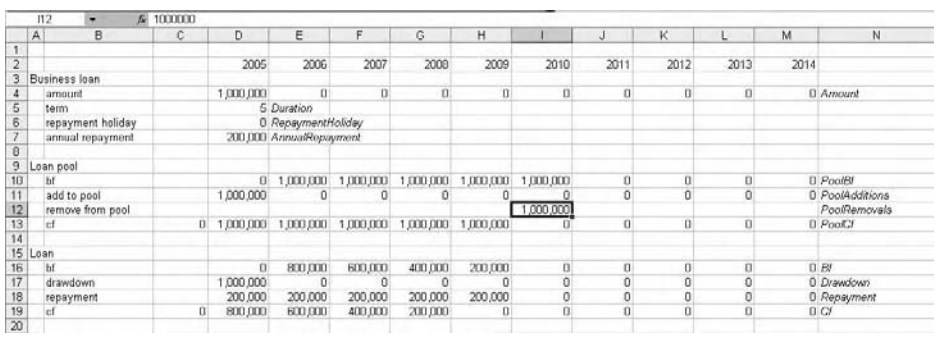
The concept of the pool is that we will add in the debt as it is drawdown. Once this has occurred, repayments will start and continue until the debt is paid off. The timing and duration of the loan are still variables.

The corkscrew is straightforward

Opening balance bf =previous closing balance(cell reference)
Add to Pool =Amount
Remove from Pool let us leave for a moment
Closing balance cf =PoolBf+PoolAdditions−PoolRemovals

Delete the existing debt repayment formulae from the debt corkscrew. Repayment is now the PoolCf/Duration. Do not worry about the overpayment that now follows.

The PoolRemovals line should show the debt dropping out after Duration number of years. As the closing balance reaches zero, the repayment formula also reads zero. Prove this by typing in the loan amount in the appropriate cell. Note the effect on the repayments.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		5	*Duration*									
6		repayment holiday		0	*RepaymentHoliday*									
7		annual repayment		200,000	*AnnualRepayment*									
8														
9	Loan pool													
10		bf		0	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	0	0	0	0	*PoolBf*
11		add to pool		1,000,000	0	0	0	0	0	0	0	0	0	*PoolAdditions*
12		remove from pool							1,000,000					*PoolRemovals*
13		cf	0	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	0	0	0	0	0	*PoolCf*
14														
15	Loan													
16		bf		0	800,000	600,000	400,000	200,000	0	0	0	0	0	*Bf*
17		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
18		repayment		200,000	200,000	200,000	200,000	200,000	0	0	0	0	0	*Repayment*
19		cf	0	800,000	600,000	400,000	200,000	0	0	0	0	0	0	*Cf*
20														

Of course we need a calculation for this. Think back to the description of the INDEX function: it looks in a range at a specified position. The range we are interested in is the

PoolAdditions line, and specifically position 1. We want the loan amount to appear in this cell. We could write =INDEX(PoolAdditions, 1) but that would be hardcoded. As with the mask treatment, the 1 and related values should be set up above the pool as a separate line. In the base column, type in =0–Duration. In the rest of the row type =previous+1, where previous is a cell reference to the adjacent cell. This should result in a number sequence in which the value of 1 appears now in the required column. Name this row as PoolIndex.

C10 =0-Duration

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
9	Loan pool													
10		index	-5	-4	-3	-2	-1	0	1	2	3	4	5	PoolIndex
11		bf	0	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	0	0	0	0	PoolBf
12		add to pool		1,000,000	0	0	0	0	0	0	0	0	0	PoolAdditions
13		remove from pool							1,000,000					PoolRemovals
14		cf	0	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	0	0	0	0	0	PoolCf
15														

Now we can write the PoolRemovals formula as:

=INDEX(PoolAdditions,PoolIndex)

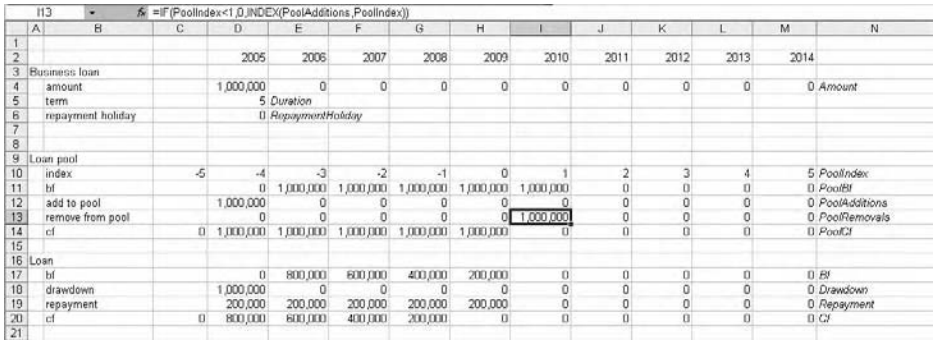
Copy the formula to the other cells in the row. Remember that the position value for INDEX must be a positive whole number, so the negative and zero index values generate #VALUE! errors. We can protect against this by rewriting the formula once more:

=IF(PoolIndex<1,0,INDEX(PoolAdditions,PoolIndex))

That is, if the PoolIndex is not a positive whole number, return zero.

Now, having proved that it works, let us find out why. It should be clear that the loan drops out where it does because the instruction in the INDEX formula is to look in the PoolAdditions row at the position specified in the PoolIndex row, in this case a 1, so return the value from position 1 in the PoolAdditions range. The next index value is 2, and the instruction in the formula is to look in the PoolAdditions range at position 2, and so on. We can test that the mechanism works if you delete the debt in year 1 and instead put the debt in year 2. Now change the loan duration and prove that the repayments are correct. Finally, split the loan so that the first tranche is drawndown in the first year and the remainder in the second. Does this work?

If you are still struggling with this you are in good company. Having taught this technique for several years I still find that this is the most frequent post-course enquiry I receive, which either suggests that I am not very good at teaching it, or that people have found it useful but did not quite grasp the issues the first time around. The usual question picks up on the formula's structure, in that the range element is an absolute reference (so make sure you have the $ signs if you are using cell references), and that the position element is a relative reference. It doesn't matter where you are in relation to the PoolAdditions line, but the position value is taken from the PoolIndex in the same column. Also, the PoolIndex is the driver of this mechanism – whatever value is used as the duration, the PoolIndex recalculates. The INDEX function simply does what it is told.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	
3	Business loan													
4		amount		1,000,000	0	0	0	0	0	0	0	0	0	*Amount*
5		term		5	*Duration*									
6		repayment holiday		0	*RepaymentHoliday*									
7														
8														
9	Loan pool													
10		index	-5	-4	-3	-2	-1	0	1	2	3	4	5	*PoolIndex*
11		bf		0	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	0	0	0	0	*PoolBf*
12		add to pool		1,000,000	0	0	0	0	0	0	0	0	0	*PoolAdditions*
13		remove from pool		0	0	0	0	0	1,000,000	0	0	0	0	*PoolRemovals*
14		cf	0	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	0	0	0	0	0	*PoolCf*
15														
16	Loan													
17		bf		0	800,000	600,000	400,000	200,000	0	0	0	0	0	*Bf*
18		drawdown		1,000,000	0	0	0	0	0	0	0	0	0	*Drawdown*
19		repayment		200,000	200,000	200,000	200,000	200,000	0	0	0	0	0	*Repayment*
20		cf	0	800,000	600,000	400,000	200,000	0	0	0	0	0	0	*Cf*
21														

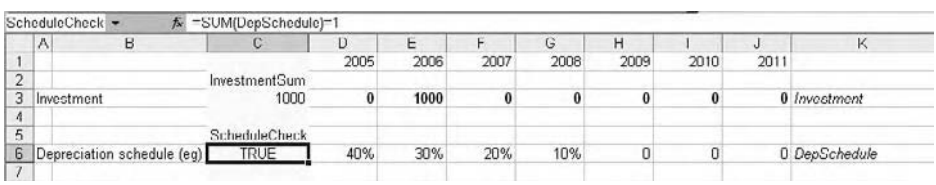
If you want to extend this repayment schedule to look at semi-annual repayments rather than annual, the rule is that the model should be set up according to the shortest time period it will use. Instead of the PoolIndex values representing whole years, they will now represent periods and you can factor this in accordingly. Likewise, if you want to include the repayment holiday you can set up the mask we used previously. I will leave you to do this.

This INDEX and pool mechanism is very useful in modelling. There are many examples when a pool comes in handy, such as straight line depreciation. In this case you can set up a pool that picks up the capital expenditure as it is incurred and holds it for the life of the asset. While it is held in the pool it is available for the depreciation calculation, which can be as simple as PoolCf/AssetLife. As seen above, this pool can handle any number of assets, provided they share the same life. I could therefore have one depreciation pool for my five year assets, another for my 10 year assets, and so on.

Other examples include corporate bonds, such that whilst the bond is in issue we pay a coupon (interest). Or venture capital, which has the features of debt to begin with and therefore drives an interest calculation, but at some stage we might want to convert to share capital and add the amount to the share capital corkscrew. We could also consider the retirement of tax losses, in which a company incurs a tax loss and can carry it forward over time to set against future profits. This period is defined by the tax authorities and is not unlimited.

Fixed schedules

The INDEX and mask combination also allows to handle issues such as accelerated depreciation (or capital allowances in the UK). The problem is that the values to use for the depreciation charge differ each year, with higher charges in the first years. If we are not sure when the capital expenditure will take place it is often difficult to apply the correct rate for the year/column. Look at the following:



	A	B	C	D	E	F	G	H	I	J	K
1				2005	2006	2007	2008	2009	2010	2011	
2			InvestmentSum								
3		Investment	1000	**0**	**1000**	**0**	**0**	**0**	**0**	**0**	*Investment*
4											
5			ScheduleCheck								
6		Depreciation schedule (eg)	TRUE	40%	30%	20%	10%	0	0	0	*DepSchedule*
7											

I have put a SUM in the base column at the start of the Investment line. The DepSchedule row lists the annual depreciation charges and I have included additional zero values to complete the forecast period. Make sure this row adds up to 100% by putting a logical check in the base column.

We then add a mechanism to count the number of charges in the depreciation schedule, in case they are varied. This is expressed as =(DepSchedule>0)+0, with a SUM in the base column.

Next we need a flag to identify that the investment has occurred. The Investment Happened line captures the capex event =Investment>0, and the line below, Coercion, converts this to a value.

D10 =InvestmentHappened+0

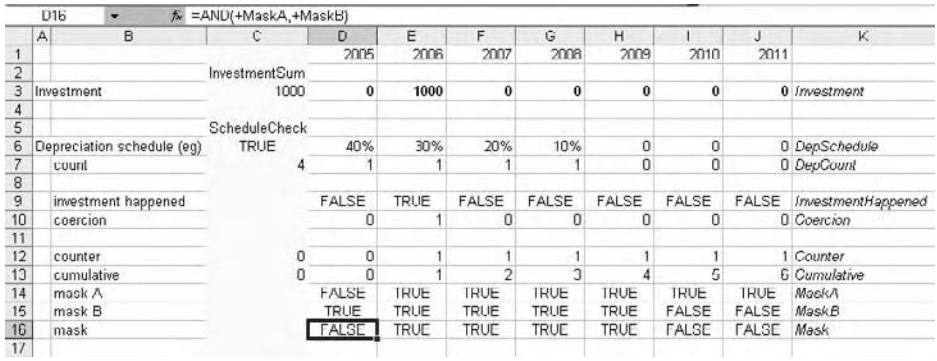
	A	B	C	D	E	F	G	H	I	J	K
1				2005	2006	2007	2008	2009	2010	2011	
2			InvestmentSum								
3		Investment	1000	0	1000	0	0	0	0	0	*Investment*
4											
5			ScheduleCheck								
6		Depreciation schedule (eg)	TRUE	40%	30%	20%	10%	0	0	0	*DepSchedule*
7		count	4	1	1	1	1	0	0	0	*DepCount*
8											
9		investment happened		FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	*InvestmentHappened*
10		coercion		0	1	0	0	0	0	0	*Coercion*
11											

We now need to create a number sequence which counts off from the event of the investment. This number sequence will be used in an INDEX formula very shortly. The Counter row recognises the InvestmentHappened event (=previous+Coercion), and the Cumulative row creates the number sequence (=previous+Counter), with appropriate zero values in the base column.

D13 =C13+Counter

	A	B	C	D	E	F	G	H	I	J	K
1				2005	2006	2007	2008	2009	2010	2011	
2			InvestmentSum								
3		Investment	1000	0	1000	0	0	0	0	0	*Investment*
4											
5			ScheduleCheck								
6		Depreciation schedule (eg)	TRUE	40%	30%	20%	10%	0	0	0	*DepSchedule*
7		count	4	1	1	1	1	0	0	0	*DepCount*
8											
9		investment happened		FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	*InvestmentHappened*
10		coercion		0	1	0	0	0	0	0	*Coercion*
11											
12		counter	0	0	1	1	1	1	1	1	*Counter*
13		cumulative	0	0	1	2	3	4	5	6	*Cumulative*

We need a mask which will identify that the investment has taken place and is ready for depreciation, but we will need another to ensure that we only apply the depreciation for the correct number of periods. Mask A tests that Cumulative>0, and Mask B tests that the Cumulative<=Count (the count of the number of depreciation charges in the schedule). The final Mask returns TRUE only when both Mask A and Mask B are TRUE.



D16 =AND(+MaskA,+MaskB)

	A	B	C	D	E	F	G	H	I	J	K
1				2005	2006	2007	2008	2009	2010	2011	
2			InvestmentSum								
3		Investment	1000	0	1000	0	0	0	0	0	*Investment*
4											
5			ScheduleCheck								
6		Depreciation schedule (eg)	TRUE	40%	30%	20%	10%	0	0	0	*DepSchedule*
7		count	4	1	1	1	1	0	0	0	*DepCount*
8											
9		investment happened		FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	*InvestmentHappened*
10		coercion		0	1	0	0	0	0	0	*Coercion*
11											
12		counter	0	0	1	1	1	1	1	1	*Counter*
13		cumulative	0	0	1	2	3	4	5	6	*Cumulative*
14		mask A		FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*MaskA*
15		mask B		TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	*MaskB*
16		mask		FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	*Mask*
17											

Finally we can set up the depreciation corkscrew. The depreciation rate row uses an INDEX formula to align the appropriate rate with the investment event:

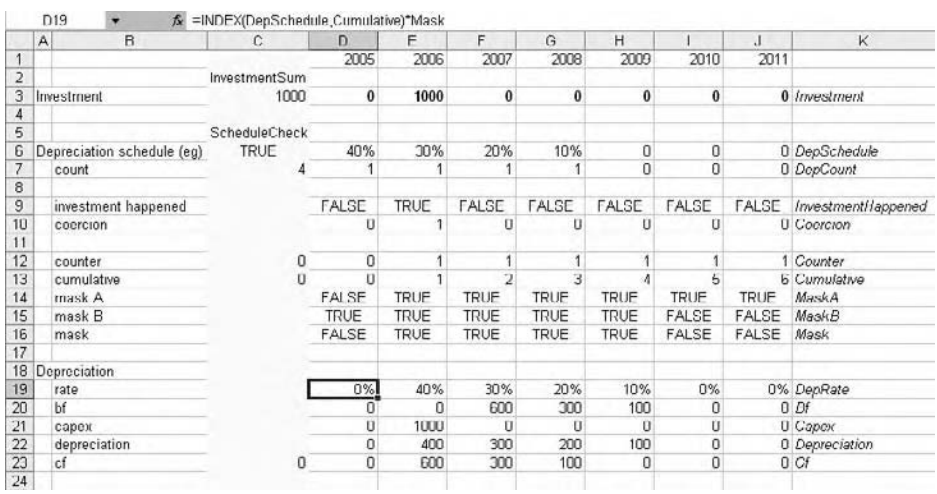
```
=INDEX(DepSchedule,Cumulative)*Mask
```

DepSchedule is the set of rates in the depreciation schedule, Cumulative is the number sequence beginning in the investment year, and Mask ensures that the formula only works in the number of years specified in the depreciation schedule.

The rest of the corkscrew is straightforward. The depreciation formula is:

```
=InvestmentSum*DepRate
```

Test that this routine works by changing the timing of the investment, or the depreciation schedule. Don't push it too far, because it will not handle more than one investment, but it is otherwise very robust. Think about creating an error trap based on ScheduleCheck in case the depreciation rates do not add up to 100%.



D19 =INDEX(DepSchedule,Cumulative)*Mask

	A	B	C	D	E	F	G	H	I	J	K
1				2005	2006	2007	2008	2009	2010	2011	
2			InvestmentSum								
3		Investment	1000	0	1000	0	0	0	0	0	*Investment*
4											
5			ScheduleCheck								
6		Depreciation schedule (eg)	TRUE	40%	30%	20%	10%	0	0	0	*DepSchedule*
7		count	4	1	1	1	1	0	0	0	*DepCount*
8											
9		investment happened		FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	*InvestmentHappened*
10		coercion		0	1	0	0	0	0	0	*Coercion*
11											
12		counter	0	0	1	1	1	1	1	1	*Counter*
13		cumulative	0	0	1	2	3	4	5	6	*Cumulative*
14		mask A		FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	*MaskA*
15		mask B		TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	*MaskB*
16		mask		FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	*Mask*
17											
18		Depreciation									
19		rate		0%	40%	30%	20%	10%	0%	0%	*DepRate*
20		bf		0	0	600	300	100	0	0	*Df*
21		capex		0	1000	0	0	0	0	0	*Capex*
22		depreciation		0	400	300	200	100	0	0	*Depreciation*
23		cf	0	0	600	300	100	0	0	0	*Cf*
24											

OFFSET

Before moving on from the lookup functions, we will take a quick look at the OFFSET function. OFFSET is one of those functions like IF, which tend to be overused and abused. In simple terms it has the following syntax:

=OFFSET(starting point, rows, columns)

Starting point is the cell from which you wish to navigate. It is not necessarily the current cell. If you are using cell references you should decide if this reference should be absolute or relative.

Rows is a value or cell reference to a cell that contains a value that represents the number of rows you wish to read down (positive value) or up (negative value) from the starting point.

Columns is a value or cell reference to a cell that contains a value that represents the number of columns you wish to read to the right (positive value) or left (negative value) of the starting point.

The immediate point to note is that this is an indirect referencing technique: in effect we are giving Excel directions to a cell, rather than addressing it specifically.

If we look back at the loan pool removals routine in the previous exercise, it reads

=IF(PoolIndex<1,0,INDEX(PoolAdditions,PoolIndex))

INDEX asks Excel to read across a row to a particular position. We can achieve the same result using OFFSET. We need to type a zero in the base column on the PoolAdditions row, which will act as our starting point. Call it PoolAdditionsBase.

=IF(PoolIndex<1,0,OFFSET(PoolAdditionsBase,,PoolIndex))

The additional comma is required because in this example the row argument is not required.

Excel interprets this as the instruction to start at the PoolAdditionsBase cell and read across (blank rows and) the number of columns specified by the PoolIndex. Test to see if it works.

D13 =IF(PoolIndex<1,0,OFFSET(PoolAdditionsBase,,PoolIndex))

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
9	Loan pool													
10		index	-5	-4	-3	-2	-1	0	1	2	3	4	5	*PoolIndex*
11		bf	PoolAdditi	0	1,000,000	1,500,000	1,500,000	1,500,000	1,500,000	500,000	0	0	0	*PoolBf*
12		add to pool	0	1,000,000	500,000	0	0	0	0	0	0	0	0	*PoolAdditions*
13		remove from pool		0	0	0	0	0	1,000,000	500,000	0	0	0	*PoolRemovals*
14		cf	0	1,000,000	1,500,000	1,500,000	1,500,000	1,500,000	500,000	0	0	0	0	*PoolCf*
15														

Make sure that the OFFSET formula is suitably covered with the error trap which will prevent from reading off the edge of the spreadsheet or from returning the text in the row heading. I do not think that the resulting formula is an improvement on the original IF … INDEX.

We often find that people hard code the row and column references into their OFFSETS, and as a model develops and routines are moved around, there always remains a nagging doubt that it might not be reading what it should. Because of the indirect nature of the referencing, you will find that the usual audit techniques (F2, Ctrl+[) point to the start cell

and the cells containing the row and column values, but not to the target cell. Try it with the depreciation schedule OFFSET.

Financial

In a book about financial modelling it might be expected that we would look at the Excel financial functions in some detail. But we won't, other than to recognise that these functions do not always work in the same way that you might have been taught at business school or in the textbooks. For example, the concept of Net Present Value is based on the value of the cash flows into and out of an investment, discounted by the cost of capital or hurdle rate. If you have never looked at Excel's Help topic for NPV, you might assume we could enter the investment amount and list the expected cash flows, and then discount to their present values. But we must be careful about the timing of the investment and the returns: Excel assumes that the investment takes place in time 1, rather than the present time 0 (so it is really a future NPV). If the first cash flow (usually the investment) takes place in time zero the formula should be written:

=NPV(DiscountRate, CashFlow 1 … CashFlow n)+CashFlow 0

This can be illustrated using the following example. The investment (outflow) of 300 occurs in the first period, followed by three returns (inflows) of 100. The interest rate is 10%. The Excel NPV function gives a value of (46.65). If we calculate this arithmetically, we set up an additional row to show the period number 0–3, and write the following discount formula:

=CashFlow/(1+rate)^period

The SUM of this row is (51.31), a difference of 4.66.

D11 fx =CashFlow/(1+Rate)^Period

	A	B	C	D	E	F	G	H	I
1	Period			0	1	2	3	*Period*	
2									
3	Cash flows			(300.00)	100.00	100.00	100.00	*CashFlow*	
4									
5	Interest/discount rate			10%	*Rate*				
6									
7	Net present values								
8		NPV function		**(46.65)**					
9									
10									
11		discounted cash flows		(300.00)	90.91	82.64	75.13	*DCF*	
12									
13		sum of DCFs		**(51.31)**					
14									

Change the period row to 1–4 and you will see that the Excel NPV and the arithmetical NPV now agree.

D11 =CashFlow/(1+Rate)^Period

	A	B	C	D	E	F	G	H	I
1	Period			1	2	3	4	*Period*	
2									
3	Cash flows			(300.00)	100.00	100.00	100.00	*CashFlow*	
4									
5	Interest/discount rate			10%	*Rate*				
6									
7	Net present values								
8		NPV function		**(46.65)**					
9									
10									
11		discounted cash flows		(272.73)	82.64	75.13	68.30	*DCF*	
12									

To make Excel agree with the time zero investment NPV, rewrite the function as:

=NPV(Rate,E3:G3)+D3

A further refinement is to recognise that cash flows are normally treated as year end flows. In some circumstances we may need to consider them as mid year cash flows, and this is easily done by changing the period row:

D11 =D3/(1+Rate)^Period

	A	B	C	D	E	F	G	H
1	Period			0.5	1.5	2.5	3.5	*Period*
2								
3	Cash flows			(300.00)	100.00	100.00	100.00	*CashFlow*
4								
5	Interest/discount rate			0.10	*Rate*			
6								
7	Net present values							
8		NPV function		**(51.31)**				
9								
10								
11		discounted cash flows		(286.04)	86.68	78.80	71.64	*DCF*
12								
13		sum of DCFs		**(48.93)**				
14								

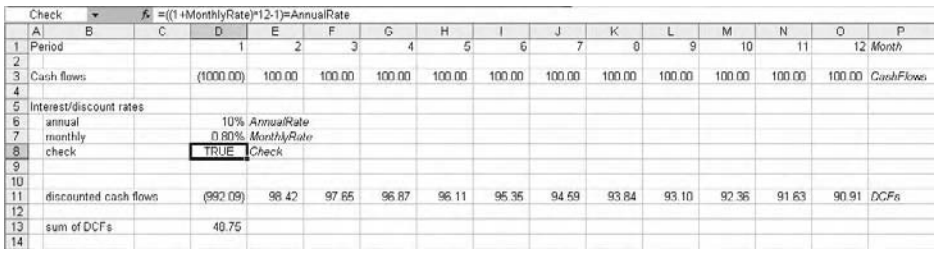
Excel's NPV will not recognise this, although you could use the XNPV function if you have the Analysis ToolPak add-in installed.

A common source of confusion arises when calculating the quarterly or monthly NPV and failing to amend the discount rate appropriately. Remember that if we are calculating simple interest we can divide the annual rate by the number of periods, but if using compound interest we should use the following formula:

=(1+annual rate)^(1/period)–1

You can use this formula as an approximate audit check:

=(1+annual rate)=(1+monthly rate)^12−1



Check	=((1+MonthlyRate)^12-1)=AnnualRate													
	A B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Period		1	2	3	4	5	6	7	8	9	10	11	12	*Month*
2															
3	Cash flows		(1000.00)	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	*CashFlows*
4															
5	Interest/discount rates														
6	annual		10%	*AnnualRate*											
7	monthly		0.80%	*MonthlyRate*											
8	check		TRUE	*Check*											
9															
10															
11	discounted cash flows		(992.09)	98.42	97.65	96.87	96.11	95.35	94.59	93.84	93.10	92.36	91.63	90.91	*DCFs*
12															
13	sum of DCFs		40.75												
14															

The reality is that most financial institutions require their analysts to write the full arithmetical treatment for these and other financial calculations, where possible. We all know the basic compounding and discounting formulae and therefore their appearance in the worksheet is easy to check.

The exception would be the internal rate of return function. If you have covered IRR at business school or in your finance studies, you will be aware that we use an iterative technique to calculate the IRR and we can safely use the Excel IRR function. The IRR function uses a default guess of the interest rate at 10%. If the cash flow does not begin with the investment as a negative amount, Excel gives up on the iteration and returns the #NUM! error. Also, you should know that if the cash flow changes signs more than once, there will be another IRR – Excel will not return this unless you change the guess:

=IRR(CashFlow, myguess%)

As with NPV, if the cash flows are irregular you can use the XIRR function if it is available on your PC.

Dates

Calculations involving dates are fairly straightforward if we understand that Excel will read dates as numbers, provided they are in written in a valid date format. We can prove this quite simply by entering a date such as 25/5/05 (for the UK; use 5/25/05 in the US or your local date convention). The date appears in the cell and should be right aligned, which shows that Excel is treating it as a number (if it is left aligned Excel is reading it as text and we will be unable to use the date for calculation purposes). Strip the formatting from the cell using Ctrl+Shift+~ or Ctrl+Shift+!. In this example we can see a date serial number of 38,497, which is the number of days which have elapsed since 1st January 1900 (unless you are an Apple Mac user, in which case day 1 was 2nd January 1904. If this is an issue check under Tools, Options, Calculation and the 1904 date system check box). Reformat the cell using Ctrl+# (default date format). If we then enter another date such as 25/12/2005 we can calculate the number of days between the two dates. Note that the dates do not need to be in the same format.

NOW()

The NOW() function can be entered into the worksheet to return the current date and time according to the computer's internal clock. The result is subject to the normal recalculation rules, so that it will only update when a recalculation occurs, through editing or writing formulae or F9 manual recalculation. If you take off the date formatting, we see that the function is calculated to several decimal places, which implies that Excel is working to a split second level of accuracy. Because of this I do not normally use NOW() for routine date calculations, as the results change very slightly during the course of a day. It is useful as a time stamp, but we can achieve this just as easily in the page header or footer settings. A variation on the theme is to use the TODAY() function, which returns the date serial number without the decimals. Both techniques assume your PC's clock is correct.

A neat trick to insert the current date into the workbook is Ctrl+; (semicolon). This is a hardcoded value and will not update.

DATE(year, month, day)

Sometimes we find that the information about the date of an event is already located in the model but not in valid date format. For example, we may know that the company has a financial year end of 31st March, but the years themselves are entered as column headings across the forecast period. We can use the DATE function to assemble this information into a valid date value.

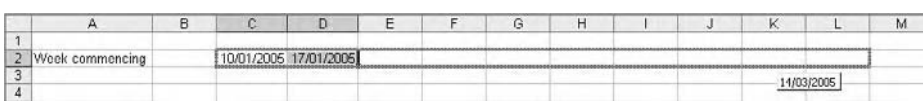
1 Name the row of years on the workings sheet as FYEYears. (FYE = financial year end).
2 Enter the day and month of the year end on the inputs sheet, and name them as FYEDayIn and FYEMonthIn.
3 Link the day and month to the workings sheet and name accordingly.
4 Write the DATE function as:

```
=DATE(FYEYears, FYEMonth, FYEDay)
```

5 When the formula is copied across the row, we should see valid year end dates generated for each year.

Date series

While we are on the subject of dates, we might want to remind ourselves that Excel can help set up sensible date series as, for example, column headings. If we require a row of dates for the 'week commencing', we can type in the first two dates of the sequence and use the AutoFill method to copy the series across the row – select both cells, click and drag the little box (AutoFill handle) at the bottom right corner of the active range.



We can use the same functionality to put in month ends, by typing in the date of the first month end and this time right-click and dragging the AutoFill handle. At the end of the operation Excel displays a shortcut menu from which we can select Fill Months. If you have entered a month end date (31/5/05) Excel is intelligent enough to recognise this and will give the appropriate month ends, rather than generating spurious dates such as 31/06/05.

Other useful functions

LARGE/SMALL

I rather like the LARGE and SMALL functions primarily because they offer functionality that a lot of modellers don't know about. We have looked at MAX and MIN and recognised that they return the largest and smallest values from a range, respectively. But what about other values? Look back at the minimum cash balance routine on page 98. We briefly considered that it might be that there might be two or more years in which this balance occurred. We can pick out these values using SMALL, the syntax for which is:

=SMALL(range, value)

A value of 1 is the first smallest value (=MIN), 2 is the second smallest, and so on. You can work out how LARGE works. Unfortunately SMALL and LARGE do not help with duplicate values, because if there are two years in which the cash balance is zero, SMALL will only identify the value itself. However, as previously noted, cash flow calculations are calculated to several decimal places and duplicates may be unlikely to occur.

I once used LARGE to set up a dynamic sorting system. We needed to show the exposure to a number of liabilities the value of which changed frequently. I should point out that Data, Sort was not an option. We used LARGE, MATCH and INDEX to solve the problem. Don't even think of using RANK.

	A	B	C	D	E	F	G	H	I
1									
2		**Live data**					**Sorted data**		
3		*Item*	*Value*			*Order*	LARGE	MATCH	INDEX
4		A	-46			1	=LARGE(Value,Order)	=MATCH(G4,Value,0)	=INDEX(Item,H4)
5		B	-24			2	-3	8	H
6		C	-43			3	-11	12	L
7		D	-42			4	-20	10	J
8		E	-22			5	-22	5	E
9		F	-37			6	-24	2	B
10		G	-3			7	-37	6	F
11		H	3			8	42	4	D
12		I	-44			9	-43	3	C
13		J	-20			10	-44	9	I
14		K	-45			11	-45	11	K
15		L	-11			12	-46	1	A
16									

We can combine SMALL with COUNTIF and COUNT to tackle a common modelling problem. In this example we will consider a series of annual debt service cover ratios (DSCR) over a long forecast period, during which the debt will be fully paid off. For financial

management purposes we need to identify any periods in which the DSCR falls to critical levels, that is potentially in breach of the loan covenant. Simply put, we need the lowest DSCR. Unfortunately, because of the zero values which follow the repayment of the loan, we cannot use the MIN function. Instead we can use the COUNTIF function to tell us how many values above zero are in the DSCR row.

=COUNTIF(DSCR,“>0”)

This then returns a value of 8. We can use a plain COUNT (DSCR) function to tell us that there are 30 cells in the row. If we then subtract the 8 from the 30 we see that there are 22 cells which contain zero values. The MIN of this range would be zero, so we would like to know the next biggest number after zero, so the following formula would do the trick:

=SMALL(DSCR,COUNT(DSCR)–COUNTIF(DSCR,“>1”)+1)

I would prefer to break the above logic into four separate cells to reduce the risk of error.

DSCRSmallest =SMALL(DSCR,DSCRBlank+1)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2				Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	Year 7	Year 8	Year 9	Year 10	Year 11	Year 12	Year 13	Year 14
3																	
4	DSCR			1.42	1.25	1.66	1.35	1.47	1.53	1.19	1.55	0	0	0	0	0	0
5																	
6		DSCR greater than zero		8	*DSCROverZero*												
7		Count of DSCRs		30	*DSCRCount*												
8		Blank DSCRs		22	*DSCRBlank*												
9		**Smallest DSCR**		**1.19**	***DSCRSmallest***												
10																	

RAND

I often find myself having to create sequences of numbers in order to demonstrate or test a routine, and this can be rather time consuming. The RAND function generates random numbers between 0 and 1 at up to 15 decimal places, which in itself is moderately useful. You can multiply by 10 or 100 to generate more meaningful numbers, but this usually results in a wide range of values. To generate numbers within a particular range, say between 10 and 20, try the following:

=RAND()*(20–10)+10

This formula generates a random number between 0 and 1 and multiplies it by 10, and adds 10 to the result. The value therefore cannot be below 10 or above 20. Substitute in your own values.

The random numbers change every time Excel recalculates, so you may want to convert the formulae to values by using Copy and Edit, Paste Special, Paste Values.

To really crank this up, select the range first, type the RAND formula and press Ctrl+Enter. Then press Ctrl+Shift+1 to format to two decimals, followed by Ctrl+C to copy, and Shift+F10, S, V, Enter, to paste values.

ISTEXT

Cell comments (Shift+F2) are a useful means for storing additional information in the spreadsheet. However, under Tools, Options, View, Comments they can be disabled and

the user may not realise they are there. In the old days of Lotus 1–2–3, we were able to write comments directly into formulae, and we can still do it using Excel. The trick here uses the ISTEXT function. This is one of the IS family of functions which return a logical TRUE or FALSE depending on the test. The syntax is simple:

=ISTEXT("text string") returns TRUE
=ISTEXT(calculation) returns FALSE

As we saw previously, TRUE has a value of 1, FALSE of 0. We can therefore write an annotated formula such as:

=NPV(DiscountRate, CashFlow)*ISTEXT("DiscountRate is the company's real terms cost of capital, and CashFlow is the project cash flow in real terms")

In this case, the ISTEXT formula multiplies the NPV by 1.

CONCATENATION

Concatenation is the method of linking together a series of text strings, cell references and/or values. It can be used for reporting results, as in the following:

="The NPV of the project is" & NPVResult & "using a" & DiscountRate & "discount rate"

The & ampersand character is referred to as the concatenation operator and each element of the formula needs to be prefixed with this symbol. Note that each text string must be enclosed in quotes, and spaces should be included for readability.

Alternatively, we can use the Excel CONCATENATE function:

=CONCATENATE("The NPV of the project is", NPVResult, "using a", DiscountRate, "discount rate")

Needless to say, concatenation should be thought of as generating a text string and so should not be used for calculations. This is not strictly true, as you can prove using

=CONCATENATE(1, 2, 3)*10

TEXT

The concatenation example above does not actually look very tidy when complete, because Excel loses the number formatting of the original cells. To apply the number format within the concatenation formula, we can use the TEXT function, which has the following syntax:

=TEXT(number, "format text")

Number is the value or cell reference containing a value to be formatted; 'Format text' is one of Excel's number formats written in quotes (see Chapter 5)

With the NPV example above, I would like to format the NPV value in £000s, with no decimal places, and the cost of capital as percentage, two decimals. Assuming that we are not going to use the usual formatting techniques, we could write the individual formulae as:

=TEXT(NPVResult, "#,##0"); and
=TEXT(DiscountRate, "0.00%")

Combining this with the concatenation routine we get:

=CONCATENATE("The NPV of the project is", TEXT(NPVResult, "#,##0"), "using a", TEXT(DiscountRate, "0.00%"), "discount rate")

Is this worth the effort? And just in case it isn't obvious, the result of the formula is now text and cannot be used for subsequent calculations.

INT and MOD

INT or integer rounds a number *down* to the nearest integer, or whole number*. MOD returns the modulus or remainder when one number is divided by another. When I am not teaching or writing about financial modelling I like to run marathons, and I wrote a neat little marathon calculator to help me work out my predicted time when running at a particular rate.

If I run an 8-min mile (apologies for the non-metric units), it would take me 26.2* 8 = 209.6 min to run the full marathon, which I would like to express in hours and minutes.

1 Use the INT function to divide 209.6 by 60, which is 3 (hours).
=INT(TotalMinutes/60).

2 Use the MOD function to return the remainder, which is 30 (minutes).
=MOD(TotalMinutes, 60).
Note that we do not need the division operator in the MOD function.

3 To show off, we can concatenate the two results together and format the output by writing:

=CONCATENATE(Hours,":",TEXT(Minutes,"#,#00"))

This gives the result 3:30

* There is also the ROUND(value, number of places) function, which will round a number up or down to the specified number of decimal places. You might also look at the Excel ROUNDUP and ROUNDDOWN functions.

CHAPTER 5

Model use

Introduction

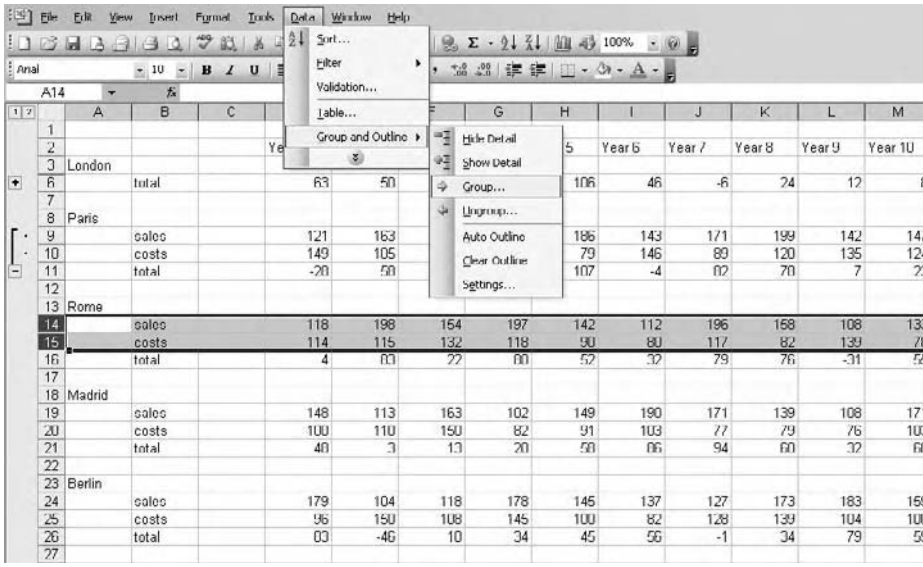
The test of a good model is ultimately how the users respond to it. Back in Chapter 1 we recognised the need to engage with the users at the outset to find out exactly what their requirements are, and this may involve both discussion and iteration to refine the purpose and function of the model. We must also ensure that expectations are realistic and that the limitations of the model are known. The users must be able to comprehend how the model works but without getting overwhelmed with detail. As model developers we may be quite confident about examining a workings sheet that contains several hundred rows of code, or perusing the structure of the pro forma financial statements covered with telephone number-sized figures, but our users may not feel quite the same way. Have you ever watched someone looking at a model for the first time? See how long it is before they check their email, or their mobile phone, or fetch a glass of water, or engage in any other form of displacement activity before actually getting down to the task at hand.

The demonstration workbooks for this Chapter are in Chapter 5: Model Use folder on the CD-ROM.

Grouping and outlining

Even a relatively simple model may contain many rows of detail which serve to distract or confuse (or intimidate) the user. Although it is possible to simply hide the appropriate rows, it is more helpful to allow the user to hide or expand the detail as required through the use of grouping and outlining. For a model that has been laid out in a sensible and meaningful way this is fairly straightforward. I shall refer to rows but the same principles apply equally to columns.

1 Select the rows to be grouped. You should decide if the heading should appear above or below the group when the outline is collapsed.
2 Use the Data, Group and Outline, Group command. The left hand edge of the screen expands to show the outlining buttons.
3 Repeat as necessary.



Grouping sections of the model to make them more manageable

It is possible to get Excel to Auto Outline your work, but Excel will assume that the summary rows are at the bottom of the group; to amend this, use the Data, Group and Outline menu and choose Settings.

We can create up to eight levels of grouping within a section, each one being numbered on the outlining toolbar to the left of the screen. Clicking the outline level buttons will expand and collapse individual groups, and clicking on the outline number buttons will collapse or expand all groups at that level.

Use the Data, Group and Outline, Clear Outlines command to clear the outlines if not required.

The benefit of outlining is that information can be collapsed and expanded as needed. In large models this can be very reassuring as the amount of information on screen is much reduced, but can be easily revealed. Be aware that audit techniques (F2, Ctrl+[etc.) may point to cells in collapsed ranges. If the outline symbols are not shown, the collapsed rows are hidden and can be revealed by selecting the row above and below and using Format, Row, Unhide, but note that rows and columns can be hidden independently of outlining. If outline symbols are restored, these rows are still grouped and can be collapsed or expanded as before. The disappearance of the outline symbols is the commonest source of difficulty so make sure that the worksheet contains appropriate documentation to advise users to use Tools, Options, View and Outline symbols should this happen.

As might be expected, there are a number of keyboard shortcuts in relation to outlining:

Group rows (or columns)	Alt+Shift+right arrow
Ungroup rows (or columns)	Alt+Shift+left arrow
Display or hide outline symbols	Ctrl+8
Hide rows	Ctrl+9
Unhide selected rows	Ctrl+Shift+(

Hide columns	Ctrl+0
Unhide columns	Ctrl+Shift+)

Do not use the number keypad for these shortcuts.

Data inputs

Once a model has been developed and tested it can be handed over to the users. Assuming we have put in appropriate protections for the workings calculations and the output reports, we may still need to restrict the data being used to drive the model. There are several techniques we can use to restrict data entry to the model.

Data validation

We can set constraints on the boundaries of the values that the user has to enter. This is a simple exercise.

1 Select the cell(s) for which the validation will apply.
2 Use the Data, Validation command.
3 Under the Validation criteria we can specify the constraints to apply – whole number, decimal, etc.
4 In the Data section we can identify the criteria to use (between, not between, equal to, and so on).
5 If you have chosen the Between criterion, you then specify the lower and upper values. With the other criteria you must identify the threshold value.
6 You may wish to enter an Input Message or prompt to guide the user. This will appear when the user selects the cell or cells with the validation. There is no other visual indication in the worksheet that validation is in use.
7 You can also set up an Error Alert or warning if the user attempts to enter invalid data. There are three styles of alert: Stop, which offers the user the opportunity to Retry or Cancel; Warning, which does allow the user to continue with the invalid data, and Information, which gives the options to OK or Cancel.

Custom validation

I may wish to set the unit price of my product as being between 10 and 20, which is easily done with the data validation options of Whole Number, Between . . . I would also like to restrict prices to even numbers.

1 Select the range.
2 Use Data, Validation, and choose Custom.
3 Enter the following formula:

   =MOD(first cell in range, 2)=0

   The MOD (modulus) function returns the remainder after a value has been divided by a number, in this case 2. The data validation requires that each number entered in the specified range must be exactly divisible by 2.
4 Test that the data validation works.

In the depreciation masking exercise in Chapter 4, it was noted that the technique only worked if there was only one investment, regardless of when it occurred. We can use data validation to ensure that only one value is typed in a range.

1 Select the range.
2 Use Data, Validation, and again choose Custom.
3 Enter the following formula, using the appropriate absolute cell references (not range names):

```
=COUNTIF(Investment,">0")=1
```

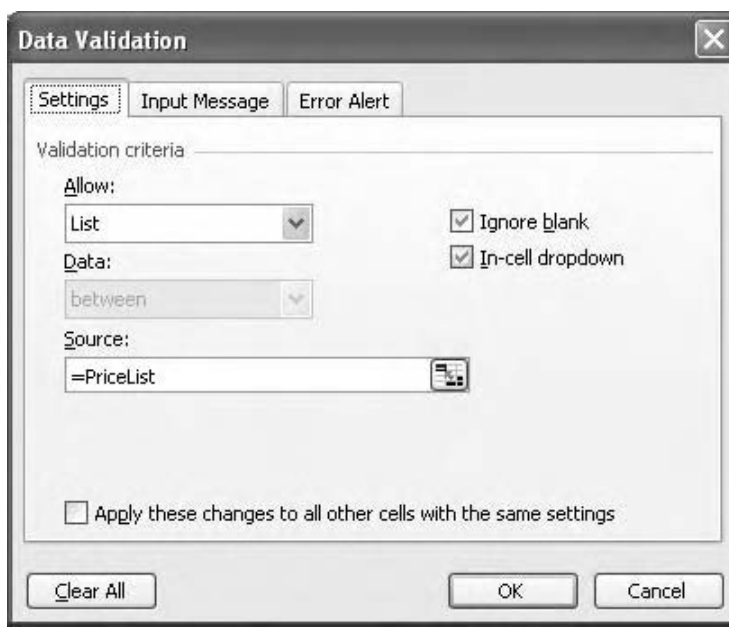
The COUNTIF function counts the number of values in the range which have a value greater than zero. If the count exceeds 1, then the validation restricts further entry.
4 Test that the validation works. Note that zero values and negative numbers could be entered without comment – amend the formula as required.

Problems with validation

If a user copies or fills values from cells outside the validation range, no error is generated if the values breach the validation rule. If validation is applied to existing formulae, and these formulae subsequently evaluate to invalid numbers, the validation mechanism will not activate. Validation should therefore be restricted to data entry (inputs) only.

Drop-down lists

There is one useful feature of validation which fixes the problem about the numbers themselves. We can provide a list of values (e.g. the prices as even numbers) which will then be listed in the validation cell. Either type these values into the same worksheet or enter them



Setting up simple dropdown list functionality with Data validation

directly into the dialog box. My own preference, as always, is to have the values accessible in the worksheet for inspection and review. Note that the values should be listed on the same sheet, which makes sense if we are using the inputs sheet approach. If you need to refer to a list of values on another sheet you will have to use a range name, because data validation will not accept cell references to other sheets.

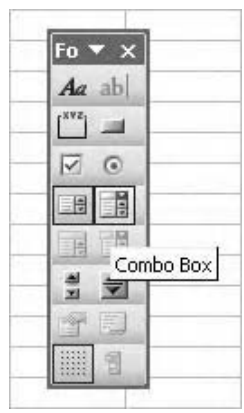
1 Select the cell for validation.
2 Use the Data, Validation command and Allow List.
3 Specify the source cells; if using a range name prefix it with the equals sign, for example, =PriceList
4 Make sure the In-cell dropdown checkbox is ticked.
5 Enter any input message and error alert as required – they may be redundant with this technique as we have more control over what the user is doing.
6 Choose OK.

Click on the validated cell. The dropdown list indicator should appear, from which we can select the value. It is still possible to type in a different value, at which point the error alert will appear, but the user is fairly restricted now and may not consider typing in values of their own. Note that with the validation list, the value selected from the list is the value that is entered in the cell, and is used in the dependent calculations.

List and combo boxes

A more sophisticated form of controlled data entry can be achieved using list and combo boxes. Most people associate these with macros, but they can be directly embedded in the worksheet. They are not too dissimilar to data validation lists, but instead of entering the selected value into the cell, they enter the position number of the value that was selected. We can explore this using the price list mentioned previously, and again I recommend that the source data is written into the worksheet. Unlike validation lists, these data can be on any sheet, and we can also use text rather than just values. The data shown in the list and combo boxes cannot be edited.

The difference between a combo box and a list box is that the combo box produces a dropdown list and if not active only shows one value from the list. The list box can be sized so that the full list is permanently visible, and even if reduced in size will always show two

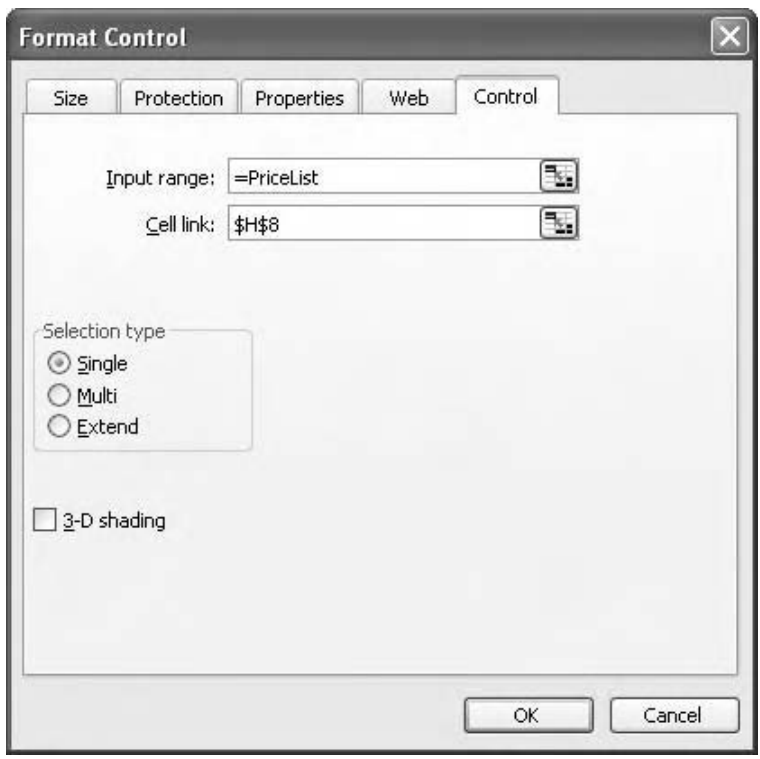


The List Box and Combo Box in the Forms toolbar

values from the list, the others being accessed using the spin controls. Which value is currently being used in the model?

A little thought needs to be given to the set up of the box. Both combo and list boxes require an input range, which contains the source data, and a cell link – a reference to the cell in the workbook which will contain the result of the selection and which can be anywhere in the workbook. Range names are useful in this context. The box itself is a workbook object and sits on top of the workbook, rather than in it. Both types of box are set up in the same way.

1 To set up the combo/list box, access the Forms toolbar by right-clicking on an existing toolbar and choosing it from the list.
2 Click on the Combo or List box tool and click and drag in the workbook to draw the box.
3 The box then appears with edit handles, and the box can be resized or repositioned at will.
4 Right-click on the box and choose Format Control.
5 In the dialog box, enter the input range (the source data) and the cell link (where the information will be placed).



The Input range contains the source information, the Cell link is where the result is to be stored in the worksheet

6 With the Combo box you can specify how many lines you want to show when the box is selected, and with both boxes you can specify if you want 2-D or 3-D effects.
7 Choose OK.

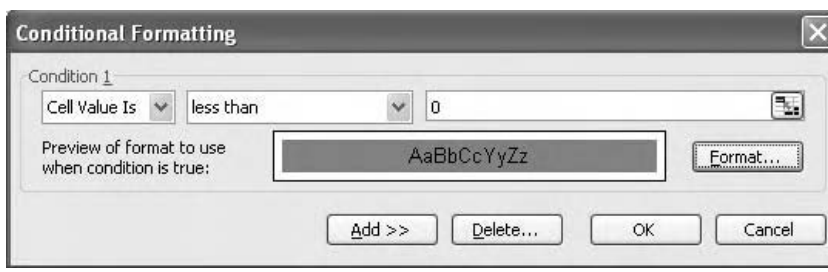
Click in the worksheet to take the selection off the box. Test the box by selecting an item in the list, using either the drop down list (Combo box) or the spin control (List box). Note that a number appears in the cell identified as the cell link. This number corresponds to the position of the selected item in the list, and is not the item itself. This number can then be used for lookup and other types of function, or for running scenarios (Chapter 6).

If you need to edit the boxes you must right-click on them first. It is good practice to name the boxes, which can be done by selecting the box and typing a name into the Name Box on the formula bar. As worksheet objects you can group and ungroup them, and the Select Objects tool from the Drawing toolbar is useful in this context.

Conditional formatting

The ability to change the appearance of cells based on logical conditions is a very useful feature of Excel, although in my experience few modellers get beyond a trivial implementation of conditional formatting. The usual demonstrations for this technique involve changing the cell colour of all negative numbers, which we can very quickly review before looking at how we can really benefit from this feature.

1 Select the range to which the conditional format will apply.
2 Use the Format, Conditional Formatting command.
3 Set Condition 1 as being that the Cell Value Is, Less than, 0.
4 Click the Format button and select the Patterns tab.
5 Choose a colour such as red and choose OK, and OK again.



Simple conditional formatting

Inspect your range and note the appearance of the cells containing negative numbers. Prove that the formatting is dynamic by changing the values.

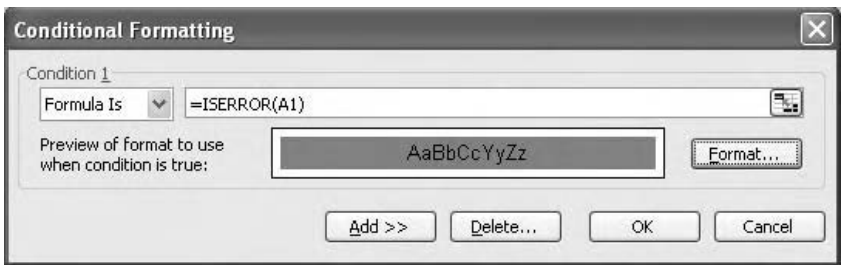
Now let us do something sensible. In the audit sheet section of Chapter 2, we saw how we could use F5 Edit, Go To, Special to locate cells containing errors. We can also use conditional formatting to highlight these cells. The basis for this is the ISERROR function.

1 Select the range to be audited. Make sure the active cell is the top left cell in the selection.
2 Use the Format, Conditional Formatting command.
3 Under Condition 1, change the Cell Value Is to Formula Is in the dropdown list.

4 Type in the test condition as

=ISERROR(Cell reference)

Where cell reference is a relative cell reference to the top left cell in the selected range.
5 Click the Format button and select the Patterns tab.
6 Choose a colour such as red and choose OK, and OK again.



A conditional format that colours in cells which contain error values

Now inspect the worksheet and note the appearance of the error cells. I suggest this makes them somewhat easier to locate. As the errors are corrected, the colour automatically disappears.

In Chapter 2 I suggested that we set up an audit summary cell that summarised all the checks on the audit sheet, and that this cell is linked to the output sheets. This meant that the outputs themselves document the audit status of the model, but in reality I might only want to know if the audit has failed, rather than that it has passed. This is easy to set up.

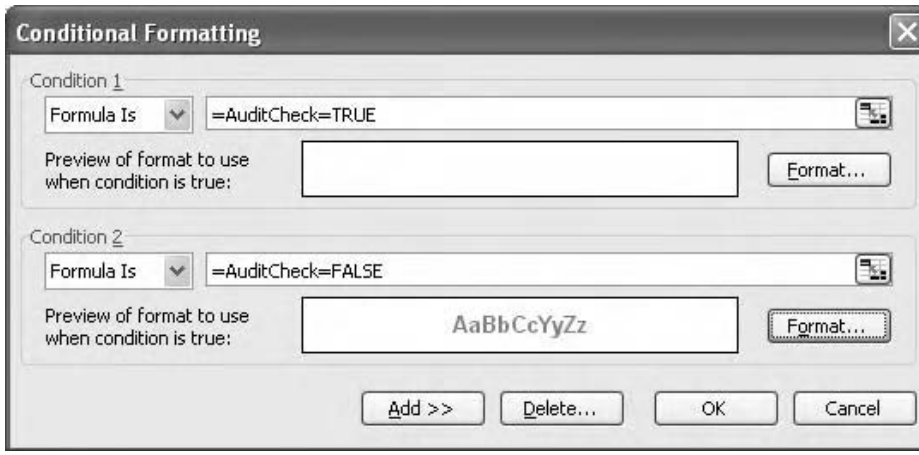
1 Select the output cell containing the link to the audit check cell.
2 In the conditional formatting dialog box, use the following formula (assuming the audit check cell is named AuditCheck)

=AuditCheck

Because AuditCheck contains either the value TRUE or FALSE, we don't need to specify further.
3 Set the format to white text (unethical, but it will serve for this purpose).
4 Optional: You may want to specify a second condition to emphasise the fact that the audit check has failed, in which case you will need to confirm that this is the false outcome:

=AuditCheck=FALSE

Select the appropriate format, and test.



The TRUE option applies a font colour of white, the FALSE option applies a bold, red font

We can use the last technique to good effect if we select the whole sheet and set the AuditCheck=TRUE outcome to have no format, and AuditCheck=FALSE to have text strikethrough. This means that the outputs sheets are unusable if the model has failed an audit check.

In Chapter 2 we considered the use of the base or initialisation column down the left hand edge of the workings calculations. Its purpose is to handle initialisation and time-independent values, and I would normally recommend that it should be coloured, to distinguish it as having a different function to the adjacent calculations. We can use conditional formatting to do this. The condition formula is:

```
=COLUMN()=4
```

This assumes that column D is the base column. Amend as required, and set an appropriate fill colour.

We can extend this logic further in the context of the quarterly/annual modelling problem described in Chapter 3. The rolling sum and the corkscrew methods were both described, in which the annual figure appears in every fourth cell in the row. We can use custom formatting to emphasise this.

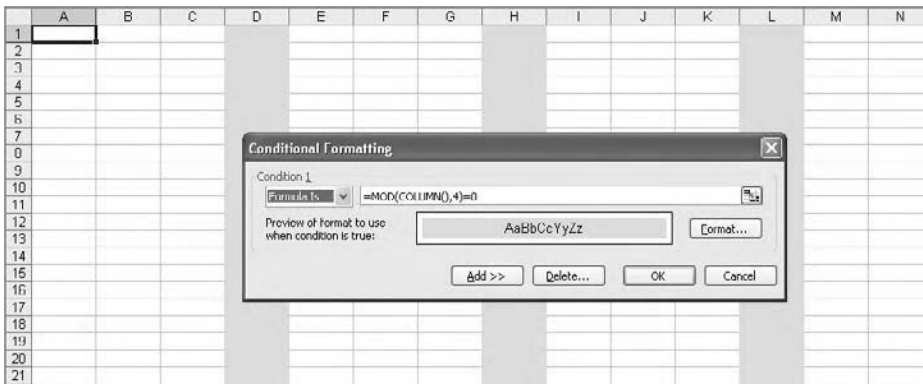
1 Select the sheet (Ctrl+A).
2 Use Format, Conditional Formatting.
3 The Condition 1, Formula Is,

```
=MOD(COLUMN(),4)=0
```

Choose an appropriate colour format.
4 Choose OK.

The COLUMN function returns the column number, and the MOD function divides this number by 4. If the remainder is zero, the format is applied. The sheet should now have coloured stripes every fourth column. When applied to the output sheets, we should be

able to prove that the reports are based on the annual figures because if the quarterly columns have been hidden there should be no unformatted columns.



A visual differentiation of time periods

Problems with conditional formatting

One of the issues with conditional formatting is that the condition is not explicit in the worksheet itself. Conditional formatting takes priority over any other formatting of the same type, so in the examples above, it would not be possible to apply a different fill colour to a column or cell. When using multiple conditions it is important to get the sequence right:

Cell value	Less than	10	Blue
Cell value	Less than	100	Green
Cell value	Less than	1000	Red

This sequence gives the expected results.

Cell value	Less than	1000	Red
Cell value	Less than	100	Green
Cell value	Less than	10	Blue

This sequence gives unexpected results - any value below 1000 is coloured red, and the subsequent criteria are not applied.

Excel only allows three conditional formats on each sheet, so with a couple of sheet level formats you may find yourself running out of functionality.

Custom formatting

Excel offers a number of useful number formats and of course you have learnt the short-cuts for the most common:

Ctrl+Shift+!	Thousands, 2 decimal places
Ctrl+Shift+$	Currency, 2 decimal places

Ctrl+Shift+%	Percentage, 2 decimal places
Ctrl+Shift+^	Exponential format
Ctrl+Shift+~	General number format (only on US/UK keyboards)
Ctrl+1	Format cells dialog box

Microsoft has never yet offered a shortcut to reduce (or increase) the number of decimals.

You will know that having applied a format you can select the next range and repeat the format command using F4.

The principle

In the following example the results on the outputs sheet are some very big positive and negative numbers.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2		**Financial year ending December**			**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**
3														
4		Revenue			0	0	28140673	31517553	34669309	38136239	41560501	45309666	48934439	52849194
5		Operating costs			0	0	10305035	11541639	12695803	13965383	15222268	16592272	17919854	19353226
6		Depreciation			0	0	7354127.5	6618714.7	6279121.3	6521360	6197625.4	5930422.2	6395194.6	6122603.0
7														
8		Operating Profit			0	0	10481510	13357199	15694384	17649496	20140608	22786972	24619691	27373364
9														
10		Net interest			600714	2928972.3	4741207.2	3936302.4	3062197.2	2357643.2	1939478.8	1329341	986080.5	479507.76
11		FX Loss			0	1626933.8	5145527.9	3751947.5	2508444.9	1929573	1286382	964786.49	471673.39	246556.55
12		Profits before tax			-600714	-4555906	594774.93	5669949.5	10123742	13362380	16922747	20492844	23162827	26647300
13														
14		Tax at 30%			0	0	0	3324431.31	3037122.6	4008713.9	5076824.1	6147853.3	6948848.1	7994190
15		Profits after tax			-600714	-4555906	594774.93	5337518.1	7086619.4	9353665.8	11845923	14344991	16213979	18653110
16														
17		Dividend			0	0	0	542971.14	5123524.9	8004623.6	10717533	13256754	13095789	12361076
18														
19		Retained earnings for the year			-600714	-4555906	594774.93	4794547	1963094.5	1269042.3	1128389.8	1086237.4	3118189.7	6292034.1
20														

"Good enough for government work"

The accountants teach us that more than three digits tends to confuse, so it would be helpful if we could display the number in the format shown.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2		**Financial year ending December**			**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**
3														
4		Revenue			-	-	28.14	31.52	34.67	38.14	41.57	45.31	48.93	52.85
5		Operating costs			-	-	10.31	11.54	12.70	13.97	15.22	16.59	17.92	19.35
6		Depreciation			-	-	7.35	6.62	6.28	6.52	6.20	5.93	6.40	6.12
7														
8		Operating Profit			-	-	10.48	13.36	15.69	17.65	20.15	22.79	24.62	27.37
9														
10		Net interest			0.60	2.93	4.74	3.94	3.06	2.36	1.94	1.33	0.99	0.48
11		FX Loss			-	1.63	5.15	3.75	2.51	1.93	1.29	0.96	0.47	0.25
12		Profits before tax			(0.6)	(4.6)	0.59	5.67	10.12	13.36	16.92	20.49	23.16	26.65
13														
14		Tax at 30%			-	-	-	0.33	3.04	4.01	5.08	6.15	6.95	7.99
15		Profits after tax			(0.6)	(4.6)	0.59	5.34	7.09	9.35	11.85	14.34	16.21	18.65
16														
17		Dividend			-	-	-	0.54	5.12	8.00	10.72	13.26	13.10	12.36
18														
19		Retained earnings for the year			(0.6)	(4.6)	0.59	4.79	1.96	1.27	1.13	1.09	3.12	6.29
20														

The effect of custom formatting

It is not really worth the effort to simply divide all the results by 1,000,000, because Excel has enough work to do on the underlying calculations without worrying about these trivial divisions. Instead, we use a custom format. The cells will be formatted so that they are shown as decimal values, with negative numbers in parentheses and coloured red, and with zero values replaced with hyphens. We will build the format in steps:

1 Select the range to be formatted.
2 Use Ctrl+1 Format, Cells.
3 Select the Category of Custom and tab to the Type box.
4 Type in 0.0,, and choose OK. All the values in the range are now shown in decimal format. The commas in the format code are instructions to divide by a thousand, twice. Please note that this is the UK/US thousands notation: use the appropriate characters for your own regional style, so for example, when working in Germany I write 0,0... (When working in Turkey I might write 0.0,,, to handle the billions).
5 At the moment the format does not clearly differentiate between positive and negative numbers. Return to the Format cells dialog box. After the existing 0.0,, type a semicolon ; and then the format for the negative numbers, which will be in red with parentheses. The code for this is:

```
0.0,,;[red](0.0,,)
```

The first part of the format code is for positive numbers, the second part for negative numbers. The colour is enclosed in square brackets, and again use the appropriate language to name the colour. Excel has eight named colours: black, blue, green, red, yellow, cyan, magenta, and white. It is considered unethical to format values or text with white. Click OK to inspect the results in the worksheet.
6 On careful inspection you may note that the decimal points of the positive and negative values no longer line up. On the grand scheme of things, this may not be anything to worry about, but some people, usually fairly senior in the organisation, like to see everything lined up all neat and tidy. The problem here is that the closing bracket of the negative value has pushed the value into the cell, whereas the positive numbers are still hard up against the right-hand edge of the cell. We need to fix this, so select the numbers again and press Ctrl+1.
7 To allow for the bracket of the negative numbers, we need to move the positive numbers into the cell by the same distance. Amend the positive part of the format code with an underscore followed by a bracket:

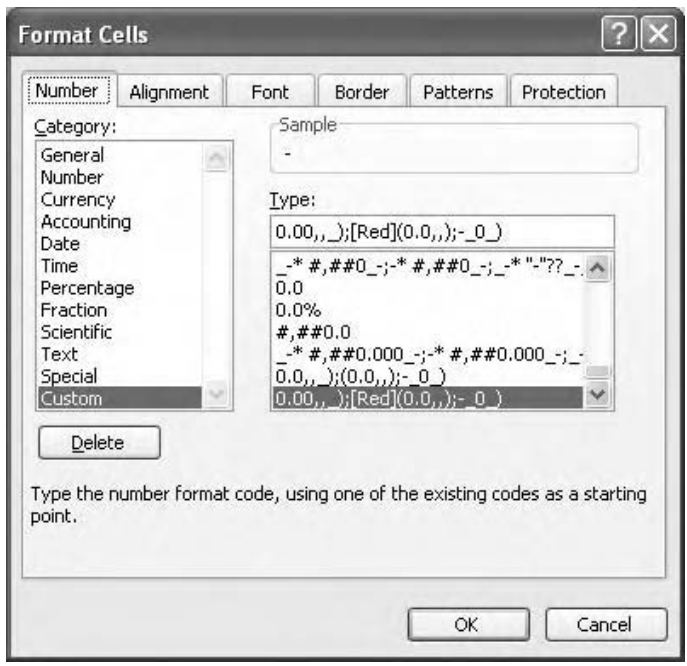
```
0.0,,_);[red](0.0,,)
```

The underscore_ character is the code to make the following character invisible. Click OK to see the very minor effect on screen, but verify that the decimal points of both positive and negative numbers do in fact line up. I am often asked if we could just use a space in the format code, rather than the nonsense of the invisible character, and indeed a space would do the trick just as well. The difficulties are that firstly the space is then difficult to see in the format code, and secondly that spaces do not have a fixed measurement – if you have ever tried to line up text on two different lines in Microsoft Word you will have discovered this.
8 The results now look quite attractive in the worksheet, but we still have zero values displayed. These just clutter up the screen, and it would be helpful if they could be converted into hyphens. Select the cells and return to the Format Cells dialog box.
9 The first part of the format code is for positive numbers, the second part is for negative numbers, and the third part is for zero values. For neatness the hyphen needs to

be in the same position as the decimal point. Type another semi-colon and the following code:

0.0,,_);[red](0.0,,);-_0_)

The hyphen is the character to represent the zeros (if omitted the cells would appear blank), the _0 hides the decimal value and the _) hides the bracket, the combined effect of which is to push the underscore into the same position as the decimal point. Click OK.



The completed custom format code

10 For completeness I will mention that there is a fourth component of the custom format which is for text. Type a further semi-colon and, for example, a colour (in square brackets). Type an @ sign, and click OK. If you omit the @, Excel substitutes the word General into the code. I do not have much use for this format option as my text is to the left and right of the workings area.

At this stage you will probably want to stampede off to your other output sheets to format them as well. There are a couple of useful techniques here. You probably already know about the Format Painter tool, where you click on a cell containing the source format, click the Format Painter, and then click and drag over the destination cells. A little known trick is to double-click the Format Painter, after which the tool stays active as you repeat the

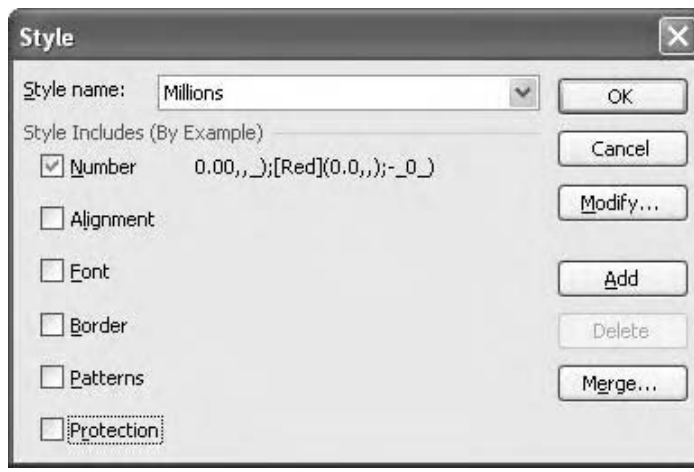
click and drag action over cells on different sheets (it normally switches itself off). You will lose any borders and colours on the destination sheets but they are easy to restore.

Alternatively note that your custom formats will appear at the very bottom of the custom format list in the Format cells dialog box, while the formatted workbook is open.

Style

A more sophisticated solution is to set up a Style.

1. Select a cell that contains the custom format, and run Format, Style.
2. Type in a style name, such as Millions. Switch off all options other than Number.
3. Choose OK.



Clear the check boxes that do not apply to the Millions style

To apply the format to other sheets, select the destination cells and use the Format, Style command (or Alt+'), and select the style from the list. You can also customise your toolbar with the Style dropdown list. To use the Millions style in other workbooks, you can use the Format, Style, Merge command.

Reporting

When using the Millions format we recognise that Excel is rounding the numbers quite shamelessly, so we should change the page footer to include the standard caveat 'Numbers may not agree due to rounding'. Also, Excel treats any value of 49,999 or less as zero – if this magnitude of number is significant for your purposes, amend the format code to 0.00 etc. I would also strongly recommend that you only use this formatting on your output sheets: the inputs sheet values must be in the same units as expressed in the data book or source documentation (Chapter 2). The workings values should be left as they are, with perhaps comma formatting to make them readable. Over the time you have spent developing your model, you have probably become quite familiar with some of the numbers and

would recognise them easily. If you were to apply the millions format they become difficult to appreciate, particularly when Excel starts rounding them.

Currency

Another example of the use of custom formatting is to include a currency symbol in the cell, but showing the symbol on the left hand side rather than immediately adjacent to the value. Using Ctrl+1 we enter the currency symbol followed by an asterisk * and a space, followed by the appropriate number format. This odd combination instructs Excel to show the symbol and then to repeat the space character as many times as possible before showing the value. Remember to include the currency symbol and the repeater on both sides of the format, otherwise negative values will be unaffected.

€* #,##0_);[red]€* (#,##0)

In some circumstances we might want to show the currency abbreviation rather than the symbol, for example GBP for British Pounds, or USD for US dollars. We find that we can happily enter GBP as the currency prefix but not USD. This is because certain letters are reserved characters, and in this case the S is reserved for Seconds in the date and time formats, so we find that the letters M, D, H, and E are not available. The workaround is simply to enclose the string in quotes:

"USD"* #,##0_);[red]"USD"* (#,##0)

A type of conditional format

We can also use custom formatting for a basic type of conditional formatting. If we need to flag up numbers that are above a particular value, for example cash balances in excess of 100,000, we can select the range and apply the following format:

[blue][>=100,000]0;0

The zero outside the square brackets represents the number format to be applied to values above 100,000. The zero after the semi-colon represents the formatting of numbers which are below 100,000. We can add a second condition, which in combination give three potential formats.

[blue][>=100,000]0;[green][>=50,000]0;[red]0

Values above 100,000 appear blue, above 50,000 green, and below 50,000 red. Take care in putting the conditions in the correct order. I think this is of marginal interest.

Protection

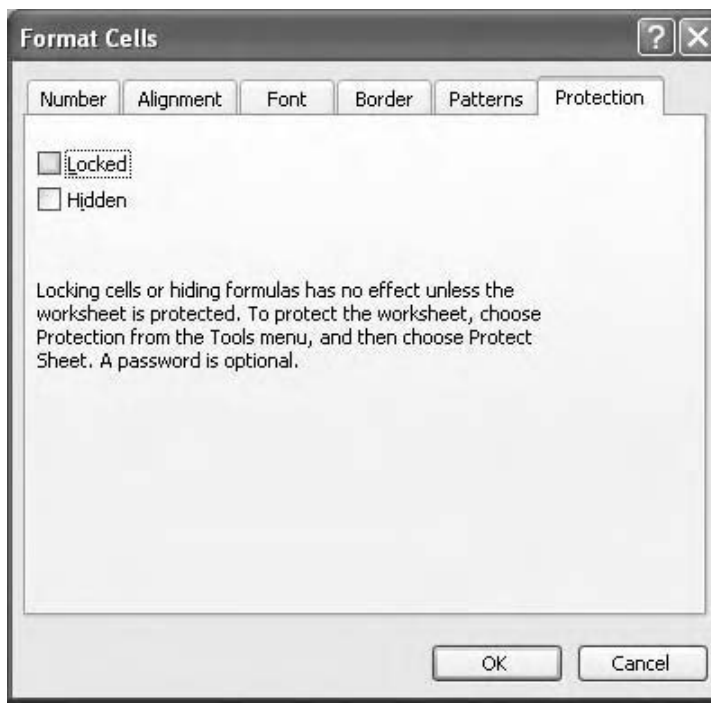
Workbook and worksheet protection allows us to prevent users from opening files or changing the contents of a sheet, unless they have the password. You should be aware that

if you forget or lose the password, not even Microsoft will be able to help you. There are third party codebreakers who may be able to help, but their services are at a premium. I prefer to use the techniques described previously to restrict the way users can interact with my models and I only use protection as a last resort. Even then, I make sure that the password is known, and I have even written it in the model documentation. If the user knows that they are not supposed to change anything and yet they go ahead and unprotect the worksheet, it suggests that they are acting wilfully and deliberately and must therefore take the responsibility if anything goes wrong.

Sheet protection

With sheet level protection there are two steps. The first is to identify those cells which will remain unprotected when protection is switched on, and then there is the protection command itself.

1 Select the cells which will be unprotected. The use of a fill colour may be appropriate as there are no visible differences between protected and unprotected cells.
2 Use Ctrl+1 Format, Cells and go to the Protection tab.
3 Uncheck the Locked box.



Protection#1: unprotect the cells

4 This has no effect until the worksheet is protected.
5 Use the Tools, Protection, Protect Sheet command.
6 The options relate to Contents, Objects and Scenarios. Select or deselect as required. (Objects are things like embedded graphs and macro buttons). Click OK.

If you now attempt to edit a protected cell Excel will respond with a dialog box notifying of the protection.

There is also a slightly unethical option at step 3, in which the other option is that the cell content is Hidden when the worksheet is protected. If this is applied the user cannot see the formula on the formula bar. It is probably worthwhile deselecting the Hidden box while setting up protection.

Workbook

Workbook protection is used to prevent structural changes to the file. Users will not be able to move, delete, or insert sheets, but they will be able to write, edit, and delete values and formulae unless worksheet protection has also been applied. A protected workbook does not require a password to open it, but a password may be required to disable the protection.

Workbook protection is a one step process. Use Tools, Protection and Protect Workbook. Provide a password if required, and re-enter the password at the prompt.

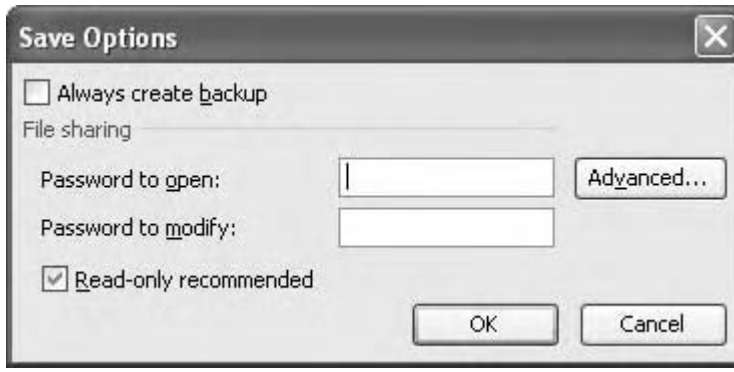
Protection#2: protect the workbook

Use Tools, Protection, and Protect Workbook to undo the protection.

Password protection

A different way of approaching protection is to save the file and set up a password for the workbook to be opened or modified. This command sequence relates to the more recent versions of Excel/Windows.

1 Run the File, Save As command (F12).
2 Click the Tools button in the dialog box.
3 Choose General Options from the list.
4 Enter a password in the Password to Open and/or Password to Modify box, click OK and confirm the password.



Setting the read-only attribute

5 The file can be made Read-only, which will allow users to open and modify the file but they will be unable to save the file with the same name.

With Password to Open, the user is prompted for the password and cannot open the file without it. Password to Modify, if provided correctly, allows the user to open the file and amend the file. If they do not supply the password they can open a read-only version of the file and will be prompted for a new filename if they attempt to save their work.

With the modelling methodology and structure proposed in Chapter 1, it would be appropriate to protect the workings and outputs sheets because there should be no reason for the user to modify the calculations or the results. The inputs sheet would remain unprotected so that the users can run sensitivities and generally flex the assumptions. But given such a clear layout, the use of protection is practically redundant.

Problems with Excel commands

Under certain conditions your users will complain that Excel isn't working properly. Menu commands and other tools may not be available if:

- the workbook is protected
- Group mode is active
- Tools, Options, View, Show all is off
- Print Preview mode
- the window is frozen
- Edit mode is active (you never know...)

Documentation may be required to support users if such techniques have been applied.

CHAPTER 6

Sensitivity analysis and scenarios

Introduction

Almost by definition, the purpose of any financial model is to explore the effects of changing the input assumptions. Sensitivity analysis, also referred to as what-if analysis, tends to refer to the process of adjusting one or two key drivers and observing or recording the results; scenario management refers to the process of changing several drivers to create specific operational or economic situations. A further refinement involves the assessment of likelihood or risk involved with a particular scenario. It is worth emphasising that we should only use the one model for analysis. It is often tempting to develop scenarios in a series of copies of the original workbook, but this can lead to immense problems in the future if one of the common assumptions or calculation routines is then revised. Even a relatively trivial adjustment can then take several hours or days to distribute to the suite of scenario specific workbooks, and even with the development of the change tracking functionality in Excel it can be difficult to reconcile one workbook with another.

In this chapter we will look at a number of techniques for manipulating or flexing financial models, but before we start, backup your work first. It is not helpful to embark on a review of the model's assumptions and then to lose the original figures. For sensitivity and Goal Seek analysis, cell comments may be sufficient. We will consider other techniques for scenario management.

The demonstration workbooks for this chapter are in Chapter 6: Sensitivity Analysis and Scenarios folder on the CD-ROM.

Goal Seek

A good starting point is the Goal Seek tool. You have built your model and generated a result, *R1*. Assuming you need result *R2*, what should input *I* be in order to obtain this result? You can change the input manually and then inspect the result (which is fine if you are on billable time), but a more efficient way is to use Goal Seek.

Select the cell containing the result you wish to specify. This cell must contain a formula. Run the Tools, Goal Seek command, and identify this result cell as the Set Cell. Next,

specify the result you would like in the To value box. Finally, identify the By changing cell reference. This cell must contain an input value. Click OK.

The Set cell and changing cell can be on any sheet

Note the use of the singular throughout. We cannot specify the results for a range of cells, neither can we obtain the results by changing several inputs, although one workaround is to put link formulae on the inputs sheet, such that the value generated by Goal Seek is copied across the row. It does not matter which sheets contain the Set Cell or the Changing Cell.

Excel uses an iterative technique to obtain the result. If you recall the discussion concerning the use of iteration in Chapter 3, the Tools, Options, Calculation dialog offers the constraints of either 100 iterations, or until the maximum change is 0.01 (the defaults). Goal Seek therefore follows whichever of these constraints apply. Make sure that this is not obscured by formatting, if needs be retype the number generated by Goal Seek with appropriate rounding and common sense.

If you are also using iteration for circular code elsewhere in your model, Goal Seek can produce absurd results as it iterates its own iterations, even if the results under analysis are not involved in the circularity.

Data tables

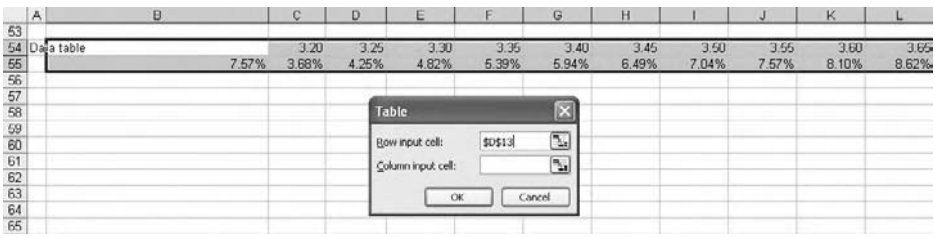
Data tables have been around since the earliest versions of Lotus 1-2-3 and Excel, and although they are very useful tools they contain several limitations which have never been addressed. These will be considered below. The concept of the data table is simple – Goal Seek offers the ability to specify one input variable and to see the effect on one formula. The data table allows us to provide a range of input values and to record the results as these values are run through the model. We can use either a one-way analysis, in which one variable is tested and the results of one or more formulae are recorded, or a two-way analysis, in which two variables are tested against each other and the results of a single dependent formula are displayed. Lotus 1-2-3 had the facility for three-way analysis, but Microsoft has never pursued this.

The one-way data table

The one-way or one-variable data table allows you to specify a range of values to be substituted into a single input variable cell, and to record the results of the analysis. On the inputs sheet (or equivalent), locate the cell which contains the variable to be tested. If this is at the start of a row or column, you should enter link formulae in the rest of the range.

Further down the sheet, either write the formula you wish to use, or put in a link to the formula if it is located elsewhere in the workbook (it can be helpful to split the screen or open a new window). With a one-way table you can test several formulae. In the row beginning in the cell above and to the right of the first formula cell, enter the list of values you wish to run through the model. If you have more than 255 of them, you will need to transpose the range such that the formula cell is above and to the right of the column of values. In either case there should be a blank cell in the top left position. Select the blank cell, the formula cell(s), the row (column) of inputs, and the cells in which the results are to appear, and then run the Data, Table command.

In the dialog box, specify the Row input cell as the cell into which the row of numbers are to be placed (Excel is not asking you for the location of the data table – you have already done that). Click OK. Inspect the data table.

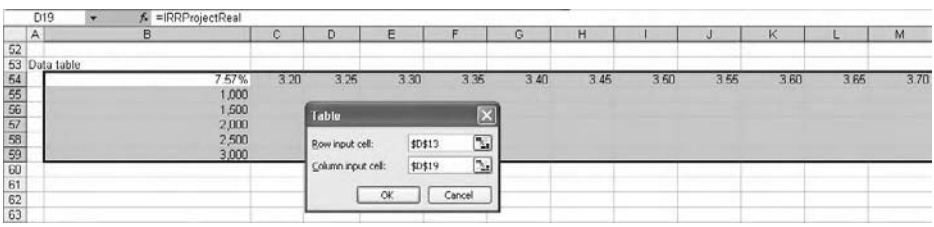


A one-way data table. The input cell is off screen but on the same sheet

If the data table is showing the same result across the row (or down the column), press F9 (see below). If this has not helped, check that you have appropriate link formulae in the input row, that the input row actually feeds into the model, and that the formula under test is actually dependent on this particular input.

The two-way data table

The two variable data table allows us to test the interaction of two input variables, but we are restricted to one formula. The basic structure is as above, but the formula cell is placed at the intersection of the input row and the input column.



Two way data table. The formula is at the intersection of the row and column input values

Observations

Once a data table has been set up the values in the input row/column can be changed at will.

Unlike Goal Seek, data tables must be set up on the same sheet which contains the input variable under analysis. Several of the other modelling rules proposed earlier in the book are also redundant when using data tables.

The data table formula is an example of an array formula (Chapter 3). This type of formula is recognised by the curly brackets { }. Although an efficient way of writing code, problems arise if the user attempts to edit the formula. On pressing Enter, we are informed that 'That function is not valid'. The immediate solution is to press Esc. Array formulae can only be edited as a whole range, so any action such as inserting or deleting columns, or deleting cells, will generate the error message. The data table, or more specifically the results of the data table, must be deleted if other such actions are required. The TABLE formula itself cannot be written manually.

Table recalculation

Data tables can be very memory intensive, as Excel will recalculate the data table along with all the other calculations in the model every time something changes. This can slow down the model's functionality considerably, so you can change the recalculation options under Tools, Options, Calculation, and choosing Automatic Except Tables. This will allow the model proper to recalculate as before, and recalculation of the data table is forced by pressing F9 (Recalculate). This can lead to problems if, for example, the data table is printed without having been updated. The solution is to recognise that the formula driving the data table is not subject to the recalculation exception and calculates normally. You should make sure that the current model input for the data table (the value in the row or column input cell) also appears in the data table input row or column. The result underneath or next to this value should be the same as the driving formula. If it is not, press F9.

Bear in mind that recalculation options are persistent and remain in effect if other workbooks are opened in this session; they are also workbook attributes and will be saved in the file, to be put into effect automatically when the workbook is next opened.

As with Goal Seek, data tables only allow for one row input cell and one column input cell. If the variable is to be used across the forecast period, then make sure you have link formulae in the relevant row, otherwise the table will show only the sensitivity to changing the first value. A disadvantage of this limitation is that it is not possible to model trends or manipulate values in other ways. For example, if I were looking at the sensitivity of my new product to price, I could create a data table to show the effect of price change on NPV. I could put an escalator formula in the input row to increase the price by 2% each quarter, but would I actually want to pursue this strategy at the higher price end of my data table values? Likewise, the data table would not allow me to explore price skimming and other techniques.

Scenarios

Scenarios are used for more sophisticated analysis, where the effects and interactions of multiple variables need to be explored. These can be set up as formal, defined, scenarios – worst case/most likely/best case, in which the variables in each scenario have been chosen to represent a particular position. Alternatively the model can be used to test particular permutations

of the variables, which can be either controlled, by using data validation/dropdown lists to restrict the analysis to a specific set of variables, or uncontrolled, in which the user can change any and all variables.

Setting up a scenario requires some careful thought beforehand. The first point is that the values used in the base model must be documented, so that the user can always undo their changes. Second, how are the results to be used? It is difficult to display the results of two or more scenarios for comparison without using Paste Value techniques. Third, make sure that the relationships between the variables are clearly understood and, if needed, fully documented. Inflation, for example, has a creeping effect over a period of time, but it tends to manifest itself in step changes – retail prices, for example, tend to go up in whole units (and the marketing types will be sensitive to the 9.99 barrier!). Similarly most wages are adjusted annually, and this would need to be addressed if using a quarterly model (see Chapter 3 and the use of masks). Make sure that linkages are recognised:the total variable cost should vary with changes in production levels, but the fixed costs should remain the same. And one that I still find slips through – interest rates are a function of inflation.

Finally, the user should be clear about the cells and ranges which are allowed to be changed. The inputs/workings/outputs structure would restrict the user to the inputs sheet only, and the sensible use of colour, cell comments, dropdown lists and the like, should guide the user into the safe use of the model. A further development of the model structure might be a variation on the usual layout – many models contain large numbers of inputs, of which only a few may be identified as key drivers for scenario analysis. In this case it might be worthwhile setting up a separate scenario inputs sheet which contains the relevant variables which then feed into the workings. A further development would be to link the key outputs back to this sheet, so that the results of the scenario can be seen alongside/underneath the assumptions. This can be quite effective if set up properly.

There are several techniques we can use to set up scenarios.

Scenario Manager

This is a tool that comes with Excel and lives under Tools, Scenarios. It has some neat functionality, but is rather restricted in what it does and generally does not conform with the principles of good modelling practice. The Scenario Manager is restricted to a maximum of 32 changing cells, which can be worked around by using link formulae to copy the changing cells across the row. Furthermore, the Scenario Manager is strictly sheet specific which means that although we might expect all the inputs to be on one sheet anyway, we cannot run the scenario from an output sheet, for example. It is possible to produce summaries and pivot tables of scenarios, but this requires the results cells to be on the same sheet as well. Another point against the Scenario Manager is that the values used for the changing cells are stored in the dialog box itself and are not visible in the spreadsheet until the specific scenario is run (the "black box approach"). This means that the significance of a particular value cannot be directly documented. A further dimension of this problem is that it can be difficult to identify the changing cells in the worksheet, although this can be easily solved by the use of cell colour while setting up the scenario in the first place. Is it worth mentioning that it is perfectly possible to create a group of scenarios which each have a different number of changing cells, or indeed use different changing cells? And in the absence of the scenario toolbar, which scenario is currently in use?

Excel's Scenario Manager – the black box

The key issue arising from the consideration of Excel's Scenario Manager is that the variables are stored in the dialog box and not in the workbook and therefore are not immediately available for inspection or audit. The need for such transparency is addressed by the use of other techniques, such as the CHOOSE function and the LOOKUP functions.

CHOOSE

This function is fairly widely used and might be described as a quick and dirty approach to scenarios. The syntax of the function is:

=CHOOSE(number, first item, second item, third item, ..., twenty ninth item)

Number is a value (or reference to a cell or formula which returns a value) representing the position of an item in a list

First item is the item returned from the list if number is 1; and so on up to 29. The item could be a value, reference, formula, or text.

In the following example, we will look at the use of the CHOOSE function to test three pricing scenarios: a constant price across the forecast period, a loss-leading scenario (low initial price then increase), and a price skimming scenario (high initial price, followed by a reduction).

For the original input variable on the inputs sheet, we can now add additional rows to reflect each scenario. Make sure each row is numbered so that we can crosscheck the scenario number against the scenario row (we will see this again in the lookup section further on). The assumptions underlying each scenario can also be documented.

	A	B	C	D	E	F	G	H	I	J	K	L	M
4													
5	Price												
6	1	constant		18	18	18	18	18	18	18	18	PriceConstantIn	
7	2	loss-leading		14	14	16	17	17	18	18	18	PriceLossLeadingIn	
8	3	price skimming		20	20	18	18	16	16	16	16	PriceSkimmingIn	
9													

The three scenarios as described on the inputs sheet

We then set up a cell which will contain the scenario number. As this contains an input value it should be located on the inputs sheet, and documented as such. It is well worth using the dropdown list/data validation functionality from Chapter 5 in this context.

On the workings sheet the original formulae that linked to the input need to be updated. The CHOOSE function can be written:

=CHOOSE(PriceCell,PriceConstant,PriceLossLeading,PriceSkimming)

D9 =CHOOSE(PriceCell,PriceConstant,PriceLossLeading,PriceSkimming)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Financial year ending**			**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	***Years***	
2													
3		Price scenario number		1	PriceCell								
4													
5	Price												
6		constant		18	18	18	18	18	18	18	18	*PriceConstant*	
7		loss-leading		14	14	16	17	17	18	18	18	*PriceLossLeading*	
8		price skimming		20	20	18	18	16	16	16	16	*PriceSkimming*	
9		scenario price		18	18	18	18	18	18	18	18	*Price*	
10													

The corresponding workings sheet, with the PriceCell switch

By changing the PriceCell value, each scenario can be run rapidly and reliably.

A further elegant touch is to set up an additional cell which returns the name of the scenario:

=CHOOSE(PriceCell,"Constant price","Loss leading","Market skimming")

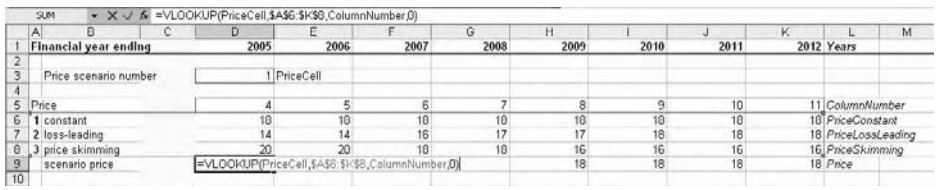
If the output sheets are linked to this cell, we can solve the problem of documenting the name of the scenario in current use.

Care is needed when using CHOOSE. If the user enters an invalid scenario number, #VALUE! errors are generated (hence the suggestion to use dropdown lists/data validation). If the order of the scenarios listed on the inputs sheet should change, or if a scenario is added or removed, the dependent CHOOSE functions will need updating. This is something we generally wish to avoid. However, it is very simple to use, and if set up sensibly can prove to be robust and reliable.

We can easily extend the use of CHOOSE such that each of the key drivers identified can have its own CHOOSE functionality. For each of these inputs we can prepare a scenario list and scenario cell. In this way, for example, we could test the price skimming scenario with corresponding high promotion/low promotion scenarios, with PriceCell set to 3 and PromotionCell set to 1, and so on into a number of permutations. Alternatively, with the example of best case/most likely/worst case, we could set up three scenario lines for each driver, such that on entering a 1 in the scenario cell the model assembles the best case scenario.

LOOKUP

Another way of running scenarios is to use the LOOKUP functions in place of CHOOSE. We can use the same example as before, but this time we recognise that the rows containing the price scenarios are effectively lookup tables. I suggested that the rows were numbered, but this technique works just as well with the row heading text. We keep the price cell as before. We will need to put a column number in each column so that the VLOOKUP can offset accordingly (see Chapter 4)



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Financial year ending**			**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	*Years*	
2													
3		Price scenario number		1	PriceCell								
4													
5		Price		4	5	6	7	8	9	10	11	*ColumnNumber*	
6	1	constant		18	18	18	18	18	18	18	18	*PriceConstant*	
7	2	loss-leading		14	14	16	17	17	18	18	18	*PriceLossLeading*	
8	3	price skimming		20	20	18	18	16	16	16	16	*PriceSkimming*	
9		scenario price		=VLOOKUP(PriceCell,$A$6:$K$8,ColumnNumber,0)				18	18	18	18	*Price*	
10													

Using a lookup table. Note the numbering in column A, which forms part of the lookup table itself

Instead of the CHOOSE function, write:

=VLOOKUP(PriceCell,PriceTable,ColumnNumber, 0)

This works by looking for the value (or text) in the PriceCell in the first row of the lookup table or scenario block. VLOOKUP then counts across ColumnNumber columns and returns the scenario price.

This technique takes the same caveats as CHOOSE, concerning invalid scenario numbers and re-arranging the scenario order, but is still effective and reliable if due consideration is given when setting up. It can also be used to create the scenario permutations and the fixed scenarios described above.

Both CHOOSE and VLOOKUP are limited in that the inputs sheet can fill up rapidly with row after row of scenario variables. Theoretically each one should be documented and with reference to the issue of linkages mentioned above, it may be necessary to advise that certain permutations of variables may not be valid (e.g. high interest/low inflation). Furthermore, the standard auditing checks (F2 Edit Cell, Ctrl+[Select Precedents) do not indicate which specific cells are used in each calculation. However, all the information is available in the workbook, and it should be fairly easy to see the differences between one scenario row and the next. The main problem is that particularly in the scenario which has been assembled from several different drivers (constant price, low promotion, low volume, etc.), it can be difficult to see the specific numbers being used in the scenario, and in some cases the scenario can be difficult to reconstruct. These methods are best used when the number of scenarios and their permutations are small.

Multiple input sheets

One solution to the problem of creating more complex scenarios is to model them separately on individual sheets. Each sheet contains just the assumptions related to that scenario, with

appropriate documentation. This gives more control and reduces the potential for multiple and/or invalid permutations. I mentioned in the introduction to this chapter that we should use the original model when carrying out scenario analysis, rather than multiple copies of the workbook, and this argument extends to the use of a single workings sheet. The implication of this is that each of the scenario specific inputs sheets should, at the user's request, feed into the workings sheet and through to the outputs. Multiple workings sheets, as with multiple workbooks, would be difficult to update, modify, or audit.

I have used and taught the following technique for several years, but unlike the other techniques I have described in this book, I have never had any feedback about it. I can only assume either that my exposition of it is so clear and concise that analysts are able to instantly comprehend the elegance of the technique and apply it themselves, or alternatively that people prefer CHOOSE and VLOOKUP. I rather suspect the latter.

There are two versions of this technique, one which uses cell references and one which uses range names. The former is the simplest and we will explore it first. Step one in either method is to copy the original inputs sheet within the same workbook. Use Edit, Move or Copy Sheet (or Ctrl+click and drag the sheet tab). Give the copied sheet a sensible name.

The cell reference method

If you have used the methods described in Chapter 2 you should have formulae on your workings sheet which link to the inputs, and you will have avoided calculations which combine references from different sheets, for example,

This

Cell E3 contains	=inputs!E4	Link to inflation rate on inputs
Cell E4 contains	=D4*(1+E3)	Inflation index calculation

Rather than this

Cell E3 contains	=D3*(1+inputs!E4)

If you have used this mixed referencing (three-dimensional calculations) you can still use this technique but be careful. Also if you have the habit of prefixing all your formulae with a plus sign (=+inputs!E4) you should proceed with care.

To force the formulae to read from the new (copied) inputs sheet, we can simply run a Replace command on the workings sheet which will convert all references to the old input sheet name to the new input sheet name. Use Edit, Replace (or Ctrl+H). My own preference is to use the exact sheet name specific string = inputs! and replace with =BaseCase! If you have mixed referencing and/or the plus sign prefix then just use the sheet name! syntax itself – your need for care here is because sometimes the sheet name description may have been used legitimately as text elsewhere in the sheet and you may not wish to replace it. If this is likely to be the case, select all the calculations on the workings sheet, because Edit, Replace will then only work within the selection. Before considering the extended use and functionality of this technique, we will look at the rather more arcane range name method.

The range name method

You may find it helpful to refer back to the treatment of range names in Chapter 3, especially the topic of sheet level names. You should also ensure that you have not used any three-dimensional calculations, otherwise the following technique will not work:

So this...

Cell E3 contains	=InflationIn	Link to inflation rate on inputs
Cell E4 contains	=D4*(1+InflationRate)	Inflation index calculation

Rather than this...

Cell E3 contains	=D3*(1+InflationIn)

If you have the habit of prefixing your formulae with the + sign you may find it helpful to run a quick Ctrl+H (Edit Replace) to convert all your =+ formulae prefixes with the simple =.

When you created the copy of the original inputs sheet, you also copied all the range names on that sheet. The names on the original inputs sheet are global names and those on the copy are local, or sheet level, names. If you inspect the formulae on the workings sheet, the references to original inputs still stand. To pick up the new inputs, the range name must be prefixed with the sheet name.

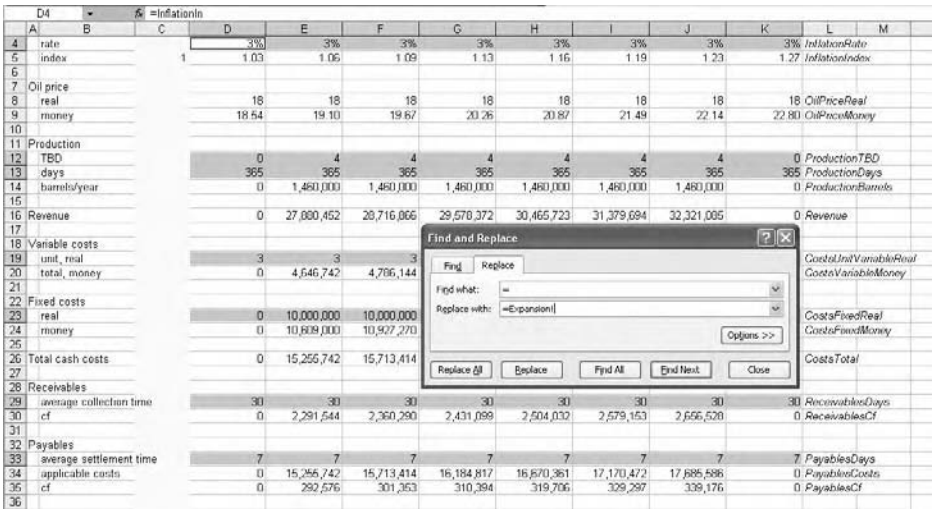
	A	B	C	D	E	F	G
7	Oil price						
8		real		=Expansion!OilPriceIn		18	18

Switching scenarios with sheet level (or local) names

If we were to attempt the same Replace command used with the cell references, we are immediately struck by the problem of the search string – if you have used the conventions suggested in earlier chapters you may have differentiated your input range names using the suffix –In (InflationIn etc.). But the sheet name needs to prefix the formula. We don't have the time to waste on searching manually through the workings sheet to identify all formulae containing In (which might include inflation and interest!).

We need a method which will allow us to identify all cells which contain references to the inputs sheet. In Chapter 1 we looked at several techniques for locating cells and information, one of which was the Ctrl+] (Select Dependents) shortcut. However, we noted that it was formula driven: if an input value was selected and there was no dependent formula on that sheet, the shortcut would not work. This leads to the big step: we now select all the inputs on the original (global names) inputs sheet and Cut and Paste them onto the bottom of the workings sheet.

When named areas are cut and pasted the name moves with range. The global input names are now all located in the workings sheet, and at this stage should all be highlighted. If you now press Ctrl+] you should find that Excel highlights all formulae that contain references to these ranges.



Switching from global names to sheet level names

Having differentiated input links from workings calculations, you can now run Ctrl+H (Edit, Replace), using the equals = sign as the search string. The Replace string is (e.g.) =Expansion! Test to see that it works.

The final step is to delete the (now blank) original inputs sheet, the pasted inputs at the bottom of the workings sheet, and the redundant names in the Ctrl+F3 Insert, Names, Define dialog box.

Complete the sequence by making a copy of the new inputs sheet. Both of these contain sheet level names.

Using multiple inputs sheets

Whether you have used cell references or range names, you should now have a model with two inputs sheets. You should now change the content of each sheet to reflect the scenario it is intended to represent, and you can change the sheet tab name accordingly.

If you inspect the formulae on the workings sheet, you can immediately identify which scenario is being used, because the formulae contain the relevant scenario sheet names.

	A	B	C	D	E	F	G
10							
11	Production						
12		TBD		=Expansion!ProductionIn		4	6
13		days		365	365	365	365
14		barrels/year		0	1,460,000	1,460,000	2,190,000

The scenario formulae are now self-documenting

To flick from one scenario to the other, simply run the Ctrl+H Replace command again. If you want to set up multiple scenarios, copy one of the inputs sheets as many times as required. If you want to document the scenario name on the outputs, you can set up a

cell on the workings which will read the scenario name from the appropriate inputs sheet

```
=BestCase!ScenarioNameIn
```

This method is arguably better than the CHOOSE/LOOKUP approach in that the workings formulae themselves describe the scenario being used. The Edit, Replace command is simple to run, and can be easily driven by macros (see Chapter 7).

As each input sheet has the same layout it may be difficult to see what is changing from one scenario sheet to the next, so the use of colour and cell comments is helpful in this context. The issue of rearranging, modifying, and removing scenarios is far simpler to manage than is the case with the CHOOSE/LOOKUP functions.

Printing scenarios

We are often required to print reports in which we compare two or more scenarios. The problem is that Excel can only calculate one scenario at a time, which is not much use. One option, is to have multiple versions of the same file, but this is inherently unreliable as any amendments to one file then need to be replicated in all the others.

Before looking at possible solutions, make sure that the name of the scenario is embedded in the worksheet. Do not rely on putting it into the page header or footer, because in the heat of the moment users forget to change the settings. We have previously introduced techniques which can be used here, using range names or concatenation, depending how the scenario is driven. With the CHOOSE and VLOOKUP type routines, simply set up a formula that will return the appropriate scenario name depending on the input number or selection from the dropdown list, for example:

```
=CHOOSE(ScenarioNumber,"Best Case","Most Likely","Worst Case")
```

Give this cell a name, such as ScenarioName, and put a link to this on each of the outputs sheets/ranges.

The concatenation trick (see page 115) is used to set up the sheet title so that it includes the scenario name:

```
="Cash Flow:"&ScenarioName
```

If we are using the multiple input sheet scenario method, set up cells on each input sheet which contain the scenario name, and give each cell the same name, using the sheet level name technique, for example, Base!ScenarioIn, MostLikely!ScenarioIn, and so on. On the workings sheet set up the usual link to the current scenario, such that when the Edit, Replace command is run the range name picks up the correct scenario name.

The solution to the problem of printing scenarios is to decide if the results are to be stored permanently in the model or if they are to be discarded after printing. In the latter case it is then simply a process of running the scenario, printing, resetting and running the next scenario, a process that calls out for macro automation. In the former case, where the results are to be stored, or perhaps even displayed alongside each other, we need to think about copying sheets with pasted values. I would always be cautious when thinking about

generating large numbers of sheets, for despite their apparent simplicity and limited content they can increase the size of the file quite considerably.

1 Set up the outputs as required, with appropriate sheet tabs and page settings.
2 Run the scenario.
3 Make a copy of each sheet, either by Ctrl + click and drag, or by grouping the sheets and right-click Move or Copy.
4 Use Ctrl + A to select the whole sheet, then Ctrl + C (Copy) and then Edit, Paste Special, Values over the formulae.
5 Repeat 2–4 for each scenario.

We can adapt the Paste Values technique if the key results from each scenario need to be presented on the same page.

Bear in mind that if the model is changed in any way the results are instantly outdated and the scenarios will need to be run again. This naturally points us towards setting up some macro functionality to achieve this.

Solver

Excel has an add-in called Solver. This tool allows a Goal Seek functionality but with multiple input cells (up to 200) for which you can specify constraints, to calculate a single dependent formula. As my intention has been to offer modelling solutions using native Excel tools, I will not go further at this point, other than to offer the usual caveat that Solver is effectively a black box and that the assumptions underpinning the input cells are not documented in the worksheet. The precedent cells used by Solver must be on the same sheet as the target formula, which would breach the inputs–workings–outputs structure in the same way as do the data table and the Scenario Manager techniques.

The way I normally deal with Solver is to explain that if you do the sort of modelling that requires the use of Solver, you probably do not need me to explain how to use it; and therefore if you do not do this type of modelling you are unlikely to need it!

Risk

Risk is a topic that is much abused in financial modelling and a proper exploration of the subject is outside the scope of this book. However, we do encounter the basic ideas of risk analysis in our modelling, and we should be able to understand why risk modelling is not as straightforward as it might seem. When carrying out sensitivity analysis, the presumption is that each of the values in the data table is equally likely. Using our professional judgement and experience we will of course recognise that this is not the case, and that within the population of values used in the table some are indeed more likely than others. When we start assigning probability to these values we are starting to think about risk. The exact definitions vary, but risk implies that all outcomes can be predicted and that we can predict the likelihood of one outcome over another. If a risk factor has a value of zero

then it will never happen, and a factor with a value of 1 is guaranteed to happen. When assigning risk to a set of values, for example in a data table, the total of the probabilities must equal 1. If it is less than 1, then there is an outcome which has not been predicted, and this leads into the concept of uncertainty. If we have not identified all the outcomes that might arise, or we are not able to assess the likelihood of their occurrence, then we are dealing with uncertainty.

Where do we obtain the information from which to derive our probabilities? With operational factors we may be able to draw on significant previous experience, for example, that we incur a 3% wastage rate in the manufacture of a type of biscuit, but with financial factors this becomes much less precise – what exactly is the interest rate risk? The risk modellers refer to objective and subjective probability – the former is based on past experience, the latter on expert opinion. We can improve the wastage rate at the biscuit factory if the food scientists apply their knowledge of the interaction of flour, water, fat, sugar, and heat to the manufacturing process – the results will be measurable and reproducible. But what about the interest rates? In the United Kingdom advertising for personal financial products always carries the strapline to the effect that 'investments may go down in value, as well as up'. What will the interest rate be next month? Next year? In 10 years?

Another point is that certain sets of figures do not show the features of random variation and therefore are not subject to chance or probability. Staying with the biscuit factory for a moment, the production capacity does not fluctuate in itself. Capacity is a variable (described as deterministic) in that we can increase or decrease production levels, but we would not claim that there is a 10% chance of production exceeding 1,000 boxes/day. It is the management who make this decision. Although the production capacity is deterministic, the appetite or demand for my biscuits is highly variable (described as stochastic), and in consultation with marketing colleagues we may be able to assign probabilities to the level of demand for the product over a particular forecast period.

When looking at carrying out any form of risk analysis careful thought needs to be given to relationships: if we are going to run a sensitivity or other analysis on a factor, are there any related factors which would be affected? The classis example is the relationship between interest rates and inflation, and we have seen models in which one has been tested independently of the other. Similarly, does inflation have an equal effect on revenues and on costs? Before carrying out any such analysis we must make sure that the correlations and dependencies of the elements of the model are fully understood if the results are to have any value.

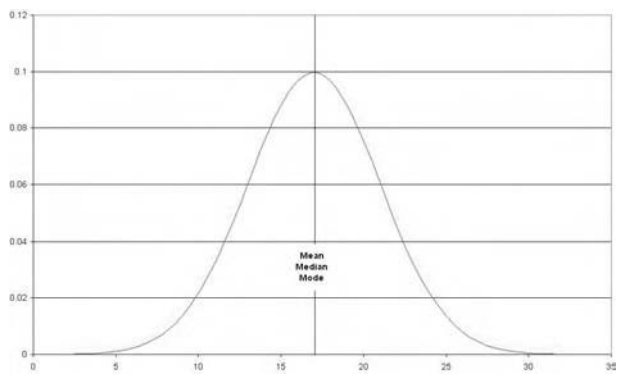
Monte Carlo simulation

For some analysts, Monte Carlo simulation is the acme of the modelling process. We use third-party software to perform this analysis, the two leading products being @RISK from Palisade,* and Crystal Ball from Decisioneering.** The analysis requires the modeller to

*http://www.palisade-europe.com/
**http://www.decisioneering.com/

define each input in terms of a statistical population and there are many populations to choose from. When the simulation is run, the software generates a random number from within the population, enters it into the input cell, recalculates the model, and stores the results. It then repeats the process up to several thousands of times, for each input specified. In this way the model can simulate the millions of permutations that can result from the interaction of the selected inputs. In my opinion this is fine for scientific and engineering purposes, but of little use for financial modelling. In simple terms, the financial variables we use in our modelling do not fit into the population distributions used by the software. Although they may have a stochastic appearance, with randomness of sorts, the only meaningful constraints that can be applied relate to the triangular distribution of lowest value-most likely value-highest value. Unfortunately, the very ease with which the software allows the modeller to set up the Monte Carlo simulation, leads us into problems.

Most people have had little exposure to statistics other than at college or at university, and most such courses are based around the statistics of the normal distribution.

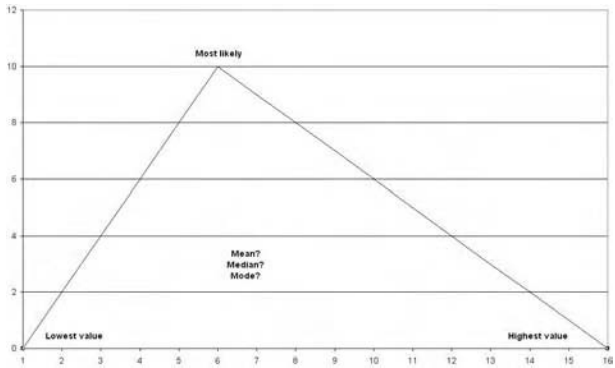


Statistics 101: the normal distribution

From this we learn expressions such as:

1 *Arithmetic mean*: The sum of all the observations in a sample, divided by the number of observations in the sample.
2 *Mode*: The most frequent, or common, value in a sample.
3 *Median*: The middle value of a sample.
The blanket expression 'average' is often used to with reference to these three definitions.
4 *Standard deviation* is used to describe the spread of the numbers in the sample about the mean (or more correctly, the square root of the arithmetic mean of the squares of the deviations from the arithmetic mean). One standard deviation of each side of the mean will include 68% of the sample, two standard deviations covers 95%, and three standard deviations covers 98% of the sample.

The moment we consider a different population distribution we realise that these terms no longer have the same meaning and must be used with much more care.



The triangular distribution: best/worst/most likely. If you must define a distribution, this is probably the easiest to justify

But when looking at models using the Monte Carlo technique we often find a casual disregard for what is very specific terminology and indeed we often find that the analyst has chosen the wrong population distribution, failed to identify the correlating factors, or has assigned a distribution to what should be a deterministic element. Quite often we note that Monte Carlo analysis has been performed simply because the software was available, and the analyst thought that it would add value to the model. There are indeed certain kinds of model in which specialist risk techniques are required, and risk modelling is a specialism in itself, but our general recommendation is for the general modeller to be aware of the techniques but to use them with caution.

CHAPTER 7

Automation

Introduction

Macros are predefined sequences of commands and events that the user can run as required, and are typically used to execute repetitive or complex tasks. They can either be recorded, which requires the user or developer to carry out the appropriate command sequence first, or they can be written using the Visual Basic Editor in Excel. Visual Basic is the macro command language used in Microsoft products.

In the early days of spreadsheets, the first macro commands were easily learned and applied by the expert spreadsheet modeller. Lotus 1-2-3 release 3, for example, had some eighty commands. It was a straightforward task to both write and debug the code. With the introduction of Windows the situation grew rather more complex. Both Lotus and Microsoft introduced macro languages (Lotus Script and a form of Visual Basic, respectively) which were considerably more advanced, and allowed for the creation and use of dialog boxes, as well as offering the ability to interrogate other applications for information. Although these languages and their successors could be learned by the expert user, their inherent complexity meant that the writing of macro code became a task better suited to the programmer, rather than the modeller.

However, some modellers believe that a model is not finished unless it has a library of macros. Certainly there are times when the use of macros is unavoidable, but using the principle of error reduction, their use should be limited. Programmers are trained to write code, and have systems and methodologies concerning specification, development, documentation, and quality control. Financial analysts and modellers tend not to have this discipline, and the archives of the investment banks are full of half-finished macro modules which were begun with the best of intentions but the developers lacked the time or expertise to see the job through to completion.

The worst type of macro-driven model is what we call the 'black box'. On opening the file, the user is confronted with an attractive 'user input screen', from which selections can be made or data entered. On clicking the OK button, the screen freezes and after a few moments the nearest printer starts spewing out paper. In the meantime, the user suspects that the macro may have ground to a halt and starts hitting the Escape key. Up pops a dialog warning that 'Code execution has been interrupted'. Should you continue, end, or debug? If you end, what has happened in the model so far – what is the effect of Undo at this point?

The demonstration workbooks for this chapter are in the Chapter 7: Automation folder on the CD-ROM.

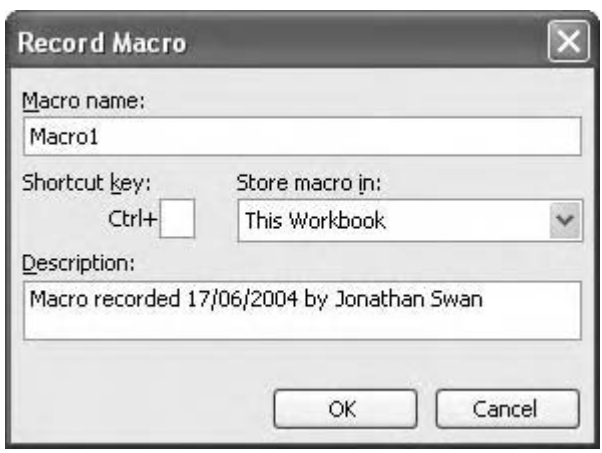
Recorded macros

The simplest way to get started with macros is to record a straightforward command sequence and then to inspect the resulting code in the Visual Basic Editor. The starting point, as always, is to decide what exactly the macro is supposed to do and then to write down the steps you will need to carry out. Remember the top-down methodology introduced in Chapter 1?

Recorded macro options

The command for recording a macro is Tools, Macro, Record New Macro. The dialog box contains the following options:

1 Name: use the range name convention of compound words – Excel will not accept spaces in macro names. Please develop the discipline of providing meaningful names, even if you are just experimenting.
2 Shortcut key: Ctrl+ - enter a keyboard shortcut used to run the macro. Hold down the Shift key for further permutations. Note that this shortcut will over-ride any existing Excel shortcut in this workbook.
3 Store macro in: this offers three choices
   a This workbook (default): the macro can only be run whilst this workbook is open; it could be used in another workbook if both are open at the same time.
   b Personal macro workbook: the macro will be stored in a workbook called PERSONAL.XLS. This is a hidden file which loads up when you launch Excel. It is stored in your XLSTART folder. The macro will then be available in any workbook that you then open or create. However, if you attempt to edit the macro (see below), Excel will warn you that you cannot edit a macro in a hidden workbook. You must use the Window, Unhide command first.
   c New workbook: means exactly that.



Not much information provided here. Develop the habit of documenting your macros from the start

4 Description: as with the macro name, develop the discipline of noting the purpose and function of the macro. You can use up to 255 characters. This information is stored in the macro header which you will see when we open the Visual Basic Editor shortly.

Iteration macro

Let us revisit the Iteration command sequence used in Chapter 3. We used iteration to calculate a routine that involved a circularity, and we noted that it is not advisable to leave iteration switched on all the time as it masks any accidental circularity. We set up a mechanism such that if the iteration was on, we set the Switch to TRUE and this activated the circular formula. When iteration was switched off, we set the Switch to FALSE and this suspended the circularity.

Let's identify the steps involved in switching the Iteration on, and then switching it off:

1 Activate the sheet that contains the Switch cell.
2 Go to the Switch cell.
3 Run Tools, Options, go to the Calculation tab and click on the Iteration check box, then OK.
4 Enter TRUE into the Switch cell.

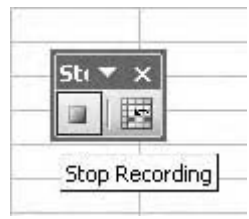
Next,

5 Activate the sheet that contains the Switch cell.
6 Go to the Switch cell.
7 Enter FALSE into the Switch cell.
8 Run Tools, Options, go to the Calculation tab and click off the Iteration check box, then OK.

In both cases, the first step is to activate the sheet that contains the Switch cell. This is because the user may run the macro from any sheet in the workbook. If we have not identified the relevant sheet, the macro may run on another sheet, with no result. We also select the Switch cell for a similar reason – we do not want the TRUE or FALSE dropping into the active cell if it is on another sheet.

I have also changed the order of events slightly. The ON macro switches on iteration before it changes the Switch cell contents. If the TRUE was entered first, Excel would complain about the circularity and generate an error message. Likewise, the OFF macro inserts the FALSE into the Switch cell before switching off the iteration. (In reality the errors would not appear during macro execution, but it shows that we are thinking the matter through).

Now we are in a position to carry out the command sequence and record it. In some cases it is worthwhile running through the sequence once or twice to rehearse it – bear in mind that any digressions or errors will also be recorded. This will lead to messy code, and it could slow down the execution of the finished macro. For some reason the Stop Recording toolbar does not always appear when you start recording a macro, so you may find it helpful to use View, Toolbars, Customize and choose the Stop Recording toolbar. Do it now, rather than recording it into your macro!



The Stop Recording toolbar

1 Use Tools, Macro, Record New Macro.
2 Give the macro a name such as IterationOn.
3 Assign a shortcut if you wish.
4 Store the macro in this workbook.
5 Write a description for the macro (Recorded macro, for switching on the iteration to calculate a circular routine).
6 Click OK.
7 Press F5 (Go To) and type in the destination Switch (assuming you have set up the appropriate name previously). Click OK.
8 Use the Tools, Options command and switch on Iteration on the Calculation tab. Click OK.
9 Type TRUE into the Switch cell, and press Enter.
10 Stop the macro recorder by either clicking on the Stop Recording button or by using Tools, Macros, Stop Recording.

Repeat the process to record the macro to turn off the iteration.

1 Use Tools, Macro, Record New Macro.
2 Give the macro a name such as IterationOff.
3 Assign a shortcut if you wish.
4 Store the macro in this workbook.
5 Write a description for the macro ('Recorded macro, for switching off the iteration to suspend a circular routine').
6 Click OK.
7 Press F5 (Go To) and type in the destination Switch (assuming you have set up the appropriate name previously). Click OK.
8 Type FALSE into the Switch cell, and press Enter.

9 Use the Tools, Options command and switch off Iteration on the Calculation tab. Click OK.
10 Stop the macro recorder by either clicking on the Stop Recording button or by using Tools, Macros, Stop Recording.

You can test the macros by using Tools, Macro, Macros (Alt+F8), and selecting them from the list, or by using your keyboard shortcut, if you assigned one. See below for running macros from buttons and menus.

Assigning macros

Macros can be run by using keyboard shortcuts, worksheet buttons, toolbar buttons, or from a menu. Before deciding which technique to use, you should consider where the macro has been stored. If the macro is in your personal macro workbook it will be available in all workbooks and can be assigned to menus and buttons. If the macro is stored in a specific workbook, it will only be available if that workbook is open, and the macros should be assigned to keyboard shortcuts and/or worksheet buttons only; if you have assigned them to the toolbars or to a menu the commands may not work if the workbook is closed or not available. Excel will attempt to open the file containing the macro but problems will occur if the macro requires specific structures and features in the workbook.

Keyboard shortcuts

On recording a macro you will be invited to assign the macro to a shortcut before proceeding. The shortcut automatically includes Ctrl, and is followed by a letter. Further combinations are allowed, holding down Shift+letter (do not press Ctrl). The use of Alt+ is not permitted.

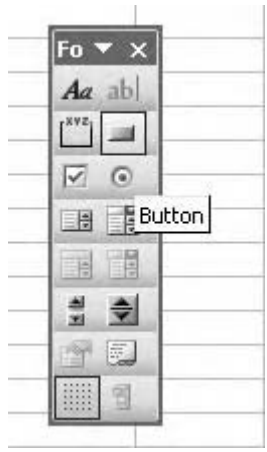
If you recorded the macro without choosing a shortcut, or wish to assign new shortcut to an existing macro, use the Tools, Macro, Macros (Alt+F8) command and choose Options. This allows you to specify the shortcut required.

Note that the shortcut assigned to the macro takes precedence over the default Excel shortcuts; even if you are not a shortcut user your colleagues may be, and it can be frustrating (and dangerous) to find that a favourite shortcut such as Ctrl+P (print) or Ctrl+S (save) has apparently been hijacked for some other purpose. This especially applies to shortcuts for macros in your personal macro workbook, which are available for any and all subsequent workbooks.

It is good modelling practice to include a list of keyboard shortcuts in the documentation of your model.

Workbook buttons

Workbook buttons are graphic objects stored in the workbook. To create a button, right-click on any toolbar and choose the Forms toolbar. Locate the Button button, and click.



The Button button on the Forms toolbar

The mouse pointer changes from the arrow to a crosshair: you can either simply click in the worksheet for a default button, or click and drag to draw a button of your own sizing. Hold down Alt while dragging to 'snap to' cell gridlines. On doing so, the Assign Macro dialog box appears, from which you can select the macro to assign to the button.

Note that the button is selected and has edit handles; whilst this is the case you can edit the button text and by right-clicking carry out other commands from the shortcut menu. If you lose the selection, perhaps by clicking back in the worksheet, DO NOT left-click the new button – this will only run the macro. To work with the macro button, right-click it first.

Edit the button text as required, and use the edit handles to resize or reposition the button. It can be helpful to name the button. Use Ctrl+F3 (Define Name) or alternatively click in the Name Box and enter a name there.

If you have several buttons, it can be easier to manipulate them if they are grouped together. You will need to show the Drawing toolbar, and click on the Select Objects button. Click and drag a marquee around the buttons and select the group. Edit handles should appear around all selected buttons (note that Ctrl+clicking does not work in this context). Right-click on any button and choose Grouping, Group. The grouped buttons can now be edited and formatted together. Buttons can be ungrouped by right-clicking and using Grouping, Ungroup. Having been grouped previously, individual buttons can be regrouped using Grouping, Regroup.

Toolbar buttons

If your macro is to be available to other workbooks, you can assign it to a custom button on a toolbar. Use View, Toolbars, Customize, or right-click an existing toolbar and choose Customize from the shortcut menu. Click on the Commands tab in the dialog box, and scroll down the list to Macros. Click and drag the smiley face Custom button onto an existing toolbar. With the button selected, either click the Modify Selection button or right-click the new button itself. The first step is to give the button a name – this name (or

description) appears in the ToolTip when the user points at the button. The & ampersand character offered by default is not necessary and can be deleted. Next click Assign Macro and select the macro to be associated with the button. Then the fun starts:either Change Button Image, and select a graphic from the list; or Edit Button Image to launch the Button Editor. You can design the image for your button at a pixel level.

Menus

Macros can be assigned to menus as required. You can use either the existing menus, or set up a new one. Note that a new menu can be assigned to the main menu bar, or docked into a toolbar.

Use the View, Toolbars, Customize command. Scroll down to New Menu, and click and drag the New Menu button onto either the main menu bar or to a toolbar. Right-click on the new menu and enter a name for the menu. Note the use of the & ampersand character – this provides the underline to a character so that the menu can be pulled down using the keyboard, for example, Iter&ation would appear as Iteration; this could be accessed using Alt+A (check that the letter you use does not conflict with any of the existing menu shortcuts). You could assign a macro directly to this new menu, but it makes more sense to add further menu commands by dragging the New Menu button from the Customize dialog box. Set up appropriate names and assign the macros as required.

Bear in mind that custom menus and toolbar buttons will reappear when you next open Excel; unless the associated macros are stored in your personal macro workbook you are likely to experience errors if the menus or buttons are used in the wrong context.

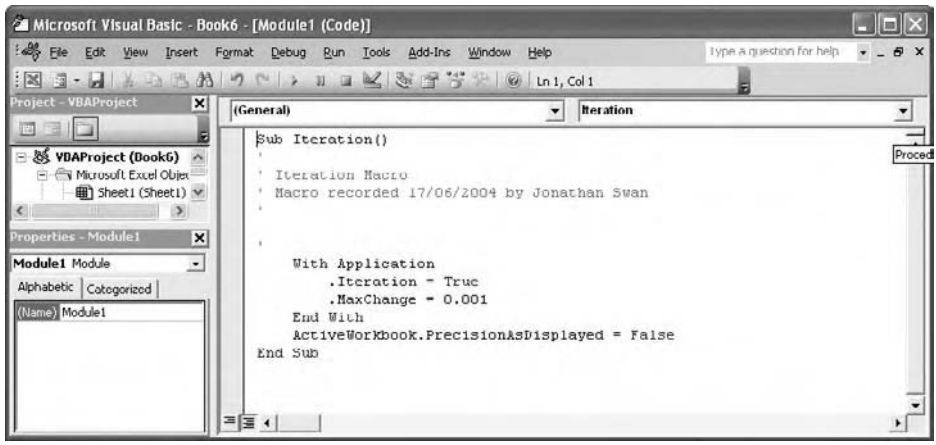
Written macros

A basic understanding of macro code allows more complex macros to be written, and allows access to the many macro commands and routines that are simply not available through recording. However, writing Visual Basic code can be daunting and difficult; in this section we will consider some of the basic concepts which might then lead into more complex code*.

Once you have recorded a few macros, you may find that the Visual Basic Editor serves as a useful introduction to writing code. The Tools, Macro, Macros, Edit (Alt+F11) command launches the VB Editor, allowing the underlying code of the macro to be inspected. (If you chose to store the macro in your personal macro workbook, you will need to unhide this file first, using the Window, Unhide command).

One feature of the recorded macro is the inclusion of spurious code; for example, in the literation macro recorded previously, the command sequence was Tools, Options, Calculation, and Iteration on/off. The VB code records two additional commands, that the Maximum Change constraint of iteration has a value of 0.01, and that the Precision As Displayed command is FALSE.

*Although not essential, you may find it helpful to check that you have Help for VB installed.



The Visual Basic Editor

Where did these two instructions come from? Why did not the macro record all the options on the Calculation tab? You can safely delete these two lines of code without affecting the macro. In fact, the With Application section can also be removed, and the Go To sequence simplified. The final macro is simply:

```
Sub IterationOn()
  Application.Iteration = True
  Application.Goto "Switch"
  ActiveCell.Formula = "TRUE"
End Sub
```

You can learn a fair amount of VB using recorded macros and some trial and error. If you have cut out too much from the code, Excel will throw up error messages and return you from the workbook to the VB Editor. The error dialog box may not be particularly informative, but usually the Editor will highlight the offending code. If you can supply the correction, you can then press F5 to continue with the execution of the code. Alternatively, you may be able to cancel the macro using Ctrl+Break. If the macro does not appear to work in the manner you think it should, you can enter Step mode using F8. Each time you press the key, one line of the macro is carried out.

It is good practice to document your work. Comments can be inserted anywhere in and around the macro, as long as you use the single apostrophe ' prefix.

Branching macros

The iteration exercise serves as a useful introduction to written macros – we might want to combine the functionality of the two separate macros into just one – if iteration is off, switch it on, and if it is on, switch it off.

We considered the use of the IF function in Chapter 4: we specify the test condition, the outcome to show if the condition evaluates to TRUE, and the outcome when the condition is FALSE. We can write the same functionality into the macro, using an IF THEN ELSE sequence.

As with the recorded macro, it is good practice to write down what you want the macro to achieve and the steps required. In this case:

1 If the iteration is OFF,
   a Switch iteration on;
   b Set the Switch to True,
2 If the iteration is ON,
   a Set the Switch to False;
   b Switch the iteration off.

As we will run this macro from a worksheet button, we will have the button itself confirm the iteration status, so in each case above we will have a third step in which the text on the button changes.

Writing the macro

Use Tools, Macro, Visual Basic Editor (or Alt+F11 but caution – Shift+F11 inserts a new worksheet). Use the Project Explorer window to check where the current module will be stored (current workbook or personal workbook).

In the code window, write the following:

Command	Explanation
	Comments can be entered into the macro if they are prefixed with ' (single apostrophe)
Sub IterationSwitch()	'Name of macro
IterationValue=Application.Iteration	'Returns the current content of the Switch cell
If IterationValue = TRUE Then	'Test condition
ActiveSheet.Shapes("IterationButton").Select	'Selects the workbook macro button
Selection.Characters.Text = "Iteration is OFF"	'Changes the text on the button
Application.Goto "Switch"	'Select the Switch cell
ActiveCell.Formula = "FALSE"	'Enters FALSE in the Switch cell
Application.Iteration = False	'Switches off the iteration
Else	'If SwitchValue = False
ActiveSheet.Shapes("IterationButton").Select	'Selects the button
Selection.Characters.Text = "Iteration is ON"	'Changes the text
Application.Iteration = True	'Switches on the iteration
Application.Goto "Switch"	'Selects the Switch cell
ActiveCell.Formula = "TRUE"	'Enters TRUE in the Switch cell
End If	
End Sub	

I am assuming that this macro will only be run from the button on the sheet; if it is likely that it could be run from a keyboard shortcut, toolbar button, or menu, we would need to include an instruction to go to the appropriate sheet. Note that the location of the button itself is not an issue, provided that you have named it IterationButton (use the Name Box).

Now test the macro and its operation in the workbook. If you have errors, use the F8 Step and F5 Resume commands as required.

Quarterly/annual macro

In Chapter 3 we looked at techniques to set up our outputs in such a way so that we could either report on an annual or a quarterly basis. With either the rolling sum or the corkscrew technique we have to hide the Q1, Q2, and Q3 columns, which can prove rather time consuming. We can automate this process with a simple written macro. This macro also introduces the use of the ScreenUpdating property, which allows us to stop the screen from redrawing while the macro runs.

```
Command                                          Explanation
Sub QuartersToAnnual()
'Quarterly to Annual Macro
'Macro written 25/05/2004 by jswan               'Usual documentation
'Converts a quarterly report to an annual report
'by hiding each Q1-Q3 column,
'then unhides row headings
Application.ScreenUpdating  = False              'Stops the screen
                                                 flickering while the macro
                                                 runs
For Each Col In Worksheets("Sheet1").Columns     ' Substitute sheet names
                                                 as required
   If Col.Column Mod 4 <> 0 Then                 'If the column number is
                                                 not exactly divisible by 4
      Col.Select                                 'Select the column
      Selection.EntireColumn.Hidden = True       'Hide the column
   End If
Next Col                                         'Repeat for next column
   Range("A1:D1").Select                         'Select first four columns
   Selection.EntireColumn.Hidden = False         'Unhide the columns
   Range("A1").Select                            'Move active cell to A1
End Sub
```

Error handling

VBA is now a very robust tool and will trap errors as they are written, or on execution. As mentioned above, the step mode and other debug features of the VBA editor are useful,

provided you can understand what Excel is telling you! In some cases it can be helpful to build your own error handling routine if you want to test your code for yourself, or to trap events like the user pressing the Escape key whilst the macro is running.

This example also introduces a call to a simple subroutine and uses a message box.

```
On Error GoTo Error_Report                    'Code at top of macro,
                                              which will run the
                                              Error_Report subroutine
                                              on error
Application.EnableCancelKey = xlErrorHandler  'This recognizes the
                                              Escape key
CODE                                          'This is your macro
Exit Sub                                      'This ends the macro if
                                              no errors occur
If Err = 18 Then                              See if we can write a
                                              resume routine
  MsgBox "You pressed Escape"
Error_Report:                                 'Start of subroutine; note
                                              the colon:
MsgBox "Error: " & Err.Number & " " &         'Displays a message box
Err.Description                               with the error number and
                                              description.
End Sub                                       'The macro terminates
```

User-defined functions

With its various add-in packs and third-party add-ins Excel can offer several hundred functions, which is still not enough for some users and so Excel allows us to write our own. As with macros I am generally cautious about the use of user-defined functions (UDFs) as their management can be a problem, especially if the function does not reside in the workbook itself. I have seen too many models in which the UDF is resident on the analyst's PC back in the office. Do make sure that the UDF is stored in the workbook, and bear in mind that the recipient's macro security settings may not accept it (Excel 2003/XP). I have also noticed that writing UDFs can be addictive and that some people end up creating functions that effectively duplicate existing Excel functionality. Bear in mind from the outset that functions return values – they cannot carry out instructions to perform tasks. In this section we will consider some very simple examples of UDFs, but without going into very much detail. Remember that if you are looking for help about UDFs, use the Visual Basic Help and search for 'Function Statement'.

Random numbers

In Chapter 4 we looked at the use of the Excel RAND function to return a random number between two values, such as between 10 and 20.

```
=RAND()*(20–10)+10
```

We can write a UDF to perform this operation.

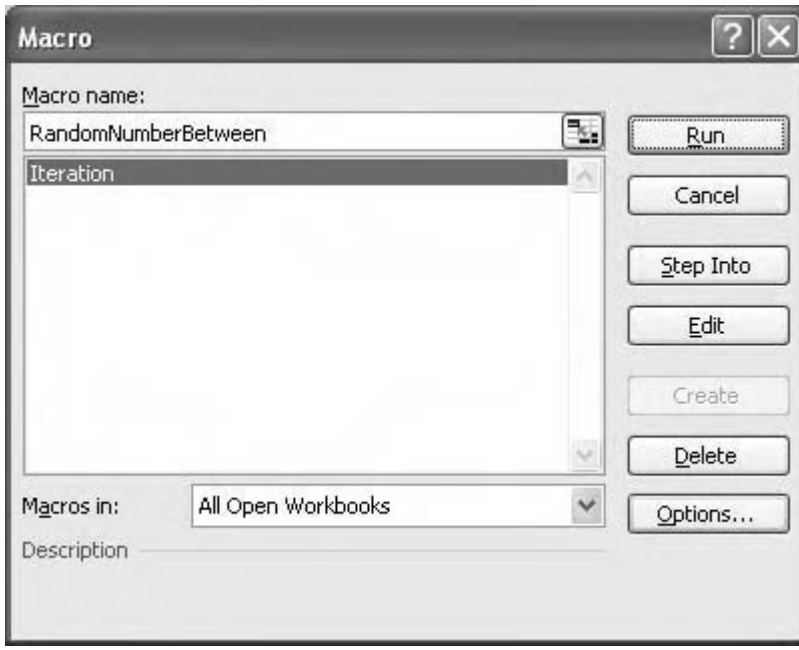
1 Use the Tools, Macro, Visual Basic Editor command (or press Alt+F11).
2 Use the Insert, Module command.
3 Write the following code:
```
Function RandomNumberBetween(lower As Integer, upper As Integer)
RandomNumberBetween =Rnd()* (upper−lower) + lower
End Function.
```
4 Alt+Tab back to the worksheet and write the function in a cell:
```
=randomnumberbetween(10, 20)
```
5 Test the operation of the function. Note that it does not recalculate automatically (or on pressing F9). If you want to fix this, insert the following line after the function definition: Application.Volatile True
6 Test that this amendment works. You may need to press F2 to edit the function in the worksheet first.

The function name is RandomNumberBetween, and we have specified that the required arguments are the lower value (as an integer) and the upper value (as an integer). Excel then generates a random number using the VB Rnd function (as opposed to the worksheet RAND function – it took me a while to work that one out), and performs the arithmetic to ensure that the number is between the upper and lower values. The Volatile instruction makes Excel run the function on recalculation. Note that despite the use of upper case in the function name in the VB editor, the worksheet uses lower case only. You will have noticed that as you write the function, Excel will attempt to fill in key words and capitalise according to its own rules. As with macros, if you make errors you can use the F8 Step mode to run through the function. Remember to reset after debugging any mistakes.

If you save the file which contains your UDFs with a name such as functions.xls, you can access these functions from your other models by prefixing the function name with the (path and) filename:

```
=functions.xls!randomnumberbetween(10,20)
```

You can also write the function by using the Insert, Function command, where it will be listed in the User Defined category. It will not contain a description, but you can fix this by using Tools, Macro, Macros. The dialog box only lists macros, but if you type the function name you should be able to click on the Options button. Write the appropriate description for the function, and close the dialog box.



The user-defined function is not listed, so you must type in the function name

Local range names

One of the problems with using range names is when the user accidentally creates local names which conflict with existing global names. In the modelling methodology set out in the first part of this book, I suggested that names on the inputs sheet should have the suffix '…In' to differentiate the input names from those on the workings sheet. If, however, the 'In' is omitted, and the same range name is created on the workings sheet, the latter name becomes a local or sheet level name and can only be referenced on that sheet. To refer to the range from another sheet, we must include the sheet name along with the range name (e.g.=Workings!UnitPrice). This can lead to problems – the Paste Name and Go To commands do not identify conflicting names and list only the names on the current sheet, which may then be a mixture of global and local names. The following UDF attempts to resolve this, and is used with the Paste List command.

```
Function LocalNameCheck(TestName As String, Reference)
NameVerify = Names(TestName).RefersTo
IF NameVerify = Reference then
    LocalNameCheck  = 0
Else
    LocalNameCheck = NameVerify
End If
End Function
```

The function is written into the worksheet alongside the list of names. The TestName is the cell reference of the range name in the list, and the Reference is the range reference. NameVerify returns the global name reference, and the IF checks that it matches with the reference from the list in the worksheet. If they are the same, the function returns zero, but if they are different (because the name in the worksheet references a local range name) the function returns the conflicting global range reference. The function works reliably in the inputs/workings model structure if the check is run on the workings sheet, but is less reliable in other model layouts.

Appendix

Keyboard shortcuts

This is a list of the keyboard shortcuts I think are most useful for routine modelling. Refer to Excel Help for a full list, using "keyboard shortcut" as the search string, or read the Using Shortcut Keys topic in the Contents of Help.

Toolbars

Press F10 to activate the main menu bar and then Ctrl+Tab to cycle through the toolbars, using the arrow keys to move to the buttons and Enter to select.

General

Ctrl+Z	Undo
Ctrl+Y	Redo
F2	Edit Cell
F4	Repeat last action
Ctrl+R	Fill Right
Ctrl+S	Save
F12	Save As
Alt+=	Autosum
F11	Create Chart
Ctrl+Shift+"	Copy cell above
Ctrl+;	Insert date
Shift+F10	Shortcut menu

Selection

Ctrl+A	Select sheet
Shift+Space	Select row
Ctrl+Space	Select column

Shift+arrow	Select in direction
Ctrl+*	Select current region

Navigation

F5	Go to/Go back
Ctrl+PgUp	Next sheet
Ctrl+PgDn	Previous sheet
Ctrl+Home	Go to A1/top left of sheet
Ctrl+End	Go to bottom right of sheet

Auditing

Ctrl+[	Select precedents
Ctrl+]	Select dependents
Ctrl+`	View formulas (toggle)

Formatting

Ctrl+Shift+1	Comma, two decimals
Ctrl+Shift+4	Currency, two decimals
Ctrl+Shift+5	Percentage, no decimals
Ctrl+Shift+~	General number format
Ctrl+#	Date format
Ctrl+B	Bold
Ctrl+I	Italic

Group and outline

Alt+Shift+right arrow	Group rows (or columns)
Alt+Shift+left arrow	Ungroup rows (or columns)
Ctrl+8 (not number keypad)	Display or hide outline symbols
Ctrl+9 (not number keypad)	Hide rows
Ctrl+Shift+(	Unhide selected rows
Ctrl+0 (not number keypad)	Hide columns
Ctrl+Shift+)	Unhide columns

Names

F3	Paste names
Ctrl+Shift+F3	Create names
Ctrl+F3	Define names

The Principle of Error Reduction

The Principle of Error Reduction accepts that errors are inevitable. Some techniques are more prone to error than others. We reduce the risk of error by using alternative techniques and a consistent methodology that serves to enhance the detection of errors when they occur.

The Rules of Good Modelling

The 'rules' and statements in this section are derived from various sections of this book where they are discussed in detail. I summarise them here to offer a stimulus for discussion.

Structure: general

- Building models: design the output first
- Using models: identify the outputs first
- Separate inputs from workings and outputs
- Use the same sheet layout throughout model (each column has the same function on each sheet)
- Use cell comments
- Use colour consistently
- Use colour for incomplete/temporary formulae
- Create a 'hard edge' for the right hand edge of the model
- Document your work
- Make navigation simple and straightforward
- Garbage in, garbage out

Inputs sheet

- Numbers only – no formulae (except data tables)
- Range names should have the suffix '…In' or similar, to indicate their origin
- Document the sources of your assumptions – especially any 'plug' numbers
- Cross-check your inputs against your data sources
- Keep numbers in the same units that are given in the documentation
- The input sheet drives the timing of events in the forecast period
- Multiple inputs sheets are acceptable

Workings sheet

- Only one workings (calculations) sheet
- Formulae only – no numbers
- Use links to bring the data from the inputs sheet(s)
- Left-to-right consistency – formulae should be the same across the whole forecast period
- Use the base column to preserve the rule of left-to-right consistency
- No 3-D calculations – no calculations which include references to other sheets

- A general increase in complexity from top to bottom
- Use basic number formatting
- Keep formulae simple and short
- Sign – liabilities should be positive
- Use group and outline techniques to keep the workings manageable

Outputs sheets

- Contains links to workings sheet
- No links to other output sheets
- Summary formulae only
- No values
- Use appropriate number formatting
- Use consistent sign convention
- Use graphs to "tell the story"

Range names

- Use consistent and meaningful names
- Remember that many people do not like and distrust names

Audit sheet

- A rule-based methodology can only work if you have the techniques to detect and locate exceptions (Chapter 2)
- 3D audit calculations are acceptable – audit check formulae are *about* the model, not part of it
- Crosscheck workings calculations with the equivalent summary formulae on the outputs sheets
- Crosscheck output sheets with each other
- Use past experience – if something has gone wrong before, think of an audit check which might have identified it and use it in your future models
- Don't move on from a piece of modelling unless all audit checks are satisfied
- 'Stress test' the model – use extreme values
- Predictive outcomes – use values which generate known results
- Use ratios as rationality checks

Printing

- Set up the page layouts as early as possible
- Be ready to print at any stage
- Complete the outputs as you go

Saving

- Save frequently, and rename/renumber after major steps
- Don't save multiple copies of the same model

- Don't distribute copies of your model
- Keep track of the versions

Formulae

- Avoid long formulae
- Keep formulae short
- Break complex logic into simple steps
- If today you can't immediately understand a formula which you wrote last week, it is too complex
- Avoid IFs: consider using MAX/MIN or the masking techniques
- Avoid OFFSET: use INDEX and MATCH functionality
- Clarify sign switching using =0-
- Avoid circular formulae; if you have to, use iteration but make sure it is controlled with a switch
- Use range names
- Do not use 3-D calculations
- Do not prefix formulae with +. Use =

File links

- Don't link to other files; if you have to, put your file link formulae on a specific sheet

Macros

- Avoid using macros; if you have to, make sure they are documented, simple to use, and thoroughly tested.
- Don't produce black boxes

Index

& character
  see ampersand character
3-D calculations
  see three-dimensional calculations
3-D names
  see three-dimensional names

ABS function, 43
AND function, 45
ALT key
  see keyboard shortcuts
alteration errors, 27
ampersand character, 115
Apply Name command, 56
arithmetical checks, 42
array formula, 40–1, 88–9
array type functions, 60
assumption sheets
  see input sheets
audit, 8–9
audit check, 33
audit sheet, 32–3
audit toolbar, 29
audit tools, 28
AuditCheck, 33
AutoComplete, 19

balance sheet, 7, 8, 43
base column, 18, 72–4
black box, 139–40, 151
blank cells, 35–6
BODMAS
  see brackets, order, division, multiplication, addition and subtraction
bottom-up approach, 2
brackets, order, division, multiplication, addition and subtraction, 71
budgeting model, 2, 9, 10

calculation errors, 27
calculations sheets
  see workings sheets
calculations, absence of, 5
calculations, time-dependent, 71
cascade effect, 6
cash flow ratio analysis, 44
cell comments, 4, 19, 114
cell reference, 30–1, 49, 143
cells, editing, 56
change checks, 46
CHOOSE function, 140–2
circularities, 79–80
  debugging, 80
  and iteration, 79
  tracking, 80
coercion, 89
COLUMN function, 125
columns, hidden, 39
combo boxes, 121
commission errors, 27
concatenation, 115–16
conditional formatting
  see formatting, conditional
control key
  see keyboard shortcuts
corkscrew, 73–5
corkscrew, loan, 75
compensating errors, 43
COUNT function, 67–8
COUNTIF function, 114, 120
  see also SMALL function
Create Names command, 53–4
  keyboard shortcuts, 53
currency symbol, 131
custom formatting
  see formatting, custom
custom views, 16–17

data entry, 19, 119–21
data tables, 136, 138
  one-way, 136–7
  recalculation, 138
  two-way, 137
data validation, 119–21
date, 111–13
DATE function, 112
date series, 112
Define Name command, 54, 59, 63, 64, 65, 66
  keyboard shortcuts, 54
dependent, 24, 141

documentation, 4, 10–11
domain errors, 28

editing, 28
error handling, 160–1
error values, 38
  #DIV/0! 32
  #N/A, 32
  #NAME? 31
  #NUM! 32
  #REF! 31
  #VALUE! 31
  for tracking circularities, 82
errors, 38
  types of, 26–8

file links, 10, 38
files
  new, 17
  saving, 17
fill, 19
financial checks, 43
financial functions, 109
financial model
  development, 17
  map, 46–7
  structure, 1, 3
  use, 117
flags, 76
footers
  see headers and footers
formatting, conditional, 123–4, 126
  problems with, 126
formatting, custom, 126
  reporting, 130
  style, 130
formulae, viewing, 29

global names, 63
Goal Seek, 135–6
grouping, 18, 117–18

hardcoded values, 34–5
headers and footers, 15–16
hidden columns and rows, 39
hidden sheets, 39
HLOOKUP, 32, 99
hyperlinks, 24–5

IF function, 90–1
INDEX function, 102–4
  and mask, 105
input sheets, 4
input sheets, multiple, 142–3
inputs
  colour, 5
  errors, 27
  location of, 34
INT function, 116
'intellectual challenge' approach, 48
internal rate of return, 45, 111
intersection formula, 61
IRR
  see internal rate of return
ISTEXT function, 114–15
iteration, 79, 82–7
iteration macro, 153
iteration status check, 41

keyboard shortcuts, 155, 165–6
  Alt + PgUp/Alt + PgDn, 20
  Ctrl + [, 24
  Ctrl +], 24
  Ctrl + Arrow, 22
  Ctrl + End, 21
  Ctrl + PgUp/Ctrl + PgDn, 20
  F2, 28
  F6, 22
  F9, 29–30
  Go To, 22
  Home/Ctrl + Home, 21
  right-click scroll tab buttons, 21

LARGE function, 113
layout, 18, 33
left-to-right consistency, 36, 71
liabilities, 20
link formula, 5–6
list boxes, 121–2
loan calculator, 2, 9
local names
  see sheet level names
logic cascade, 6
LOOKUP function, 101, 140, 142

macro buttons, 155–7
macros, 152
  assigning, 155–7
    keyboard shortcuts, 155
  writing the, 157–8
macros, recording, 152–3
macros, written, 157–8
mask technique, 75–6, 93–8
  and INDEX function, 101
MATCH function, 101
MAX function, 90
menus, 157
merged cells, 40
MIN function, 90

MOD function, 116, 125
Monte Carlo simulation, 148–50
multiple input sheets
  see input sheets, multiple

name box, 23–4
Navigation techniques, 20–5
net present value, 2, 9, 45, 109
NOW() function, 112
NPV
  see net present value
number format, 18
  see also formatting, conditional
  see also formatting, custom

OFFSET function, 67, 68, 108–9
omission errors, 27
OR function, 92
outlining, 117–18
  keyboard shortcuts, 118–19
output sheets, 7

page setup, 14–15
password protection, 133–4
pointing errors, 26–7
pool mechanism, 102–5
precedent, 24
print, 13–17, 19, 146
print area, 13–14
protection, 131–4

quarterly/annual calculation, 77
quarterly/annual macro, 160

RAND function, 114, 161
random numbers, generation of, 114, 161–2
range labels, 50
range names, 49, 144–5
  advantages of, 52–3
  converting to cell references, 58–9
  creating, 53–4
  defining, 54–5
  deleting, 57–8
  extending, 59
  and functions, 60
  listing, 68–70
  in Lotus 1-2-3, 49, 58
  moving, 59
  naming conventions, 51
  redundant, 58–9
  using, 55–6
range names, dynamic, 66–8
range names, local, 163–4
Record Macro, 152
relative names, 64–5
relative totals, 79
Report Manager, 16
reports, 13, 16
reports, management, 2, 9, 10
risk analysis, 147–8
rolling total method, 77–9
ROUND function, 44
rows, hidden, 39

Scenario Manager, 139–40
scenarios, 138–9
  printing, 146–7
Screen Updating property, 160
sensitivity analysis, 135
sheet level names, 63
sheet protection
  see worksheet protection
signing conventions, 20
SMALL function, 113–14
  see also COUNTIF function
Solver, 147
spaghetti modelling
  see three-dimensional calculations
split window, 22
stream of consciousness approach
  see bottom-up approach
structural checks, 34
SUM function, 44, 62
summary calculations, 7–8
switches, 76

text, 4
text box, 4
TEXT function, 115–16
three-dimensional calculations, 6, 37
three-dimensional names, 61–2
time periods, 71
  changing, 76
timesheeting system, 2, 9
timing errors
  see temporal errors
TODAY() function, 112
toolbar buttons, 156
top-down approach, 3, 13
Transition Formula entry, 59

UDF
  see user-defined functions
unit columns, 18
user-defined functions, 161

validation, 119–21
Visual Basic Editor, 151, 152, 153, 157–9
VLOOKUP function, 99, 142

watch window, 30
what-if analysis
  see sensitivity analysis
workings sheets, 5–7
  size of, 6
workbook protection, 133
workbook structure, 3
worksheet protection, 131–3