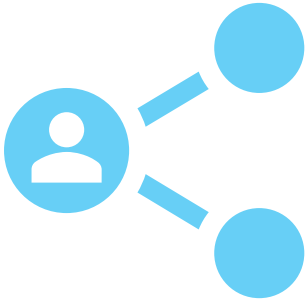
VECTOR

powering
decision confidence

# Financial Modelling

## Understand the leverage possible with VBA

# PAYING IT FORWARD



We encourage you to share with your colleagues.

Please keep the document whole to preserve the structure, intent and spirit.

While there is no single right way to approach financial modelling, we have refined our method over 30 years to deliver confidence - championing clarity, user experience, and integrity while prioritising certainty and minimising errors. VBA is an essential tool for professional financial modellers; we'd like to share some insights to give you inspiration to getting to grips with it or some pointers on best practice if you are already using it.

VBA has a higher barrier to entry than using a spreadsheet but once you know the basics you can solve 95%+ of typical transaction and reporting challenges – well they no longer become challenges!

This advice comes from both a Theoretical Astrophysicist / Project Financier and a seasoned software architect both dedicated to financial modelling for a combined tenure of 35+ years modelling transactions; we'd like to think we have something passing on!

We dedicate our time to a select number of client transactions, in-house training and our own project investments – so there is only so much we solve; however, by sharing this insight into how we work we aim to give you a boost - or a Red Bull F1 pit-stop experience if you're already racing.

Wherever you are on your journey, we're here to help you work smarter, not harder - spending less time tinkering with spreadsheets, more time confidently closing deals and powering ahead.

Enjoy our thoughts and approach, we hope it helps. If you like what we do check out [vectorHQ.co](vectorHQ.co)

*Nick + Ben*

# INTENTION

This Guide is intended to open the door to appreciating the power of well-designed and user-friendly VBA specifically for transaction financial models.

It is not a 'how-to' but a lite 'how-we' utilise VBA for transactions; we've worked to a place where there is no situation that we have not been able to solve relatively easily and would like to pass this on.

VBA is an incredibly powerful tool that is not often used properly, leading to its unfair reputation for turning models into Black-Boxes. Digesting this guide should enable you to avoid this Black-Box issue, unlock the full potential of your model and take a step closer to becoming the modelling guru in your team. In particular, the Debt-Guru!

This brief guide will cover the essentials of how we use VBA for automation and problem-solving, which are central to Debt/Equity Sizing and Scenario Analysis – our primary focus on transactions.

If this guide resonates with you ask us about training, if you are facing a transaction and need it go to smoothly ask us about how we can support.

**MODEL DEVELOPMENT**

**TRAINING**

## INTRODUCTION

What does VBA do? | I'm not a coder ?! | Easy Execution

## FOUNDATION

Pseudo Code | Anatomy of a routine | The VBA environment

Calling a subroutine | Low cognitive load | Basic logical structures

Variable types

## LOOPS

Overview | Common logic | For Next | Do Until | Do While

## DEBUGGING

Overview | Add watch | Stepping | Safety measures

## DESIGN FOR SPEED

Overview | Tracking performance

## INSPIRATION

Regular applications | Informing the user | Build a scenario table

Debt dashboard | Portfolios

# INTROD UCTION

We think of VBA as ‘half of modelling’ as it opens-up different ways of working and it automates things you would never do manually; and even if you did would be risking manual error. Whilst learning is never ending the essential tools can be learnt in a day in the right environment.

# WHAT DOES VBA DO?

In a nutshell - VBA allows you to interact with a spreadsheet in a programmatic way, enabling automation – it might sound boring but is super powerful.

For us it's an essential tool for solving capital structures, creating scenarios and advanced output as well as implementing structures which to do manually would be impossibly slow to create or manage.

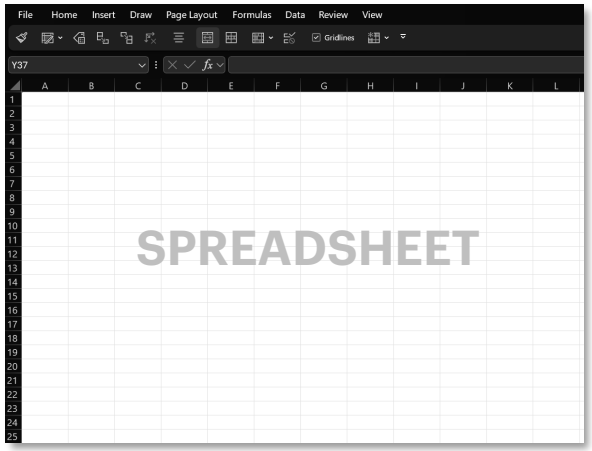
It has been embedded within Excel for 30 years – and in that time it has not really changed. As the name suggests it is a fairly, basic programming language which allows you to manipulate your spreadsheet. If you are a model developer or user in the financial sector, it makes you quicker, avoids manual errors but also provides new structural solutions to problems where the number of dimensions are an issue.

VBA is essential for:

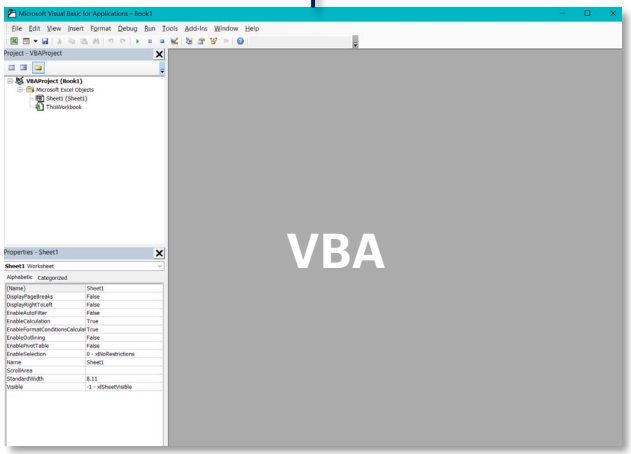
- Debt and equity sizing and principal repayment management
- Advanced scenarios and sensitivities – inc. Mine Plan management
- Portfolio and cohort modelling
- Report generation
- Automating anything you need to do more than once!

**OPEN VBA**

From anywhere in your usual Excel workspace press ALT + F11 to open the VBA interface.




ALT+F11



# I'M NOT A CODER ?!

So what?! If you can think logically and know what you want to achieve the "code" is the easy bit – you just need to know where to start. This guide will show you the core parts of what you need to know and if you want to really cement your knowledge, then let's do some Training. If you can describe what you want to achieve to somebody who knows VBA – you are 75% of the way-there. VBA will do exactly what you tell it to so if you can do things in the right order with the correct syntax – then you have an awesome tool at your disposal.

We can teach you the approach and syntax needed to harness VBA, for even quite advanced financial modelling, in a single day – the secret sauce is in learning what to make the Spreadsheet do vs the Code; and when do it. Some of our most sophisticated capital optimisation routines use nothing more than the most basic coding structure. Once you can 'tell the spreadsheet part of the model' what to do you then you unlock analysis you would never dream of doing manually and implement structural solutions that you would have otherwise avoided.

Great examples include:

- Debt sizing – this is without exception only an iterative exercise in changing variables to satisfy constraints.
- You have 10+ scenarios to run and on some of then you need to re-size debt – you never want to do this manually!
- You have a portfolio of say 120 identical, or could be identical, assets / projects / business units but there is no way you want to manage 120 worksheets when you want to make a change – but make a change to the first one and have it flow through, priceless.
- 2-dimensional table but each time the value is generated you need to re-size debt.
- You want to generate the full financial statements and Exec Summary of each scenario (not just a small handful of KPI) – you wouldn't do this manually as the next time something changed you would have to do it all again.
- You want to find the breakeven prices in each period over a project to achieve a target LLCR – where a change affects other changes you've made!


AI is very good at helping you with Syntax. Currently it will provide an OK routine given good instruction, but this is only a time saver when you already know what you are doing! Use it for debugging syntax not creating it.

# EASY EXECUTION

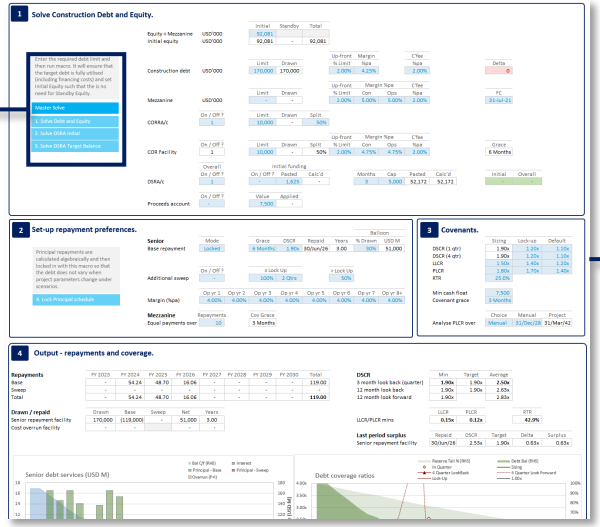
You can execute (run) macros either using ALT + F8 or more usefully by attaching them to an icon or button within the spreadsheet. When the user clicks a button, the macro runs – and we make sure they know what is happening – see further in this document. It is helpful to locate all relevant macros 'buttons' in one area or in the most relevant area – and you can be quite creative to lower the cognitive load of using the model.

Laying out macro buttons logically and clearly takes some consideration.

Enter the required debt limit and then run macro. It will ensure that the target debt is fully utilised (including financing costs) and set Initial Equity such that the is no need for Standby Equity.

Master Solve

1. Solve Debt and Equity
2. Solve DSRA Initial
3. Solve DSRA Target Balance




**LEAN + SEQUENTIAL**

In this debt dashboard we've made the macros clear but enabled the user to solve everything with a 'Master' routine or stage by stage with the button's numbered 1.2.3. This approach is helpful during development but quickly trends to just running the Master "solve everything all at once' option.

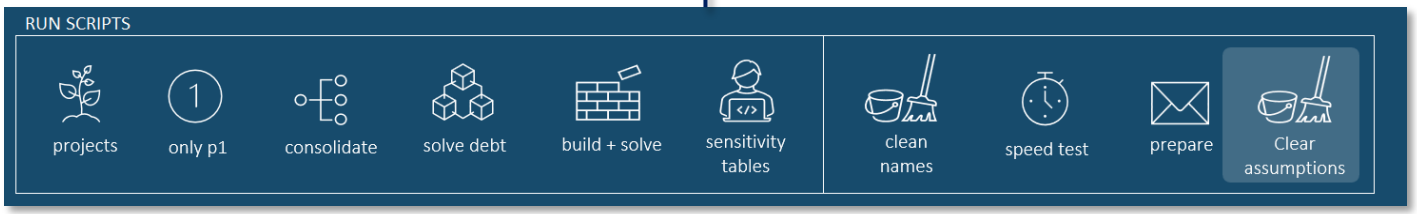
**MENU**

These are simply Excel Icons designed to communicate what macro to run. This is from an advanced portfolio model where it needed to be very clear what to do. The time to setup is say 10 minutes means the next weeks and months are easy.

**ADVANCED NOTE**

Very occasionally it is useful for a macro to run without explicit user execution; such as when an option is selected from a drop-down list.

**RUN SCRIPTS**



# FOUND ATION

In this section you will see our go-to approach for laying out any routine - especially for solving debt, running scenarios and interacting with the spreadsheet for normal situations.

Outside of transaction modelling we tend to use C# (C-Sharp)- when things get funky.

# PSEUDO CODE

In Financial Modelling – The Guide for Essential Professional Skills we advised to not start a Financial Model in Excel but by thinking about the design and documenting, this can be a sketch or whatever. An efficient professional outcome is very difficult to achieve if you don't follow this; the same is true for VBA code.

Solving a problem using VBA is broken into two parts:

1. Expressing the problem using business terms being sure to capture all of the facets of the problem. We call this Pseudo Code – for which you do not need to know the VBA language.
2. Conversion of the expressed problem into code

We always start with Pseudo Code – it sounds fancy but it's just the outline of what you want to achieve, what steps will be required - without worrying about the code itself. Think of this exercise as you are explaining your desired outcome to somebody else, who knows syntax, who will do it for you.

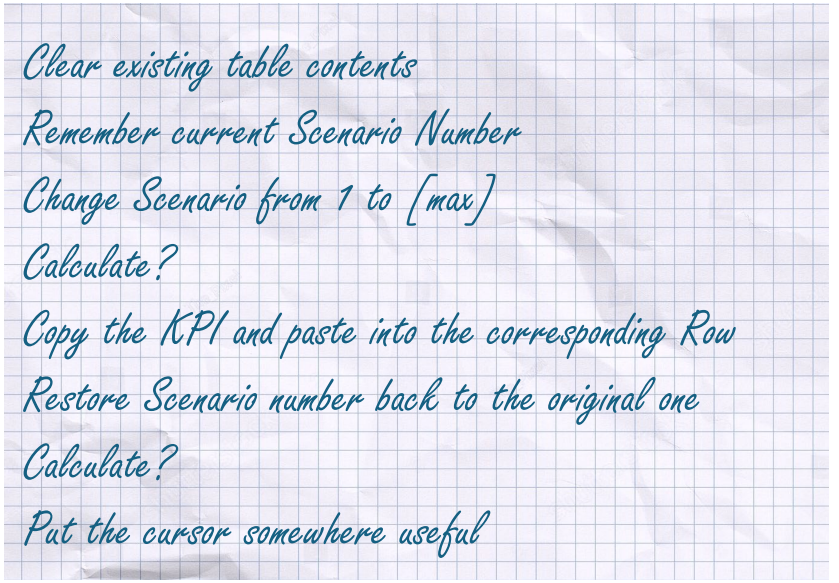
In 1994 Nick used to have send code overnight to NASA Jet Propulsion Labs to be run – when you have a 24 hour wait for the results it forces you to think ahead!

Practically, just like in the spreadsheet, requirements change so your mission is to think ahead and capture 90% of this before you start coding. It sounds like it takes longer than 'just getting started' but our clock is always ticking so we wouldn't do it if wasn't faster!

Key considerations are:

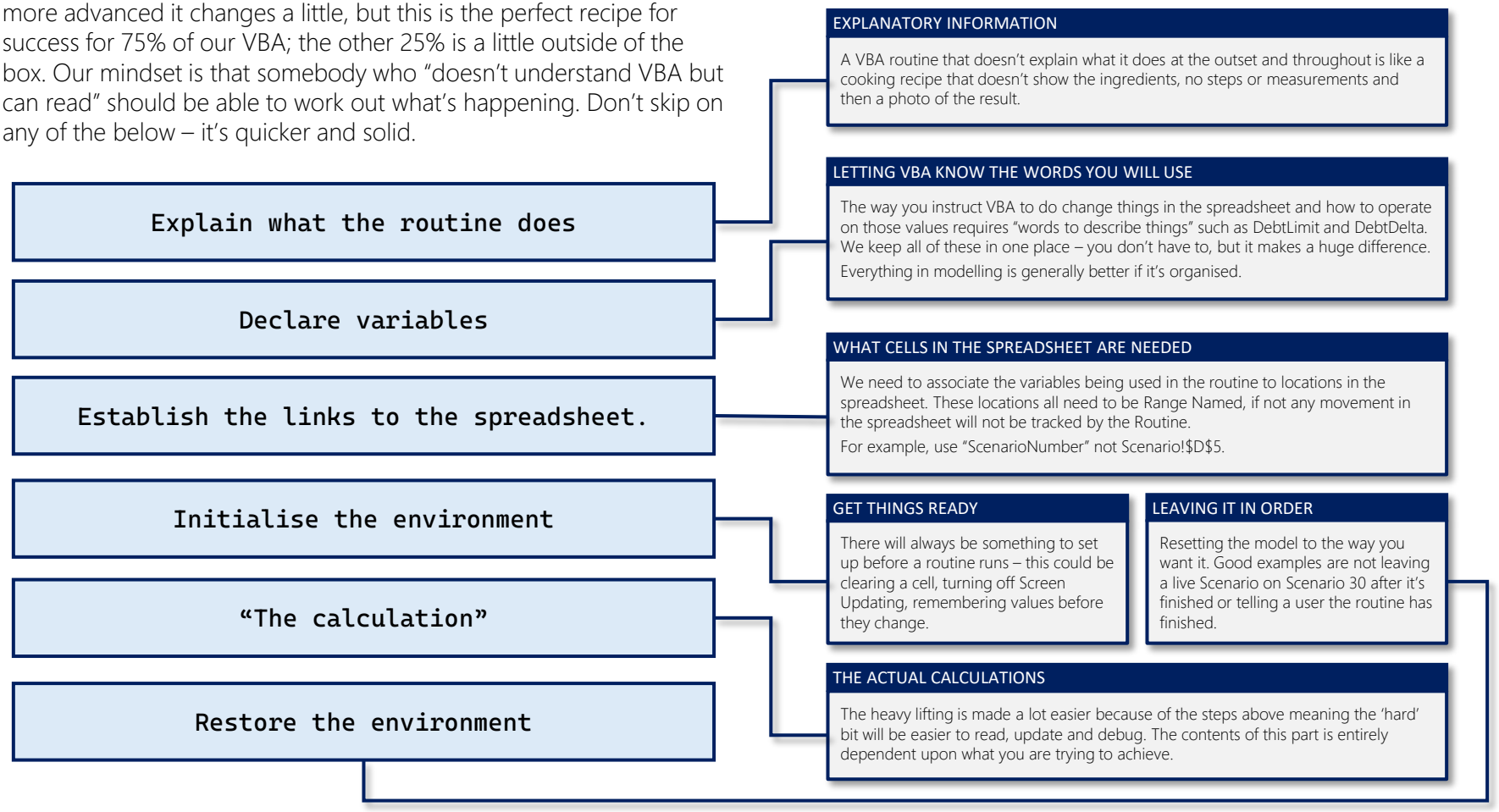
- What parts of the spreadsheet will the code interact with – do they exist yet or do you need to build them?
- What needs to change or be set up before performing any calculations?
- How do you want the spreadsheet 'left' when the routine is finished? You appreciate this part more with experience..

Here is some pseudo code to construct a simple scenario table – as you become more experienced you find yourself writing this in more detail.



# ANATOMY OF A ROUTINE

Ok – so as a starting point do this every single time. As you become more advanced it changes a little, but this is the perfect recipe for success for 75% of our VBA; the other 25% is a little outside of the box. Our mindset is that somebody who "doesn't understand VBA but can read" should be able to work out what's happening. Don't skip on any of the below – it's quicker and solid.



# ANATOMY OF A ROUTINE

Let's see what this looks like put into practice. Don't worry about what this one does or the colours – just the structure.

```
Option Explicit

Sub FeedSolve()
' Developed by Vector Financial Modelling"
' Learn more at www.VectorHQ.com
' This routine solves the circularity resulting from oversized output from pass 1 and pass2
' being feed back into the MMD sizer within the same month.

    Dim FeedMode As Range
    Dim FeedCalculated As Range
    Dim FeedPasted As Range
    Dim FeedDelta As Range
    Dim FeedLoopMax As Integer
    Dim FeedLoop As Integer

    Set FeedMode = Range("Physicals.Feed.Mode")
    Set FeedCalculated = Range("Physicals.Feed.Calculated")
    Set FeedPasted = Range("Physicals.Feed.Pasted")
    Set FeedDelta = Range("Physicals.Feed.Delta")

    'Initialise environment
    FeedMode.Value = "Dynamic"

    FeedLoopMax = 30
    FeedLoop = 0
    FeedPasted.ClearContents
    Application.ScreenUpdating = False

    'Solve feed physicals
    Do Until FeedDelta = 0 Or FeedLoop = FeedLoopMax

        FeedPasted.Value = FeedCalculated.Value

        FeedLoop = FeedLoop + 1

        Application.StatusBar = "Solving feed: " & FeedLoop

    Loop

    FeedMode.Value = "Locked"

    'Restore Environment
    Application.ScreenUpdating = True
    Application.StatusBar = False

End Sub
```

**OPTION EXPLICIT**

This is an often, over-looked aspect but should be at the top of each Module – it forces your macro to only accept variables which have been explicitly declared. Useful to avoid at you in the code - because the consequences are a headache!

**EXPLAIN WHAT IT DOES**

It's important and useful to explain what a macro does – not an essay just a few lines. It's one of those small things that increases User Experience (UX) and is also useful when you open it up months or years later!

**DECLARE (DIMENSION) THE VARIABLES**

Letting VBA know the Names of the Variables you want it to use and their type. There are only a few Types you need to consider, 90% of the time they will be Range (somewhere in the spreadsheet) or Integer.

**ESTABLISH THE LINKS TO THE SPREADSHEET**

Set allows VBA to know what locations in the spreadsheet are tied to the Variables that are Ranges. A helpful tip is to keep them in the same order and grouped by their relevance.

**INITIALISE THE ENVIRONMENT**

This is where you make sure the Spreadsheet is prepared for what you are about to do and anything that needs a starting value, for example how many loops do you want to run before it stops – a good failsafe.

**CALCULATION**

This where the core instructions live. You will note we've used line spaces to make it easier to read which becomes more important as routines evolve.

**RESTORE ENVIRONMENT**

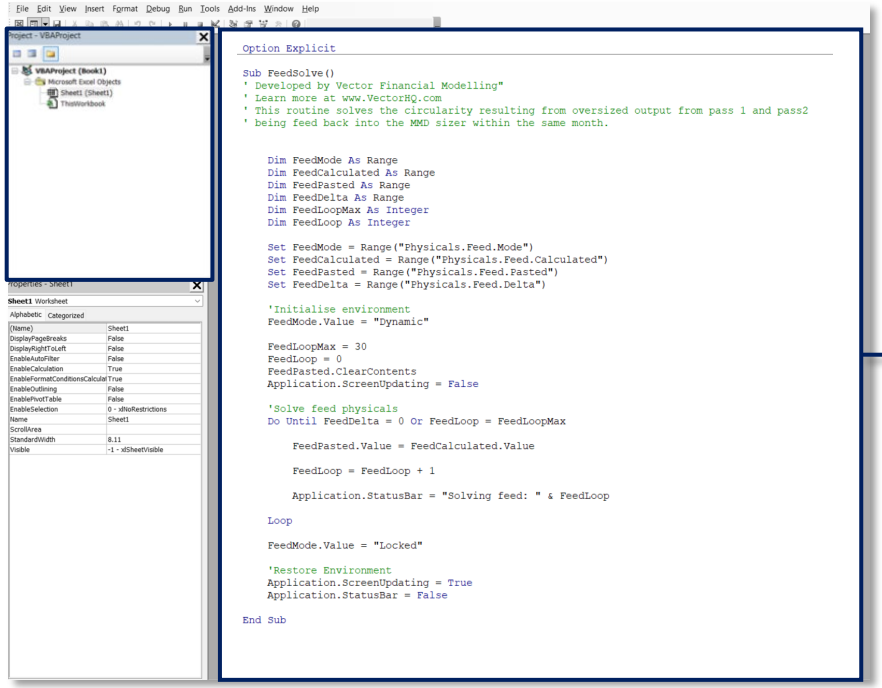
In this example we turn Screen Updating back on (switched off in Initialise for a faster compute time). It also restores the Status Bar to say Ready rather than stay on the last Solving Loop info.

# THE VBA ENVIRONMENT

## PROJECT PANEL

This is where you will see all the worksheets and modules available in the Workbook. As you create modules they will appear here – we group the contents of Modules to be relevant. For example: Debt, Scenarios, Reporting

The lower panel you don't really need to use much other than to change the name of a Module to be more relevant. There are some sneaky options here for hiding a worksheet, but we don't advise it.




## CODING PANEL

Once a Module has been inserted and ideally named clearly this is where you write your code. You spend 99.99% of your time in this area.

To start creating a macro type
Sub and the name and VBA auto completes the line adding brackets and the End Sub routine. Everything between these two lines will be considered when you run the macro.

Because you cannot use spaces or special characters in the Routine Name we recommend _ or CamelCase to make it easier to name – this is important because if you want to run it from a button / icon within the spreadsheet you need to identify it – also if you call it from another Routine then it is clear what is being called.

Keep the names clear – no prizes for abbreviation.

**OPEN THIS FROM EXCEL USING ALT+F11**

## INITIALLY

When you create a Module, you will be faced with a blank window…it's at this point you need to know where to start! Don't work in a tiny window – maximise it. Because you only need one Module window open at the same time.

# CALLING A SUBROUTINE

```
    For ScenarioLoop = 1 To ScenarioMax

        UpdateMacroConsole "Building scenario " & ScenarioLoop & " of " & ScenarioMax

        'Change the active scenario number (in the spreadsheet) to the value of this loop
        ScenarioNumberActive.Value = ScenarioLoop
        PerformCalculate

        If ScenarioNameActive = "Spare" Then GoTo Continueloop

            'Check if Debt macro needs to be solved (always solve for Base Case)
            If DebtMacroRun = "Yes" Or ScenarioLoop = 1 Then
                SolveFacilityLimit

                'Select the live line of the table
                ScenarioTableCopy.Copy

                'Paste it into the corresponding row
                ScenarioTableAnchor.Offset(ScenarioLoop).PasteSpecial (xlPasteValues)

                'If debt has just run then resolve to Base Case before proceeding with next scenario
                ScenarioNumberActive = 1
                PerformCalculate
                SolveFacilityLimit

             End If

        ScenarioNumberActive.Value = ScenarioLoop
        PerformCalculate

        'Select the live line of the table
        ScenarioTableCopy.Copy

        'Paste it into the corresponding row
        ScenarioTableAnchor.Offset(ScenarioLoop).PasteSpecial (xlPasteValues)
        PerformCalculate

Continueloop:

    Next ScenarioLoop
```

**CALLING ANOTHER ROUTINE**

If the Debt macro needs to run, then rather than repeat the code here we simply call the SolveFacilityLimit Subroutine. Any changes made to that would automatically flow through too other routines which reference it.

In Financial Modelling – The Guide for Essential Professional Skills we introduced the concept of Don't Repeat Yourself (DRY). The principle is from code development. In essence only calculate something once and then refer to it – in coding this means breaking instructions down into their own "homes" and then call them as needed rather than create one long script which may have the same instructions as in another one – they should instead all refer to the same Subroutine. In this example if a change was made to SolveFacilityLimit it will flow through to all routines that call it.

The more sophisticated a routine becomes the more subroutines we create. A Master Solve routine may literally just be two lines within a Loop structure:

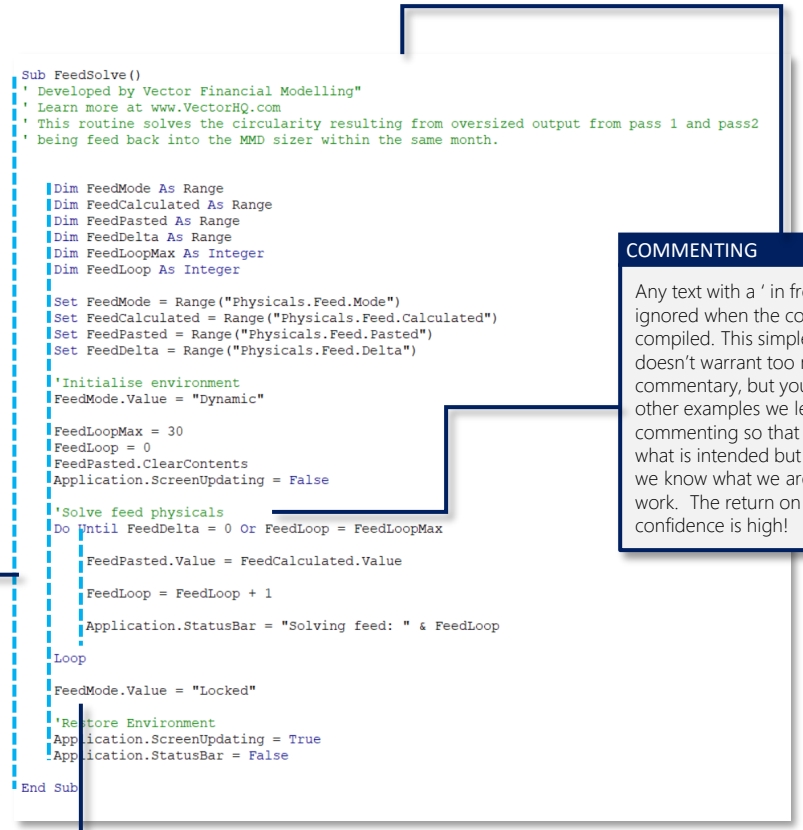
- Call DebtSolve
- Call EquitySolve

# LOW COGNITIVE LOAD

One big slab of code is the equivalent poor practice to one long formula in the spreadsheet.

Breaking blocks up, indenting the logical hierarchy, and adding explanations is essential to making the code easier to read – spending less time figuring out what is happening, where and when. This approach is essential for all users, especially you as a developer to aid debugging – which is a normal occurrence - and adapting as the requirements change.

If you are working under transaction pressure these are the parts that make it more reliable so you can be confident in the results. Think of a routine like a formula in Excel, it's better to break it up into bite-size pieces than one monster!



```
Sub FeedSolve()
' Developed by Vector Financial Modelling"
' Learn more at www.VectorHQ.com
' This routine solves the circularity resulting from oversized output from pass 1 and pass2
' being feed back into the MMD sizer within the same month.


    Dim FeedMode As Range
    Dim FeedCalculated As Range
    Dim FeedPasted As Range
    Dim FeedDelta As Range
    Dim FeedLoopMax As Integer
    Dim FeedLoop As Integer

    Set FeedMode = Range("Physicals.Feed.Mode")
    Set FeedCalculated = Range("Physicals.Feed.Calculated")
    Set FeedPasted = Range("Physicals.Feed.Pasted")
    Set FeedDelta = Range("Physicals.Feed.Delta")

    'Initialise environment
    FeedMode.Value = "Dynamic"

    FeedLoopMax = 30
    FeedLoop = 0
    FeedPasted.ClearContents
    Application.ScreenUpdating = False

    'Solve feed physicals
    Do Until FeedDelta = 0 Or FeedLoop = FeedLoopMax

        FeedPasted.Value = FeedCalculated.Value

        FeedLoop = FeedLoop + 1

        Application.StatusBar = "Solving feed: " & FeedLoop

    Loop

    FeedMode.Value = "Locked"

    'Restore Environment
    Application.ScreenUpdating = True
    Application.StatusBar = False

End Sub
```

**INTENDING**

"Tab" moves selected text 'in' by a set amount – you can do on groups of lines and if you get yourself in a pickle then paste the code into ChatGPT and ask it to re-format, then paste it back in.

**LINE SPACING**

The more complex your routine becomes the more useful it is to separate code with an empty line.

VBA ignores them and it pays huge dividends via code Clarity and lowering the users Cognitive Load.

There is nothing worse than a 'slab' of code when you are under time pressure.

**COMMENTING**

Any text with a ' in front of it will be ignored when the code is compiled. This simple example doesn't warrant too much commentary, but you will see in other examples we lean on commenting so that others know what is intended but as importantly we know what we are doing as we work. The return on efficiency and confidence is high!

# BASIC LOGICAL STRUCTURE

Before considering Loops in the next section, it is important to get to grips with the logical structures that will be needed in the tests to determine if a Loop needs to run or stop. Consider these like use IF(), OR() and AND() functions in the spreadsheet – they are your pantry staples. Rather than describe them, as they are self-evident, here are some examples.

**EXAMPLE | IF**

Using IF to determine whether to do something or not.

```
IF ABS(DebtDelta) > DebtTolerance THEN
      {Keep solving}
ELSE
      {Do something else or Stop}
END IF
```

**IF**

If the result of a debt calculation is above the nominated tolerance level, then keep going. Risky as this could just keep looping and you would have to force a stoppage which is not desirable.

**EXAMPLE | AND**

Using AND to check whether to do something or not.

```
IF AND ( ABS(DebtDelta) > DebtTolerance, DebtLoop < = DebtLoopMax) THEN
      {Keep solving}
ELSE
      {Do something else or Stop}
END IF
```

**AND**

By introducing an AND into the IF decision we can control the solving loop not going beyond a fixed number of loops. The value of DebtLoopMax could be defined within the debt dashboard or within the VBA code.

This means if the debt doesn't solve within say 30 loops, the routine will stop – keeping you in control and not sitting wondering what is happening.

# VARIABLE TYPES

Variables are the 'words' that describe concepts that VBA needs from you to 'do-things'. Variable names are by default declared within each routine. There is a concept called Global but let's not worry about that for now.

For example: You have a Scenario Number in a Scenario Manager, within the spreadsheet and you want it to cycle through 10 scenarios then you need to tell it.

- ScenarioNumber (and where that lives in the spreadsheet) – this would be called a Range as it represents something in the spreadsheet.
- ScenarioLoop which can go from 1 to 10 in increments of 1 – this would be called an Integer because it doesn't exist in the spreadsheet, so it's not a Range, it's only a construct within VBA and it only has whole number values.

There are only a handful of variable types you need to lock down for most tasks:

| VARIABLE TYPE | USE CASE |
|---|---|
| Range | Relates to a cell or array of cells within the spreadsheet. |
| Integer | Usually used for loops and counters within VBA not the spreadsheet |
| String | Mostly used for words / names. Lookup codes to find something within a string of values, FX codes, FY years etc. The least commonly needed for normal transaction situations so don't stress over this one. |
| Single | For numerical values defined within VBA rather than the spreadsheet where precision is not critical |
| Double | For numerical values defined within VBA rather than the spreadsheet where precision is critical |

# LOOPS

An essential tool in "doing something until something is satisfied" – usually the heart of all debt and scenario routines.

Loops come in different forms which we explore in this section.

# LOOPS

As the name suggests, Loops will carry out a series of actions until a test is met. There are different types of Loops, your choice of which, and the way it knows when to stop are critical to fast solving.

| TYPE | BEHAVIOUR |
|---|---|
| For.. Next | Predefined number of iterations (loops) |
| Do Until | Checks at the end of each iteration (loop) |
| Do While | Checks at the beginning of each iteration (loop) |

**DO UNTIL / WHILE**

Which one of these to use is not always obvious. It depends on the situation and how you define the test.

**NOTE**

The more experienced you become the less you find yourself using a For Next Loop because you become increasingly more focused on defining conditions that check if a Loop needs to run or not. If you are just starting out and creating a table that always has say 20 rows of scenarios then it does the job.

```
'Solve feed physicals
Do Until FeedDelta = 0 Or FeedLoop = FeedLoopMax

    FeedPasted.Value = FeedCalculated.Value

    FeedLoop = FeedLoop + 1

    Application.StatusBar = "Solving feed: " & FeedLoop

Loop
```

**EXAMPLE**

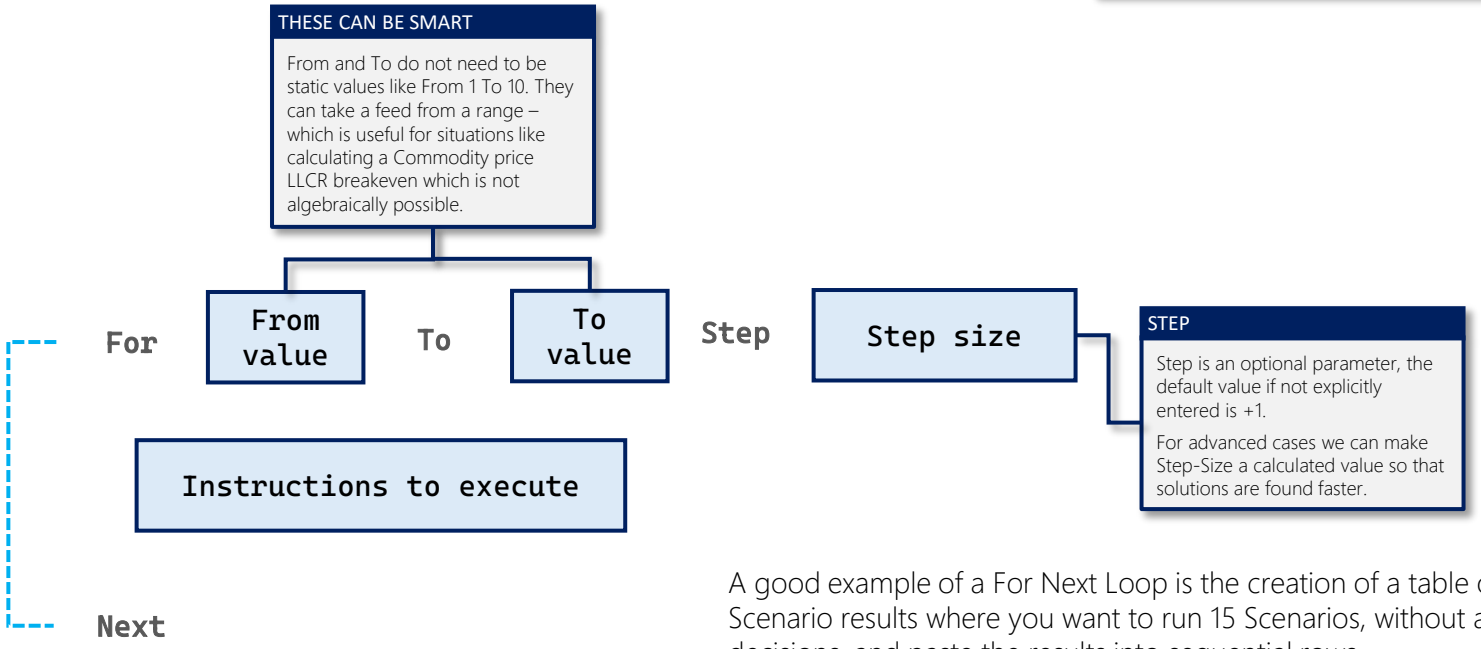
This is a Copy and Paste routine to solve a physical feedback process, in a mine processing operation. The calculated value is copied over the pasted value until the difference between the two lines (FeedDelta) = 0 or the number of Loops executed hits a predetermined maximum. It also sends a message to the Status Bar of where it is up to ,so the user isn't left wondering.

# LOOPS | FOR NEXT

The For…Next loop in Visual Basic is one of the most commonly, used Loops for simple applications – especially tables. It allows you to execute a block of code a specific number of times, based on a counter variable that is incremented or decremented in each iteration. Perfect for simple 1D and 2D tables where no 'decision' is needed for it to stop – easier to read and understand when you are first starting out.

**LOOP DIRECTION**

Loops do not need to always go 'forwards' – in some situations you may want to work from the end and work backwards. LLCR break-evens are good examples of this.



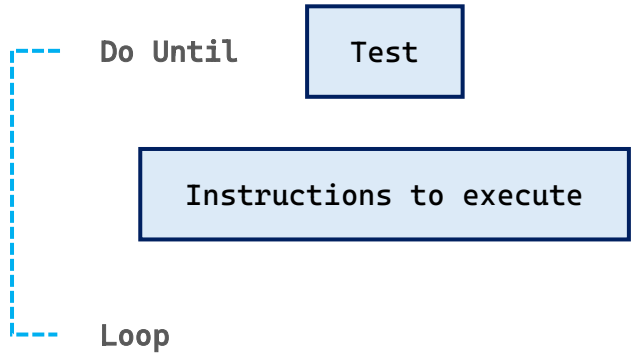

A good example of a For Next Loop is the creation of a table of Scenario results where you want to run 15 Scenarios, without any decisions, and paste the results into sequential rows.

# LOOPS | DO UNTIL

Do Loops are the most common control structure to perform an action according to a test. There are two types:

1. Do While
2. Do Until

Your choice of Do Loop is based on the logic you are wish to execute.




## When to use Do Until

A Do Until loop will run the contained logic even if the test is passed but then stop. It checks at the end of each iteration (loop)

**WARNING**

The most common issue with Loops not working correctly, or not running at all are:

- Incorrect choice of While/Until
- The way the Test is constructed.

## Application of Do Until:

Copy and Paste a calculated value until the two lines give the same result, the primary way to break and control circular references. The test in this case would be that the absolute value of the difference between the sum of each line is <>0 or within a specified tolerance.

# LOOPS | DO WHILE

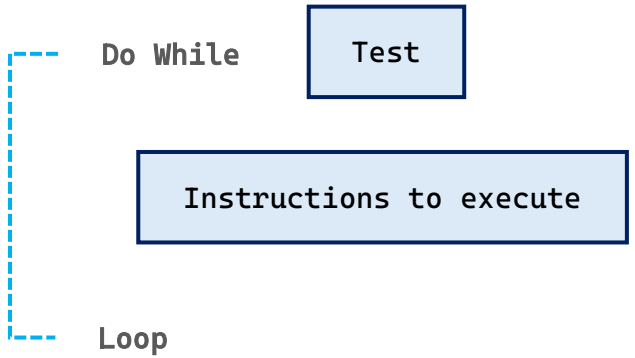
When to use Do While

A Do While loop will not run the contained logic if the test is passed. It checks at the start of each iteration (loop). This is a subtle but important difference especially when working with debt solving routines.

**WARNING**

The most common issue with Loops not working correctly, or not running at all are:

- Incorrect choice of While/Do
- The way the Test is constructed.
- Initial conditions meaning the Loop doesn't start.




Application of Do While:

In solving a debt repayment profile that maybe because of an activated sweep mechanism must not ever solve the next repayment if the debt is repaid in the preceding period. Running it would cause issues with payments beyond the debt being repaid.

# DE BUGGING

Excel spreadsheets and VBA do exactly what you ask them to - so when they're not working - let us show you the core techniques to find out why your routine isn't doing what you want it to.

# **DEBUGGING** | Add Breakpoint

There are three essential tools to know and more often that not we write code that doesn't need hardcore debugging – but when you start out these are the tools you need to know.

- Add a Breakpoint – allows you to systematically pause and inspect.
- Add Watch – track the values of results as the routine runs.
- Step-Into / Over / Out – run 'bit-by-bit' in different ways so you can assess what is happening.

```
Sub LockPrincipal()
'Once the debt is sized in the base case this macro locks the principal schedule in
'Means that scenarios won't push the loan life out

    'Dimension variables
    Dim PrincipalMode As Range
    Dim PrincipalDynamic As Range
    Dim PrincipalLocked As Range
    Dim PrincipalDelta As Range

    'Set links to workbook
    Set PrincipalMode = Range("PrincipalMode")
    Set PrincipalDynamic = Range("Principal.Dynamic")
    Set PrincipalLocked = Range("Principal.Locked")
    Set PrincipalDelta = Range("Principal.Delta")

    'Iterate
    Do Until Round(Abs(PrincipalDelta), 2) = 0

        PrincipalLocked.Value = PrincipalDynamic.Value

        PerformCalculate

    Loop

    'Lock in principal schedule
    PrincipalMode = "Dynamic"

    PerformCalculate ' to make sure values are updated

    PrincipalLocked.Value = PrincipalDynamic.Value
    PrincipalMode = "Locked"

    PerformCalculate ' to make sure values are updated

End Sub
```

**ADD A BREAKPOINT**

Select the line you want the macro to stop at and press F9. Remove it by pressing F9 again.

This allows you to run a macro only up to a select position which is very helpful when you want to understand what is happening rather than running the whole thing and trying to work out the problem. A good example is solving debt but skip solving the DSRA for now.

# DEBUGGING | Add Watch

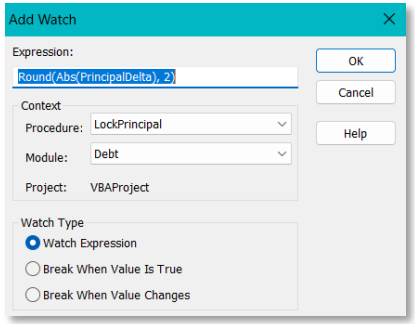
Adding a Watch allows you to keep an eye on the value of a variable throughout the solve routine. There is a lesser used equivalent in the spreadsheet too which is useful but buried. We often use it to track critical KPI all the time without needing to import the value into the worksheet.

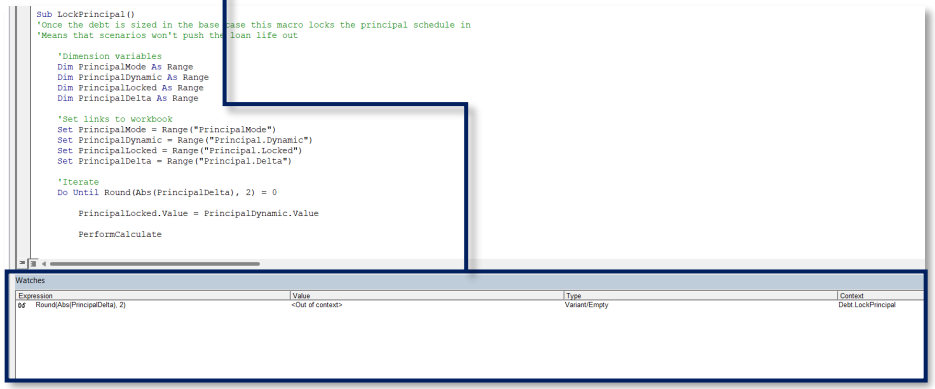
```
Sub LockPrincipal()
'Once the debt is sized in the base case this macro locks the principal schedule in
'Means that scenarios won't push the loan life out

    'Dimension variables
    Dim PrincipalMode As Range
    Dim PrincipalDynamic As Range
    Dim PrincipalLocked As Range
    Dim PrincipalDelta As Range

    'Set links to workbook
    Set PrincipalMode = Range("PrincipalMode")
    Set PrincipalDynamic = Range("Principal.Dynamic")
    Set PrincipalLocked = Range("Principal.Locked")
    Set PrincipalDelta = Range("Principal.Delta")

    'Iterate
    Do Until Round(Abs(PrincipalDelta), 2) = 0

        PrincipalLocked.Value = PrincipalDynamic.Value

        PerformCalculate

    Loop

    'Lock in principal schedule
    PrincipalMode = "Dynamic"

    PerformCalculate ' to make sure values are updated

    PrincipalLocked.Value = PrincipalDynamic.Value
    PrincipalMode = "Locked"

    PerformCalculate ' to make sure values are updated

End Sub
```

**ADD A WATCH**

Highlight the component you want to 'watch', right click and select "Add Watch". The Add Watch box will pop-up where you can choose how you want to watch it.

When you press OK the Add Watch Window will pop up showing all Watches you have created – these are 'live' so as the routine runs you can track the results.



# DEBUGGING | Stepping

Stepping allows you to incrementally work through a routine without just letting it run all the way through. There are three primary Step methods.

- Step-Into – Press F8 to run the macro line by line
- Step-Over – Press SHIFT + F8 to skip the next logical structure – useful for when you do not want a Subroutine to be executed.
- Step-Out – a little more advanced, when you are within a logical structure, say a loop, you can use this to exit that specific logic and go back to where you were. Kind of.

```
Sub LockPrincipal()
'Once the debt is sized in the base case this macro locks the principal schedule in
'Means that scenarios won't push the loan life out

    'Dimension variables
    Dim PrincipalMode As Range
    Dim PrincipalDynamic As Range
    Dim PrincipalLocked As Range
    Dim PrincipalDelta As Range

    'Set links to workbook
    Set PrincipalMode = Range("PrincipalMode")
    Set PrincipalDynamic = Range("Principal.Dynamic")
    Set PrincipalLocked = Range("Principal.Locked")
    Set PrincipalDelta = Range("Principal.Delta")

    'Iterate
    Do Until Round(Abs(PrincipalDelta), 2) = 0

        PrincipalLocked.Value = PrincipalDynamic.Value

        PerformCalculate
```

**STEPPING**

When you are stepping the active line of code is highlighted in yellow, so you know where you are up to. Combine this with an Add-Watch to fully appreciate the variables values.


**TOOLBARS**

There are a several useful commands accessible from the Debug and Edit toolbars.

It is beyond the scope of this document to go through each one but in training we introduce these as needed. We find they are not heavily used if you use the shortcuts already prescribed but good to know they are there.

# SAFETY MEASURES

When working in VBA there are a few things to be aware of which will affect the speed of your work and manage user expectations. Keep in mind these Safety Measures protect you from losing your work – don't be the instigator of a Denial-Of-Service attack on your own work!

- Top tip! You can't Undo a Macro once it's run – Excel's Undo-Stack is cleared. Therefore, save the file before running. See our Modelling Guide on file saving to avoid any issues.
- Build in Fail Safes into all Loops – you do not want to be in the situation where you are pressing Escape hoping the macro will stop when it's clearly in a never-ending death spiral!
- If you are about to run a macro that has consequences, like clearing information and rebuilding, or will take a long time to run – like building 20+ assets as individual sheets or a long debt solve then it's helpful to pop a message box up that explains what is about to happen with an Ok or Cancel button – this is not a fancy thing that takes time – its's a really helpful step to avoid inadvertent execution.
- If you have turned Screen Updating off, you may need to force Excel to update messages in the Status Bar / plot's etc., this is achieved using the Do Events command. Keep this in mind if you can't see anything happening. Do Events is like a wake-up call for the spreadsheet part of the file.

# DESIGN FOR SPEED

A macro can only be as good as the underlying model it operates on - conversely, a good model with a poorly written macro yields the same inefficiency.

In this section, we introduce some key concepts we always consider to ensure both parts of the system operate as efficiently as possible.

# DESIGN FOR SPEED

For many routines speed will not be an issue – however in larger models and especially debt solving and portfolio models it becomes a 'whole situation' – especially under the time expectations of a transaction.

We are always comfortable that our routines are running at an optimum speed by following the guidelines below.

- Turn Screen Updating Off (once you know it's working…)
- Establish the link to the spreadsheet once only using Variable Names and the Set instruction. If you don't do this and reference the range each time it is used VBA is going back and forth more times than it needs to.
- Do not execute more Loops than you need to – this can be controlled with Tolerances. For example; no need take 30 seconds to solve a $100m loan facility to $0.0001 when it solves to $1 in 2 seconds.
- Don't Repeat Yourself (DRY) – only calculate something once and then call it as needed. To be fair - this doesn't change the speed of a routine, but it has a huge impact on your efficiency and effectiveness.
- Turn Calculation Mode to Manual and then only Calculate when needed. It is good practice to then return calculation model to automatic or even better inspect and remember the calculation mode as part of initialising the environment.
- Open your Excel file as "One instance of Excel" – see our "Financial Modelling – The Guide to Essential Professional Skills" to learn more.

# TRACKING PEFORMANCE

For Debt Solving routines it is very helpful to know how long it is taking so that if something changes and it takes longer you don't slowly get used to it taking longer and can find a way to make it work faster.

This approach worked well for many years and is worth the time to implement. It has been superseded now with a dynamic macro consol (see Inspiration section) but if you want to keep an eye on speed try this.

- Within your debt dashboard, or somewhere else easy to find, create two, time formatted cells and name them appropriately something like Time_Solve_PreviousRun and Time_Solve_ThisRun.
- Within your macro create a time tracker that starts upon execution and finishes when it is complete.
- Before running paste, the ThisRun into PreviousRun, this means when it is finished you will have a comparison to this time vs last time.
- Alternatively, as illustrated here you could show the information in a pop-up box upon completion.
- Keep in mind that whilst a single calculation may appear instant – when there is a Loop that needs to run 10x within a loop that needs to run 10x within a loop that needs to run 10x you are now dealing with a scalar factor of 1,000.

A time tracker looks like this

```
    'Initialise values
    EquityInitial.Value = 0
    StartTime = Timer
    MasterLoop = 0
    MasterLoopMax = 10

        Do Until DebtMasterTest = 0 Or MasterLoop = MasterLoopMax

            Call SolveDebtandEquity

            If Abs(DSRAInitialDelta) < 0.01 Then GoTo 10
               Call DSRAInitial
10
             Call LockPrincipal

             If Abs(DSRATargetBalanceDelta) < 0.01 Then GoTo 20
               Call DSRATargetBalance
20
             Application.StatusBar = "Running | Loop = " & MasterLoop
             DoEvents

        MasterLoop = MasterLoop + 1

        Loop

    'Timer information
    MinutesElapsed = Format((Timer - StartTime) / 86400, "hh:mm:ss")

    'Restore environment
    Application.ScreenUpdating = True
    Application.StatusBar = False

    MsgBox "Debt, equity and DSRA are solved after " & MasterLoop & " iterations of each routine." & vbNewLine _
        & " " & vbNewLine _
        & "Ran in " & MinutesElapsed & " minute(s)."
```

# VECTOR FOCUS

We develop transaction ready models for a wide range of situations, with deep experience in

- Renewables – especially multi-technology, multi-regional, portfolios.
- Mining – all Metals, Minerals and Processing.
- Manufacturing – Chemicals and Green Fuels.
- Infrastructure and Regulated Assets.
- Corporate Business Modelling.

We deliver in-house, in-person training for your teams, we specialise in working with small groups and building capability over their careers. Our approach has been developed over decades, is 100% hands-on and we are told our passion shows. Our current courses are:

- Financial Modelling Fundamentals
- Project Finance Modelling
- Modelling Renewables Projects
- Modelling Three Way Financial Statements
- Advanced debt modelling using Visual Basic

If you'd like us on your side in a transaction or to build your teams capabilities contact us at hello@vectorHQ.co

**MODEL DEVELOPMENT**

**TRAINING**

# INSPIR ATION

Take a look at common applications where we lean on VBA and know that even the most complex debt-solving scenarios can be achieved with a remarkably small toolkit. We can teach your team this.

# REGULAR APPLICATIONS

We use Macros for

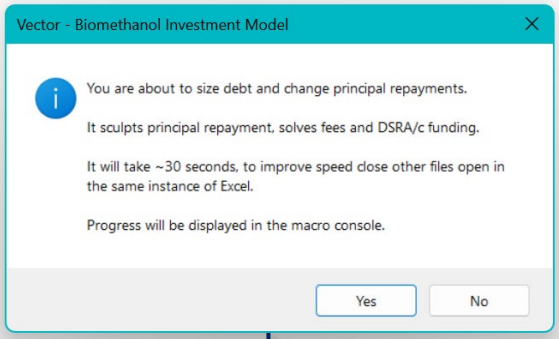
- Solving the full range of Debt and Equity Structures
- Creating Scenarios, especially when Debt or a Physical process needs to be solved for each run.
- Creating 1D and 2D Sensitivity tables – noting that if the output is Equity returns debt may need to be solved for each element.
- Creating Reports (Exec Summary and 3-Way Financial Statements for all Scenarios) and exporting to another Workbook.
- Importing and exporting data
- Spawning portfolio assets from a Master asset
- Consolidating Portfolio information into one worksheet, dynamically – avoiding the nasty INDIRECT() function.
- Tracking changes in the worksheet
- Anything that needs to be automated, such as Data Grouping and resetting the Freeze Pane position on each worksheet upon a Save and Close event
- Calculating break even price profiles when an algebraic solution is not viable.
- Creating and Clearing Range Names

Outside of these applications we use C-sharp for:

- Model auditing
- Formula manipulation and efficiency

# INFORMING THE USER

There is nothing worse than not knowing if a macro should be run or not but even more, when one is running what is it doing? Has it crashed? Do I need to do anything?




```
Sub SolveFacilityLimit_Button()
    Dim Answer As Integer

    Answer = MsgBox("You are about to size debt and change principal repayments." _
                    & vbNewLine _
                    & vbNewLine _
                    & "It sculpts principal repayment, solves fees and DSRA/c funding." _
                    & vbNewLine _
                    & vbNewLine _
                    & "It will take ~30 seconds, to improve speed close other files open in the same instance of Excel." _
                    & vbNewLine _
                    & vbNewLine _
                    & "Progress will be displayed in the macro console.", _
                      vbInformation + vbYesNo, "Vector - Biomethanol Investment Model")

    If Answer = vbNo Then Exit Sub

    BackUpEnvironment False, "Solving facility limit."

    SolveFacilityLimit

    RestoreEnvironment

End Sub
```

**LAUNCH INFORMATION**

For time expensive or extensive routines, it is very helpful to make sure the user wants to run the routine. It is very easily when working too fast to click the wrong button or even do it by accident. This also gives you the opportunity to set expectations. Small things like this contribute to higher UX for minimal extra work.

# INFORMING THE USER

Whilst a routine is running it requires only a simple line to update the status bar within Excel to inform the user of progress.

**STATUS BAR**

The status bar is a useful place to update the user on progress – this is also very useful to you as the developer. Commands are readily sent to it with simple syntax. This does not add a speed overhead.

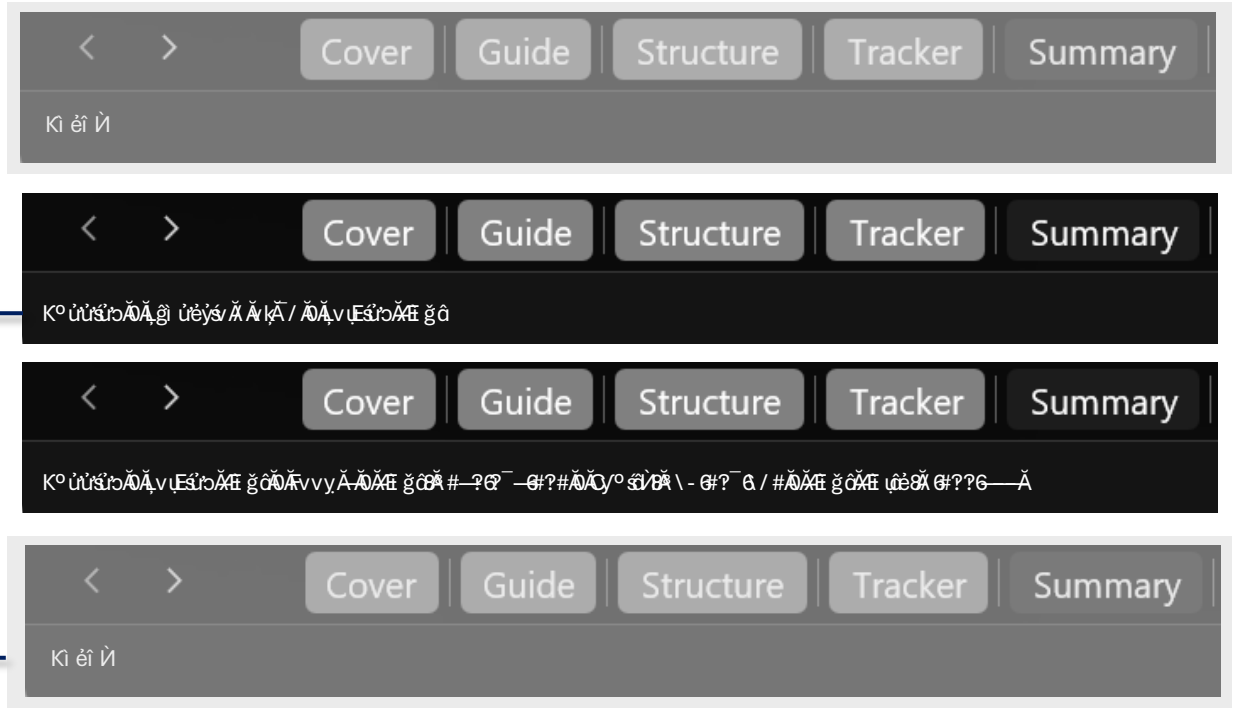
**GOOD HOUSEKEEPING**

Remember to set it back to Ready once completed with the command:

Application.StatusBar = False

**EASY AND FLEXIBLE SYNTAX**

Whatever text is contained within quotes will be displayed as static text. Information is joined using & - called a string operator. Here variables have been included which you can see are the basis of the status bar in the second image.




```
Application.StatusBar = "Running | " & "Scenerio " & ScenarioLoop & " of " & ScenarioLoopMax & " | Solving Debt"
```

# INFORMING THE USER

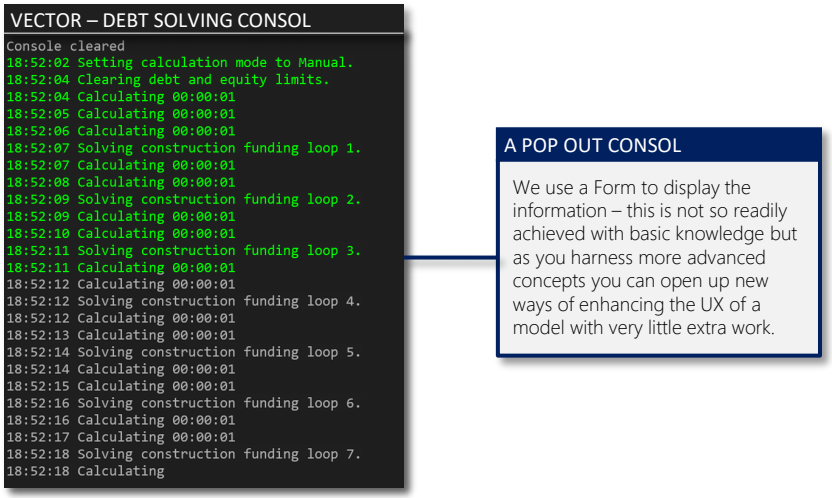
For debt-solving routines, it is very helpful to know how long they take so that if something changes and they take longer, you can respond accordingly.

As debt is solved, it is crucial to track what is happening. This can be achieved by updating cells on a dashboard. Our approach provides the user with a wide range of status updates—an invaluable tool for developers looking to identify calculation inefficiencies.

These updates add no overhead to the solve speed, unlike updating a cell in a worksheet. This console floats like a menu and is particularly useful when structural changes may be causing solve issues.

Once identified, these issues can be resolved or optimised by managing tolerances or adjusting the order of solve loops.

Our primary goal is to gain an in-depth understanding of what is happening under the hood, ensuring we achieve optimal performance.



# BUILD A SCENARIO TABLE

This simple routine for creating a Scenario Table that also solves debt – it's not intended to be a tutorial but to highlight key components covered earlier in the document.

- Embedding code
- Declaring variables with easy-to-read names
- The Structural Anatomy of a Routine
- Calling a Sub Routine
- Using commentary to show what key parts are doing

**CALLING ANOTHER ROUTINE**

If Debt needs to be solved simply provide the name and it will run before this routine continues.

BUILD TABLE

| | | | Project (Unlevered) | | | | Equity (Levered) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | NPV | | IRR | Payback (years) | NPV | | IRR | Payback (years) |
| Scenario | Tests passed | Operations (Yrs) | Proj (USD) | Rep (EUR) | | | Proj (USD) | Rep (EUR) | | |
| Base Case | Pass | 25 | 431.0 | 400.8 | 9.25% | 11.58 | 0.6 | 0.6 | 15.03% | 9.67 |
| Base Case | Pass | 25 | 431.0 | 400.8 | 9.25% | 11.58 | 0.6 | 0.6 | **15.03%** | 9.67 |
| Methanol price down 10% | Pass | 25 | 289.6 | 269.3 | 8.61% | 11.58 | (27.3) | (25.4) | **13.47%** | 9.67 |
| Methanol price down 15% | Pass | 25 | 217.5 | 202.2 | 8.25% | 11.58 | (40.3) | (37.5) | **12.55%** | 9.67 |
| Methanol profile 2 | Pass | 25 | 289.6 | 269.3 | 8.61% | 11.58 | (27.3) | (25.4) | **13.47%** | 9.67 |
| Plant efficiency down X-2% | Pass | 25 | 418.0 | 388.7 | 9.19% | 11.67 | (2.9) | (2.7) | **14.86%** | 9.67 |
| Power price up 20% | Pass | 25 | 369.7 | 343.8 | 8.96% | 11.83 | (15.6) | (14.6) | **14.21%** | 10.17 |
| Con delay 2 months | Pass | 25 | 370.5 | 344.5 | 8.65% | 12.25 | (64.9) | (60.4) | **12.34%** | 12.58 |
| Cons costs up 10% | Pass | 25 | 258.0 | 239.9 | 8.33% | 12.42 | (43.4) | (40.3) | **12.80%** | 10.67 |
| Purchased pellet price up 10% | Pass | 25 | 431.0 | 400.8 | 9.25% | 11.58 | 0.6 | 0.6 | **15.03%** | 9.67 |
| Woodchip price up 10% | Pass | 25 | 367.0 | 341.3 | 8.95% | 11.83 | (16.4) | (15.2) | **14.17%** | 10.17 |
| SOFR+2% | Pass | 25 | 431.0 | 400.8 | 9.25% | 11.58 | 0.6 | 0.6 | **15.03%** | 9.67 |
| Transport price up 10% | Pass | 25 | 431.0 | 400.8 | 9.25% | 11.58 | 0.6 | 0.6 | **15.03%** | 9.67 |
| Transport price up 20% | Pass | 25 | 431.0 | 400.8 | 9.25% | 11.58 | 0.6 | 0.6 | **15.03%** | 9.67 |
| Ramp-up profile 2 | Pass | 25 | 406.6 | 378.2 | 9.12% | 11.75 | (11.5) | (10.7) | **14.45%** | 10.17 |
| People and other costs up 10% | Pass | 25 | 395.7 | 368.0 | 9.08% | 11.75 | (8.8) | (8.2) | **14.56%** | 9.92 |
| Combined case 1 | Pass | 25 | 158.5 | 147.4 | 7.91% | 12.17 | (58.1) | (54.0) | **11.57%** | 10.42 |

```
Sub BuildScenarioTable()

    'Declare variables
    Dim ScenarioNumberActive As Range
    Dim ScenarioNameActive As Range
    Dim ScenarioTableAnchor As Range
    Dim ScenarioTableInner As Range
    Dim ScenarioTableCopy As Range
    Dim DebtMacroRun As Range
    Dim ScenarioNumberOriginal As Integer
    Dim ScenarioLoop As Integer
    Dim ScenarioMax As Integer

    'Establish the link to the ranges in the spreadsheet
    Set ScenarioTableInner = Range("ScenarioTableInner")
    Set ScenarioTableAnchor = Range("ScenarioTableAnchor")
    Set ScenarioNumberActive = Range("Scenario.Number.Active")
    Set ScenarioNameActive = Range("Scenario.Name.Active")
    Set ScenarioTableCopy = Range("ScenarioTableCopy")
    Set DebtMacroRun = Range("Flex.DebtMacroRun")

    'Initialise environment
    ScenarioMax = 20
    ScenarioTableInner.ClearContents
    ScenarioNumberOriginal = ScenarioNumberActive.Value

    For ScenarioLoop = 1 To ScenarioMax

        UpdateMacroConsole "Building scenario " & ScenarioLoop & " of " & ScenarioMax

        'Change the active scenario number (in the spreadsheet) to the value of this loop
        ScenarioNumberActive.Value = ScenarioLoop
        PerformCalculate

        If ScenarioNameActive = "Spare" Then GoTo Continueloop

            'Check if Debt macro needs to be solved (always solve for Base Case)
            If DebtMacroRun = "Yes" Or ScenarioLoop = 1 Then
                SolveFacilityLimit

                'Select the live line of the table
                ScenarioTableCopy.Copy

                'Paste it into the corresponding row
                ScenarioTableAnchor.Offset(ScenarioLoop).PasteSpecial (xlPasteValues)

                'If debt has just run then resolve to Base Case before proceeding with next scenario
                ScenarioNumberActive = 1
                PerformCalculate
                SolveFacilityLimit

            End If

        ScenarioNumberActive.Value = ScenarioLoop
        PerformCalculate

        'Select the live line of the table
        ScenarioTableCopy.Copy

        'Paste it into the corresponding row
        ScenarioTableAnchor.Offset(ScenarioLoop).PasteSpecial (xlPasteValues)
        PerformCalculate

Continueloop:

    Next ScenarioLoop

    'Restore the environment
    UpdateMacroConsole "Restoring original scenario"
    ScenarioNumberActive.Value = ScenarioNumberOriginal
    SolveFacilityLimit
    PerformCalculate
    Application.StatusBar = False

End Sub
```

**THE ENGINE ROOM**

The "Calculation" part is made easier to read and edit because of the house-keeping steps taken above.

# DEBT DASHBOARD

This is the top part of a lite Debt Dashboard which combines the key numerical parts of the Term Sheet with key output to enable sense checking. We lay components out in a sequential manner with a focus on items that give us confidence the solving routines are working. Do not confuse this with an Executive Summary or an Input sheet it is a rare instance of inputs and outputs being on the same worksheet.

## EXPLAIN TO THE USER

In the grey boxes is a brief description of the macros impact and anything that is important to know. It's not extensive – nobody would read it all.

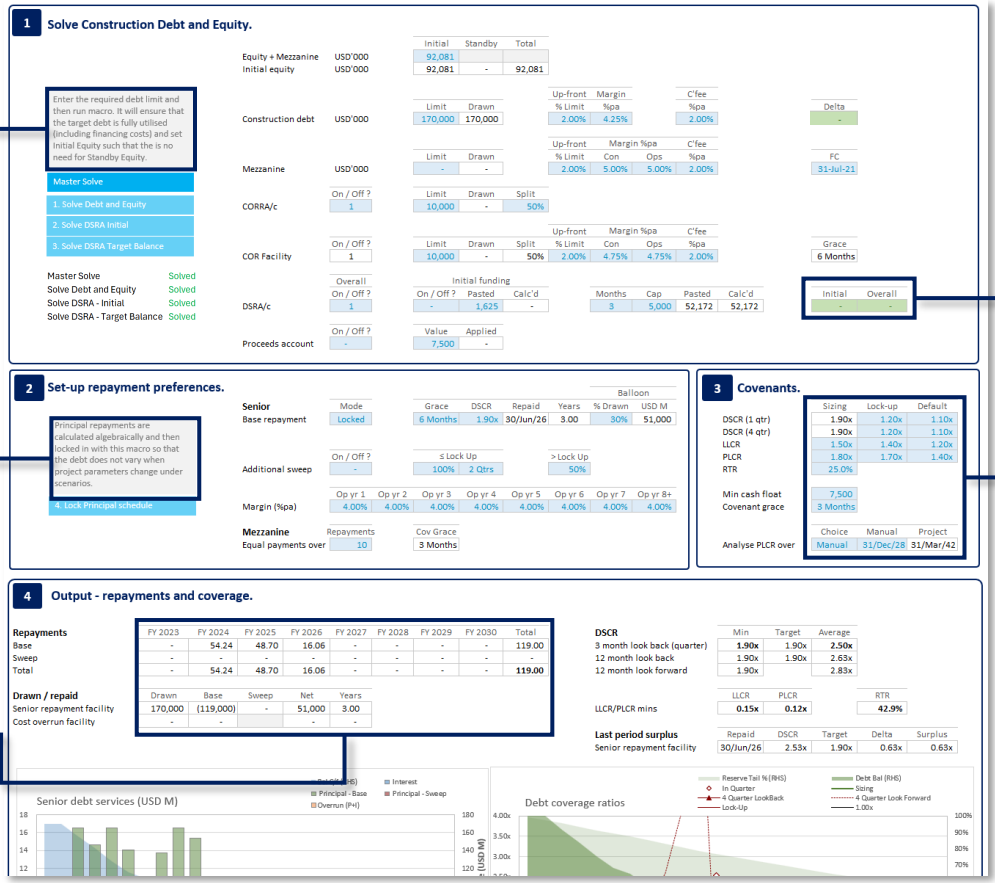
## SOURCE OF INPUTS

In this case the blue cells are input cells but in deals where we are assessing different lenders terms these are often located in a Scenario Manager and linked in.

## SELECTED OUTPUT

In this instance we've collated the principal repayments from a monthly debt sheet into a Financial Year summary – because that's what the deal team were focussing on. Having it here means a user doesn't need to switch between sheets to see what is happening.

It is also another place to build an additional test to be incorporated in the Master Test Panel in the model.




## HIGHLIGHT THE DELTAS

Lite use of Conditional Formatting enables the user to easily visually spot when something has not solved. Green = Ok, Red = Not Ok! This is an example of lowering the Cognitive Load through simple UX.

You can go town with these but with experience you know what the most useful checks are.

## COVENANTS

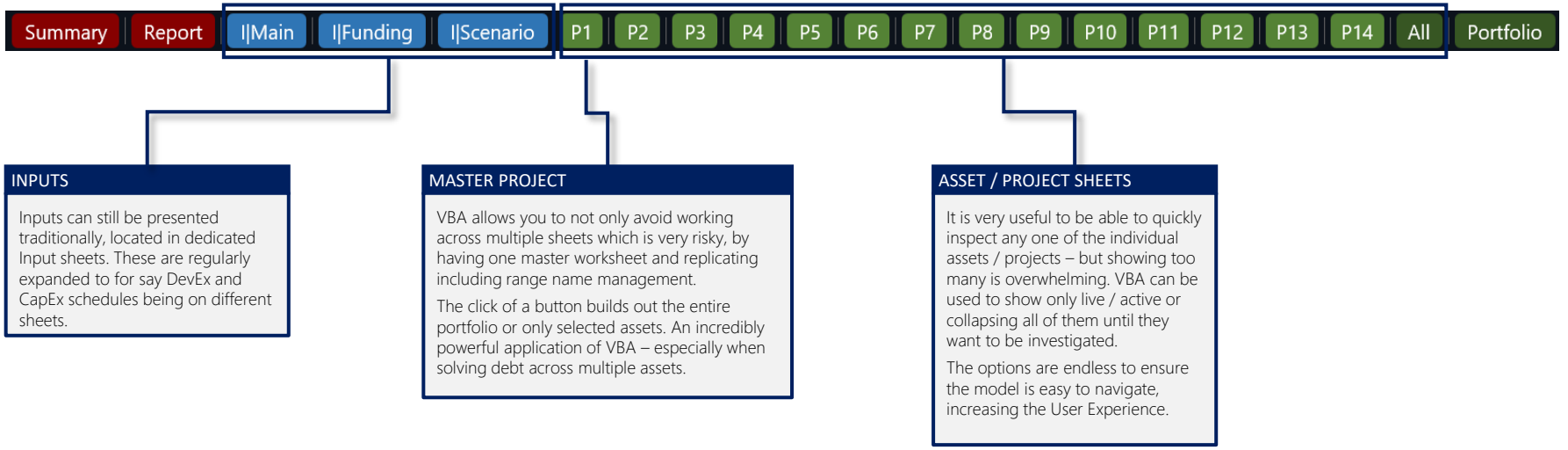
We drive all covenant tests, forward and backward looking from one location so that as the Term Sheet updates an unfamiliar user can make the changes and know what macro needs to re-run.

# PORTFOLIOS

There are several ways to build a portfolio model, over the past few years we have developed many portfolio models and optimised our approach to minimise speed, maximise User Experience and Clarity. There are a lot of considerations to be navigated, and many are non-trivial, but the result includes:

- One master asset (project / worksheet) to be spawned for up to ~250 individual assets (project / worksheet). 250 is extreme but many portfolios commonly have 10+ assets.
- Solve debt at project level, a bundle of projects or at portfolio level.
- Only include live projects , either structurally (only exists when required) or dynamically (exists but is switched off).
- 100+ flexible parameters to be aggregated, at asset level resolution.

Accommodates 1 asset or up to ~250 with existence and inclusion status managed by a dedicated management control panel.



# VECTOR
# WHEN IT MATTERS

Our purpose is to develop elegant solutions for complex situations enabling stakeholders to make confident investment decisions, build team capabilities - as trusted advisors. We can be in for the deal but usually once the deal is done our client's ask us to work with them each subsequent time and train their teams.

- Expertise – our two principals deliver decades of hands-on experience for 100% for your engagement.
- Responsiveness – here when you need us.
- Mission focused – your project is our project.
- Knowledge sharers – no restrictions, no licenses.

If you'd like us on your side in a transaction or to build your teams capabilities contact us at hello@vectorHQ.co