O'REILLY®

2nd Edition

Head First Android Development

A Brain-Friendly Guide

Put fragments under the microscope

Avoid embarrassing activities

Learn how Constraint Layouts can change your life

Create out-of-this-world services

Find your way with Android's Location Services

Fool around in the Design Support Library

Dawn Griffiths & David Griffiths

O'REILLY®

Android Development

What will you learn from this book?

If you have an idea for a killer Android app, this fully revised and updated edition will help you build your first working application in a jiffy. You'll learn hands-on how to structure your app, design flexible and interactive interfaces, run services in the background, make your app work on various smartphones and tablets, and much more. It's like having an experienced Android developer sitting right next to you! All you need to get started is some Java know-how.

Why does this book look so different?

Based on the latest research in cognitive science and learning theory, *Head First Android Development* uses a visually rich format to engage your mind, rather than a text-heavy approach that puts you to sleep. Why waste your time struggling with new concepts? This multi-sensory learning experience is designed for the way your brain really works.

"If you're starting out in mobile development, this is the book for you. It's quite simply the best book on Android development out there."

—*Andy Parker*
Lead Software Developer at Next plc

"This is, without a doubt, the best available book for learning Android development. If you can get only one, make it this one."

—*Kenneth Kousen*
President, Kousen IT Inc., and JavaOne Rock Star

"Become an able Android developer applying up-to-date patterns and create that next killer app. *Head First Android Development* will be your friendly, accurate, and fun-to-be-with master craftsman on that path."

—*Ingo Krotzky*
Android Learner

US $69.99 CAN $92.99

ISBN: 978-1-491-97405-6

Twitter: @oreillymedia
facebook.com/oreilly

Head First
Android Development

Dawn Griffiths
David Griffiths

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Head First Android Development

by Dawn Griffiths and David Griffiths

Copyright © 2017 David Griffiths and Dawn Griffiths. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Series Creators: Kathy Sierra, Bert Bates

Editor: Dawn Schanafelt

Cover Designer: Karen Montgomery

Production Editor: Kristen Brown

Proofreader: Rachel Monaghan

Indexer: Angela Howard

Page Viewers: Mum and Dad, Rob and Lorraine

Printing History:

June 2015: First Edition.
August 2017: Second Edition

Mum and Dad

Rob and Lorraine

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The *Head First* series designations, *Head First Android Development*, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

No kittens were harmed in the making of this book, but several pizzas were eaten.

ISBN: 978-1-491-97405-6
[M]

To our friends and family. Thank you so much for all your love and support.

Authors of Head First Android Development

Dawn Griffiths

David Griffiths

Dawn Griffiths started life as a mathematician at a top UK university, where she was awarded a first-class honors degree in mathematics. She went on to pursue a career in software development and has over 20 years' experience working in the IT industry.

Before writing *Head First Android Development*, Dawn wrote three other Head First books (*Head First Statistics*, *Head First 2D Geometry*, and *Head First C*). She also created the video course *The Agile Sketchpad* with her husband, David, to teach key concepts and techniques in a way that keeps your brain active and engaged.

When Dawn's not working on Head First books or creating videos, you'll find her honing her Tai Chi skills, reading, running, making bobbin lace, or cooking. She particularly enjoys spending time with her wonderful husband, David.

David Griffiths began programming at age 12, when he saw a documentary on the work of Seymour Papert. At age 15, he wrote an implementation of Papert's computer language LOGO. After studying pure mathematics at university, he began writing code for computers and magazine articles for humans. He's worked as an Agile coach, a developer, and a garage attendant, but not in that order. He can write code in over 10 languages and prose in just one, and when not writing, coding, or coaching, he spends much of his spare time traveling with his lovely wife—and coauthor—Dawn.

Before writing *Head First Android Development*, David wrote three other Head First books—*Head First Rails*, *Head First Programming*, and *Head First C*—and created *The Agile Sketchpad* video course with Dawn.

You can follow us on Twitter at *https://twitter.com/HeadFirstDroid* and visit the book's website at *https://tinyurl.com/HeadFirstAndroid*.

Table of Contents (Summary)

	Intro	xxix
1	Getting Started: *Diving in*	1
2	Building Interactive Apps: *Apps that do something*	37
3	Multiple Activities and Intents: *State your intent*	77
4	The Activity Lifecycle: *Being an activity*	119
5	Views and View Groups *Enjoy the view*	169
6	Constraint Layouts: *Put things in their place*	221
7	List views and Adapters: *Getting organized*	247
8	Support Libraries and App Bars: *Taking shortcuts*	289
9	Fragments: *Make it modular*	339
10	Fragments for Larger Interfaces: *Different size, different interface*	393
11	Dynamic Fragments: *Nesting fragments*	433
12	Design Support Library: *Swipe right*	481
13	Recycler Views and Card Views: *Get recycling*	537
14	Navigation Drawers: *Going places*	579
15	SQLite Databases: *Fire up the database*	621
16	Basic cursors: *Getting data out*	657
17	Cursors and AsyncTasks: *Staying in the background*	693
18	Started Services: *At your service*	739
19	Bound Services and Permissions: *Bound together*	767
i	Relative and Grid Layouts: *Meet the relatives*	817
ii	Gradle: *The Gradle build tool*	833
iii	ART: *The Android Runtime*	841
iv	ADB: *The Android debug bridge*	849
v	The Android Emulator: *Speeding things up*	857
vi	Leftovers: *The top ten things (we didn't cover)*	861

Table of Contents (the real thing)

Intro

Your brain on Android. Here *you* are trying to *learn* something, while here your *brain* is, doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing how to develop Android apps?

Authors of Head First Android Development	iv
Who is this book for?	xxx
We know what you're thinking	xxxi
We know what your *brain* is thinking	xxxi
Metacognition: thinking about thinking	xxxiii
Here's what WE did	xxxiv
Read me	xxxvi
The technical review team	xxxviii
Acknowledgments	xxxix
Safari® Books Online	xl

getting started

1 Diving In

Android has taken the world by storm.

Everybody wants a smartphone or tablet, and Android devices are hugely popular. In this book, we'll teach you how to **develop your own apps**, and we'll start by getting you to build a basic app and run it on an Android Virtual Device. Along the way, you'll meet some of the basic components of all Android apps, such as **activities** and **layouts**. **All you need is a little Java know-how...**

Welcome to Androidville	2
The Android platform dissected	3
Here's what we're going to do	4
Your development environment	5
Install Android Studio	6
Build a basic app	7
How to build the app	8
Activities and layouts from 50,000 feet	12
How to build the app (continued)	13
You've just created your first Android app	15
Android Studio creates a complete folder structure for you	16
Useful files in your project	17
Edit code with the Android Studio editors	18
Run the app in the Android emulator	23
Creating an Android Virtual Device	24
Run the app in the emulator	27
You can watch progress in the console	28
What just happened?	30
Refining the app	31
What's in the layout?	32
activity_main.xml has two elements	33
Update the text displayed in the layout	34
Take the app for a test drive	35
Your Android Toolbox	36

building interactive apps

2 Apps That Do Something

Most apps need to respond to the user in some way.

In this chapter, you'll see how you can make your apps **a bit more interactive**. You'll learn how to get your app to ***do*** something in response to the user, and **how to get your activity and layout talking to each other** like best buddies. Along the way, we'll take you a bit **deeper into how Android actually works** by introducing you to **R**, the hidden gem that glues everything together.

Let's build a Beer Adviser app	38
Create the project	40
We've created a default activity and layout	41
A closer look at the design editor	42
Add a button using the design editor	43
activity_find_beer.xml has a new button	44
A closer look at the layout code	45
Let's take the app for a test drive	49
Hardcoding text makes localization hard	50
Create the String resource	51
Use the String resource in your layout	52
The code for activity_find_beer.xml	53
Add values to the spinner	56
Add the string-array to strings.xml	57
Test drive the spinner	58
We need to make the button do something	59
Make the button call a method	60
What activity code looks like	61
Add an onClickFindBeer() method to the activity	62
onClickFindBeer() needs to do something	63
Once you have a View, you can access its methods	64
Update the activity code	65
The first version of the activity	67
What the code does	68
Build the custom Java class	70
What happens when you run the code	74
Test drive your app	75
Your Android Toolbox	76

3 multiple activities and intents

State Your Intent

Most apps need more than one activity.

So far we've just looked at single-activity apps, which is fine for simple apps. But when things get more complicated, just having the one activity won't cut it. We're going to show you **how to build apps with multiple activities**, and how you can get your apps talking to each other using ***intents***. We'll also look at how you can use intents to **go beyond the boundaries of your app** and **make activities in other apps on your device perform *actions***. Things are about to get a whole lot more powerful...

Apps can contain more than one activity	78
Here's the app structure	79
Get started: create the project	79
Update the layout	80
Create the second activity and layout	82
Welcome to the Android manifest file	84
An intent is a type of message	86
What happens when you run the app	88
Pass text to a second activity	90
Update the text view properties	91
putExtra() puts extra info in an intent	92
Update the CreateMessageActivity code	95
Get ReceiveMessageActivity to use the information in the intent	96
What happens when the user clicks the Send Message button	97
We can change the app to send messages to other people	98
How Android apps work	99
Create an intent that specifies an action	101
Change the intent to use an action	102
How Android uses the intent filter	106
What if you ALWAYS want your users to choose an activity?	112
What happens when you call createChooser()	113
Change the code to create a chooser	115
Your Android Toolbox	118

the activity lifecycle

4 Being an Activity

Activities form the foundation of every Android app.

So far you've seen how to create activities, and made one activity start another using an intent. But ***what's really going on beneath the hood?*** In this chapter, we're going to dig a little deeper into **the activity lifecycle**. What happens when an activity is **created** and **destroyed**? Which methods get called when an activity is **made visible and appears in the foreground**, and which get called when the activity **loses the focus and is hidden**? And **how do you save and restore your activity's state**? Read on to find out.

How do activities really work? 120
The Stopwatch app 122
Add String resources 123
How the activity code will work 125
Add code for the buttons 126
The runTimer() method 127
The full runTimer() code 129
The full StopwatchActivity code 130
Rotating the screen changes the device configuration 136
The states of an activity 137
The activity lifecycle: from create to destroy 138
The updated StopwatchActivity code 142
What happens when you run the app 143
There's more to an activity's life than create and destroy 146
The updated StopwatchActivity code 151
What happens when you run the app 152
What if an app is only partially visible? 154
The activity lifecycle: the foreground lifetime 155
Stop the stopwatch if the activity's paused 158
Implement the onPause() and onResume() methods 159
The complete StopwatchActivity code 160
What happens when you run the app 163
Your handy guide to the lifecycle methods 167
Your Android Toolbox 168

5 views and view groups

Enjoy the View

You've seen how to arrange GUI components using a linear layout, but so far we've only scratched the surface.

In this chapter we'll **look a little deeper** and show you how linear layouts *really work*. We'll introduce you to the **frame layout**, a simple layout used to stack views, and we'll also take a tour of the **main GUI components** and **how you use them**. By the end of the chapter, you'll see that even though they all look a little different, all layouts and GUI components have ***more in common than you might think***.

Frame layouts let your views overlap one another. This is useful for displaying text on top of images, for example.

Your user interface is made up of layouts and GUI components	170
LinearLayout displays views in a single row or column	171
Add a dimension resource file for consistent padding across layouts	174
Use margins to add distance between views	176
Let's change up a basic linear layout	177
Make a view streeeeetch by adding weight	179
Values you can use with the android:gravity attribute	183
The full linear layout code	186
Frame layouts stack their views	188
Add an image to your project	189
The full code to nest a layout	192
FrameLayout: a summary	193
Playing with views	201
Editable text view	202
Toggle button	204
Switch	205
Checkboxes	206
Radio buttons	208
Spinner	210
Image view	211
Adding images to buttons	213
Scroll views	215
Toasts	216
Your Android Toolbox	220

constraint layouts

6 Put Things in Their Place

Let's face it, you need to know how to create great layouts.

If you're building apps you want people to ***use***, you need to make sure they **look exactly the way you want**. So far you've seen how to use linear and frame layouts, but *what if your design is more complex?* To deal with this, we'll introduce you to Android's new **constraint layout**, a type of layout you ***build visually using a blueprint***. We'll show you how **constraints** let you position and size your views, *irrespective of screen size and orientation.* Finally, you'll find out how to save time by making Android Studio **infer and add constraints** on your behalf.

Nested layouts can be inefficient	222
Introducing the Constraint Layout	223
Make sure your project includes the Constraint Layout Library	224
Add the String resources to strings.xml	225
Use the blueprint tool	226
Position views using constraints	227
Add a vertical constraint	228
Changes to the blueprint are reflected in the XML	229
How to center views	230
Adjust a view's position by updating its bias	231
How to change a view's size	232
How to align views	238
Let's build a real layout	239
First, add the top line of views	240
The Infer Constraints feature guesses which constraints to add	241
Add the next line to the blueprint...	242
Finally, add a view for the message	243
Test drive the app	244
Your Android Toolbox	245

7 list views and adapters

Getting Organized

Want to know how best to structure your Android app?

You've learned about some of the basic building blocks that are used to create apps, and now **it's time to get organized**. In this chapter, we'll show you how you can take a bunch of ideas and **structure them into an awesome app**. You'll learn how **lists of data** can form the core part of your app design, and how **linking them together** can create a **powerful and easy-to-use app**. Along the way, you'll get your first glimpse of using **event listeners** and **adapters** to make your app more dynamic.

Display a start screen with a list of options.

Display a list of the drinks we sell.

Show details of each drink.

Every app starts with ideas	248
Use list views to navigate to data	251
The drink detail activity	253
The Starbuzz app structure	254
The Drink class	256
The top-level layout contains an image and a list	258
The full top-level layout code	260
Get list views to respond to clicks with a listener	261
Set the listener to the list view	262
A category activity displays the data for a single category	267
Update activity_drink_category.xml	268
For nonstatic data, use an adapter	269
Connect list views to arrays with an array adapter	270
Add the array adapter to DrinkCategoryActivity	271
App review: where we are	274
How we handled clicks in TopLevelActivity	276
The full DrinkCategoryActivity code	278
Update the views with the data	281
The DrinkActivity code	283
What happens when you run the app	284
Your Android Toolbox	288

support libraries and app bars

8 Taking Shortcuts

Everybody likes a shortcut.

And in this chapter you'll see how to add shortcuts to your apps using **app bars**. We'll show you how to start activities by *adding actions* to your app bar, how to share content with other apps using the *share action provider*, and how to navigate up your app's hierarchy by implementing *the app bar's Up button*. Along the way we'll introduce you to the powerful **Android Support Libraries**, which are key to making your apps look fresh on older versions of Android.

Great apps have a clear structure	290
Different types of navigation	291
Add an app bar by applying a theme	293
Create the Pizza app	295
Add the v7 AppCompat Support Library	296
AndroidManifest.xml can change your app bar's appearance	299
How to apply a theme	300
Define styles in a style resource file	301
Customize the look of your app	303
Define colors in a color resource file	304
The code for activity_main.xml	305
ActionBar vs. Toolbar	306
Include the toolbar in the activity's layout	312
Add actions to the app bar	315
Change the app bar text by adding a label	318
The code for AndroidManifest.xml	319
Control the action's appearance	322
The full MainActivity.java code	325
Enable Up navigation	327
Share content on the app bar	331
Add a share action provider to menu_main.xml	332
Specify the content with an intent	333
The full MainActivity.java code	334
Your Android Toolbox	337

9 fragments

Make It Modular

You've seen how to create apps that work in the same way no matter what device they're running on.

But what if you want your app to look and behave differently depending on whether it's running on a *phone* or a *tablet*? In this case you need **fragments**, modular code components that can be **reused by different activities**. We'll show you how to create **basic fragments** and **list fragments**, how to **add them to your activities**, and how to get your fragments and activities to **communicate** with one another.

Your app needs to look great on ALL devices	340
Your app may need to behave differently too	341
Fragments allow you to reuse code	342
The phone version of the app	343
Create the project and activities	345
Add a button to MainActivity's layout	346
How to add a fragment to your project	348
The fragment's onCreateView() method	350
Add a fragment to an activity's layout	352
Get the fragment and activity to interact	359
The Workout class	360
Pass the workout ID to the fragment	361
Get the activity to set the workout ID	363
The fragment lifecycle	365
Set the view's values in the fragment's onStart() method	367
How to create a list fragment	374
The updated WorkoutListFragment code	377
The code for activity_main.xml	381
Connect the list to the detail	384
The code for WorkoutListFragment.java	387
MainActivity needs to implement the interface	388
DetailActivity needs to pass the ID to WorkoutDetailFragment	389
Your Android Toolbox	392

10 fragments for larger interfaces

Different Size, Different Interface

So far we've only run our apps on devices with a small screen. But what if your users have tablets? In this chapter you'll see how to create **flexible user interfaces** by making your app **look and behave differently** depending on the device it's running on. We'll show you how to control the behavior of your app when you press the Back button by introducing you to the **back stack** and **fragment transactions**. Finally, you'll find out how to **save and restore the state** of your fragment.

The Workout app looks the same on a phone and a tablet	394
Designing for larger interfaces	395
The phone version of the app	396
The tablet version of the app	397
Create a tablet AVD	399
Put screen-specific resources in screen-specific folders	402
The different folder options	403
Tablets use layouts in the layout-large folder	408
What the updated code does	410
We need to change the itemClicked() code	412
You want fragments to work with the Back button	413
Welcome to the back stack	414
Back stack transactions don't have to be activities	415
Use a frame layout to replace fragments programmatically	416
Use layout differences to tell which layout the device is using	417
The revised MainActivity code	418
Using fragment transactions	419
The updated MainActivity code	423
Rotating the tablet breaks the app	427
Saving an activity's state (revisited)	428
The updated code for WorkoutDetailFragment.java	430
Your Android Toolbox	432

dynamic fragments

11 Nesting Fragments

So far you've seen how to create and use static fragments. But what if you want your fragments to be more **dynamic**? Dynamic fragments have a lot in common with dynamic activities, but there are crucial differences you need to be able to deal with. In this chapter you'll see how to **convert dynamic activities** into **working dynamic fragments**. You'll find out how to use **fragment transactions** to help **maintain your fragment state**. Finally, you'll discover how to **nest one fragment inside another**, and how the **child fragment manager** helps you control unruly back stack behavior.

Adding dynamic fragments	434
The new version of the app	436
Create TempActivity	437
TempActivity needs to extend AppCompatActivity	438
The StopwatchFragment.java code	444
The StopwatchFragment layout	447
Add StopwatchFragment to TempActivity's layout	449
The onClick attribute calls methods in the activity, not the fragment	452
Attach the OnClickListener to the buttons	457
The StopwatchFragment code	458
Rotating the device resets the stopwatch	462
Use <fragment> for static fragments...	463
Change activity_temp.xml to use a FrameLayout	464
The full code for TempActivity.java	467
Add the stopwatch to WorkoutDetailFragment	469
The full WorkoutDetailFragment.java code	476
Your Android Toolbox	480

Whenever I see android:onClick, I assume it's all about **me**. **My** methods run, not the fragment's.

I display workout details, and I also display the stopwatch.

The transaction to add StopwatchFragment is nested inside the transaction to add WorkoutDetailFragment.

12 design support library

Swipe Right

Ever wondered how to develop apps with a rich, slick UI?

With the release of the **Android Design Support Library**, it became much easier to create apps with an intuitive UI. In this chapter, we'll show you around some of the highlights. You'll see how to add ***tabs*** so that your users can *navigate around your app more easily*. You'll discover how to ***animate your toolbars*** so that they can *collapse or scroll on a whim*. You'll find out how to add ***floating action buttons*** for common user actions. Finally, we'll introduce you to ***snackbars***, a way of displaying short, informative messages to the user that they can interact with.

We'll get the toolbar to scroll when the user scrolls the content in TopFragment.

We'll add this scrollable content to TopFragment.

The Bits and Pizzas app revisited	482
The app structure	483
Use a view pager to swipe through fragments	489
Add a view pager to MainActivity's layout	490
Tell a view pager about its pages using a fragment pager adapter	491
The code for our fragment pager adapter	492
The full code for MainActivity.java	494
Add tab navigation to MainActivity	498
How to add tabs to your layout	499
Link the tab layout to the view pager	501
The full code for MainActivity.java	502
The Design Support Library helps you implement material design	506
Make the toolbar respond to scrolls	508
Add a coordinator layout to MainActivity's layout	509
How to coordinate scroll behavior	510
Add scrollable content to TopFragment	512
The full code for fragment_top.xml	515
Add a collapsing toolbar to OrderActivity	517
How to create a plain collapsing toolbar	518
How to add an image to a collapsing toolbar	523
The updated code for activity_order.xml	524
FABs and snackbars	526
The updated code for activity_order.xml	528
The full code for OrderActivity.java	533
Your Android Toolbox	535

recycler views and card views

13 Get Recycling

You've already seen how the humble list view is a key part of most apps. But compared to some of the *material design* components we've seen, it's somewhat plain. In this chapter, we'll introduce you to the **recycler view**, a more advanced type of list that gives you *loads more flexibility* and *fits in with the material design ethos*. You'll see how to create **adapters** tailored to your data, and how to completely change the look of your list with *just two lines of code*. We'll also show you how to use **card views** to give your data a *3D material design* appearance.

Each of our ViewHolders will contain a CardView. We created the layout for this CardView earlier in the chapter.

There's still work to do on the Bits and Pizzas app 538
Recycler views from 10,000 feet 539
Add the pizza data 541
Display the pizza data in a card 542
How to create a card view 543
The full card_captioned_image.xml code 544
Add a recycler view adapter 546
Define the adapter's view holder 548
Override the onCreateViewHolder() method 549
Add the data to the card views 550
The full code for CaptionedImagesAdapter.java 551
Create the recycler view 553
Add the RecyclerView to PizzaFragment's layout 554
The full PizzaFragment.java code 555
A recycler view uses a layout manager to arrange its views 556
Specify the layout manager 557
The full PizzaFragment.java code 558
Make the recycler view respond to clicks 566
Create PizzaDetailActivity 567
The code for PizzaDetailActivity.java 569
Get a recycler view to respond to clicks 570
You can listen for view events from the adapter 571
Keep your adapters reusable 572
Add the interface to the adapter 573
Implement the listener in PizzaFragment.java 575
Your Android Toolbox 578

navigation drawers

14 Going Places

You've already seen how tabs help users navigate your apps. But if you need a *large number* of them, or want to *split them into sections*, the **navigation drawer** is your new BFF. In this chapter, we'll show you how to create a navigation drawer that *slides out from the side of your activity at a single touch*. You'll learn how to give it a header using a **navigation view**, and provide it with a **structured set of menu items** to take the user to all the major hubs of your app. Finally, you'll discover how to set up a **navigation view listener** so that the drawer *responds to the slightest touch and swipe*.

This is the CatChat app.

Tab layouts allow easy navigation...	580
We're going to create a navigation drawer for a new email app	581
Navigation drawers deconstructed	582
Create the CatChat project	584
Create InboxFragment	585
Create DraftsFragment	586
Create SentItemsFragment	587
Create TrashFragment	588
Create a toolbar layout	589
Update the app's theme	590
Create HelpActivity	591
Create FeedbackActivity	592
Create the navigation drawer's header	594
The full nav_header.xml code	595
How to group items together	598
Add the support section as a submenu	600
The full menu_nav.xml code	601
How to create a navigation drawer	602
The full code for activity_main.xml	603
Add InboxFragment to MainActivity's frame layout	604
Add a drawer toggle	607
Respond to the user clicking items in the drawer	608
Implement the onNavigationItemSelected() method	609
Close the drawer when the user presses the Back button	614
The full MainActivity.java code	615
Your Android Toolbox	619

SQLite databases

15 Fire Up the Database

If you're recording high scores or saving tweets, your app will need to store data. And on Android you usually keep your data safe inside a **SQLite database**. In this chapter, we'll show you how to **create a database, add tables to it, and prepopulate it with data**, all with the help of the friendly **SQLite helper**. You'll then see how you can cleanly roll out **upgrades** to your database structure, and how to **downgrade** it if you need to undo any changes.

Back to Starbuzz	622
Android uses SQLite databases to persist data	623
Android comes with SQLite classes	624
The current Starbuzz app structure	625
Let's change the app to use a database	626
The SQLite helper manages your database	627
Create the SQLite helper	628
Inside a SQLite database	630
You create tables using Structured Query Language (SQL)	631
Insert data using the insert() method	632
Insert multiple records	633
The StarbuzzDatabaseHelper code	634
What the SQLite helper code does	635
What if you need to make changes to the database?	636
SQLite databases have a version number	637
What happens when you change the version number	638
Upgrade your database with onUpgrade()	640
Downgrade your database with onDowngrade()	641
Let's upgrade the database	642
Upgrade an existing database	645
Update records with the update() method	646
Apply conditions to multiple columns	647
Change the database structure	649
Delete tables by dropping them	650
The full SQLite helper code	651
Your Android Toolbox	656

basic cursors

16 Getting Data Out

So how do you connect your app to a SQLite database?

So far you've seen how to create a SQLite database using a SQLite helper. The next step is to get your activities to access it. In this chapter, we'll focus on how you read data from a database. You'll find out **how to use cursors to get data from the database**. You'll see **how to navigate cursors,** and **how to get access to their data**. Finally, you'll discover how to use **cursor adapters** to bind cursors to list views.

The story so far... 658
The new Starbuzz app structure 659
What we'll do to change DrinkActivity to use the Starbuzz database 660
The current DrinkActivity code 661
Get a reference to the database 662
Get data from the database with a cursor 663
Return all the records from a table 664
Return records in a particular order 665
Return selected records 666
The DrinkActivity code so far 669
To read a record from a cursor, you first need to navigate to it 670
Navigate cursors 671
Get cursor values 672
The DrinkActivity code 673
What we've done so far 675
The current DrinkCategoryActivity code 677
Get a reference to the Starbuzz database... 678
How do we replace the array data in the list view? 679
A simple cursor adapter maps cursor data to views 680
How to use a simple cursor adapter 681
Close the cursor and database 682
The story continues 683
The revised code for DrinkCategoryActivity 688
The DrinkCategoryActivity code (continued) 689
Your Android Toolbox 691

cursors and asynctasks

17 Staying in the Background

In most apps, you'll need your app to update its data.

So far you've seen how to create apps that read data from a SQLite database. But what if you want to update the app's data? In this chapter you'll see how to get your app to **respond to user input** and **update values in the database**. You'll also find out how to **refresh the data that's displayed** once it's been updated. Finally, you'll see how writing efficient **multithreaded code** with **AsyncTask**s will keep your app speedy.

We want our Starbuzz app to update database data	694
Add a checkbox to DrinkActivity's layout	696
Display the value of the FAVORITE column	697
Respond to clicks to update the database	698
The full DrinkActivity.java code	701
Display favorites in TopLevelActivity	705
Refactor TopLevelActivity.java	707
The new TopLevelActivity.java code	710
Change the cursor with changeCursor()	715
What code goes on which thread?	723
AsyncTask performs asynchronous tasks	724
The onPreExecute() method	725
The doInBackground() method	726
The onProgressUpdate() method	727
The onPostExecute() method	728
The AsyncTask class parameters	729
The full UpdateDrinkTask class	730
The full DrinkActivity.java code	732
A summary of the AsyncTask steps	737
Your Android Toolbox	737

18 started services

At Your Service

There are some operations you want to keep on running, irrespective of which app has the focus. If you start downloading a file, for instance, *you don't want the download to stop when you switch to another app*. In this chapter we'll introduce you to **started services**, components that *run operations in the background*. You'll see how to create a started service using the `IntentService` class, and find out how its lifecycle fits in with that of an activity. Along the way, you'll discover how to **log messages**, and *keep users informed* using Android's built-in **notification service**.

Services work in the background	740
We'll create a STARTED service	741
Use the IntentService class to create a basic started service	742
How to log messages	743
The full DelayedMessageService code	744
You declare services in AndroidManifest.xml	745
Add a button to activity_main.xml	746
You start a service using startService()	747
The states of a started service	750
The started service lifecycle: from create to destroy	751
Your service inherits the lifecycle methods	752
Android has a built-in notification service	755
We'll use notifications from the AppCompat Support Library	756
First create a notification builder	757
Issue the notification using the built-in notification service	759
The full code for DelayedMessageService.java	760
Your Android Toolbox	765

bound services and permissions

19 Bound Together

Started services are great for background operations, but what if you need a service that's more interactive? In this chapter you'll discover how to create a **bound service**, *a type of service your activity can interact with*. You'll see how to ***bind*** to the service when you need it, and how to ***unbind*** from it when you're done to save resources. You'll find out how to use **Android's Location Services** to get *location updates from your device GPS*. Finally, you'll discover how to use **Android's permission model**, including *handling runtime permission requests*.

Bound services are bound to other components	768
Create a new service	770
Implement a binder	771
Add a getDistance() method to the service	772
Update MainActivity's layout	773
Create a ServiceConnection	775
Use bindService() to bind the service	778
Use unbindService() to unbind from the service	779
Call OdometerService's getDistance() method	780
The full MainActivity.java code	781
The states of a bound service	787
Add the AppCompat Support Library	790
Add a location listener to OdometerService	792
Here's the updated OdometerService code	795
Calculate the distance traveled	796
The full OdometerService.java code	798
Get the app to request permission	802
Check the user's response to the permission request	805
Add notification code to onRequestPermissionsResults()	809
The full code for MainActivity.java	811
Your Android Toolbox	815
It's been great having you here in Androidville	816

relative and grid layouts

Meet the Relatives

There are two more layouts you will often meet in Androidville. In this book we've concentrated on using simple *linear and frame layouts*, and introduced you to *Android's new constraint layout*. But there are two more layouts we'd like you to know about: **relative layouts** and **grid layouts**. They've largely been superseded by the constraint layout, but we have a soft spot for them, and we think they'll stay around for a few more years.

gradle

The Gradle Build Tool

Most Android apps are created using a build tool called Gradle. **Gradle** works behind the scenes to find and download libraries, compile and deploy your code, run tests, clean the grouting, and so on. Most of the time you *might not even realize it's there* because Android Studio provides a graphical interface to it. Sometimes, however, it's helpful to dive into Gradle and **hack it manually**. In this appendix we'll introduce you to some of Gradle's many talents.

iii

art

The Android Runtime

Ever wonder how Android apps can run on so many kinds of devices? Android apps run in a virtual machine called the **Android runtime (ART)**, not the Oracle Java Virtual Machine (JVM). This means that your apps are quicker to start on small, low-powered devices and run more efficiently. In this appendix, we'll look at how ART works.

iv

adb

The Android Debug Bridge

In this book, we've focused on using an IDE for all your Android needs. But there are times when using a command-line tool can be plain useful, like those times when Android Studio can't see your Android device but you just *know* it's there. In this chapter, we'll introduce you to the **Android Debug Bridge (or adb)**, a command-line tool you can use to communicate with the emulator or Android devices.

the android emulator

Speeding Things Up

V

Ever felt like you were spending all your time waiting for the emulator? There's no doubt that using the Android emulator is useful. It allows you to see how your app will run on devices other than the physical ones you have access to. But at times it can feel a little...sluggish. In this appendix, we'll explain why the emulator can seem slow. Even better, we'll give you a few tips we've learned for **speeding it up**.

leftovers

The Top Ten Things (we didn't cover)

Vi

Even after all that, there's still a little more.

There are just a few more things we think you need to know. We wouldn't feel right about ignoring them, and we really wanted to give you a book you'd be able to lift without extensive training at the local gym. Before you put down the book, **read through these tidbits**.

1. Distributing your app 862
2. Content providers 863
3. Loaders 864
4. Sync adapters 864
5. Broadcasts 865
6. The WebView class 866
7. Settings 867
8. Animation 868
9. App widgets 869
10. Automated testing 870

how to use this book

Intro

In this section, we answer the burning question:
"So why DID they put that in a book on Android?"

Who is this book for?

If you can answer "yes" to all of these:

1. Do you already know how to program in Java?
2. Do you want to master Android app development, create the next big thing in software, make a small fortune, and retire to your own private island?
3. Do you prefer actually doing things and applying the stuff you learn over listening to someone in a lecture rattle on for hours on end?

OK, maybe that one's a little far-fetched. But you gotta start somewhere, right?

this book is for you.

Who should probably back away from this book?

If you can answer "yes" to any of these:

1. Are you looking for a quick introduction or reference book to developing Android apps?
2. Would you rather have your toenails pulled out by 15 screaming monkeys than learn something new? Do you believe an Android book should cover *everything*, especially all the obscure stuff you'll never use, and if it bores the reader to tears in the process, then so much the better?

this book is ***not*** for you.

[Note from Marketing: this book is for anyone with a credit card or a PayPal account]

We know what you're thinking

"How can *this* be a serious book on developing Android apps?"

"What's with all the graphics?"

"Can I actually *learn* it this way?"

We know what your *brain* is thinking

Your brain craves novelty. It's always searching, scanning, *waiting* for something unusual. It was built that way, and it helps you stay alive.

So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it *can* to stop them from interfering with the brain's *real* job—recording things that *matter*. It doesn't bother saving the boring things; they never make it past the "this is obviously not important" filter.

How does your brain *know* what's important? Suppose you're out for a day hike and a tiger jumps in front of you—what happens inside your head and body?

Neurons fire. Emotions crank up. *Chemicals surge*.

And that's how your brain knows…

This must be important! Don't forget it!

But imagine you're at home or in a library. It's a safe, warm, tiger-free zone. You're studying. Getting ready for an exam. Or trying to learn some tough technical topic your boss thinks will take a week, ten days at the most.

Just one problem. Your brain's trying to do you a big favor. It's trying to make sure that this *obviously* unimportant content doesn't clutter up scarce resources. Resources that are better spent storing the really *big* things. Like tigers. Like the danger of fire. Like how you should never have posted those party photos on your Facebook page. And there's no simple way to tell your brain, "Hey brain, thank you very much, but no matter how dull this book is, and how little I'm registering on the emotional Richter scale right now, I really *do* want you to keep this stuff around."

We think of a "Head First" reader as a learner.

So what does it take to *learn* something? First, you have to *get* it, then make sure you don't *forget* it. It's not about pushing facts into your head. Based on the latest research in cognitive science, neurobiology, and educational psychology, *learning* takes a lot more than text on a page. We know what turns your brain on.

Some of the Head First learning principles:

Make it visual. Images are far more memorable than words alone, and make learning much more effective (up to 89% improvement in recall and transfer studies). It also makes things more understandable. **Put the words within or near the graphics** they relate to, rather than on the bottom or on another page, and learners will be up to *twice* as likely to solve problems related to the content.

Use a conversational and personalized style. In recent studies, students performed up to 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tell stories instead of lecturing. Use casual language. Don't take yourself too seriously. Which would *you* pay more attention to: a stimulating dinner-party companion, or a lecture?

Get the learner to think more deeply. In other words, unless you actively flex your neurons, nothing much happens in your head. A reader has to be motivated, engaged, curious, and inspired to solve problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking questions, and activities that involve both sides of the brain and multiple senses.

Get—and keep—the reader's attention. We've all had the "I really want to learn this, but I can't stay awake past page one" experience. Your brain pays attention to things that are out of the ordinary, interesting, strange, eye-catching, unexpected. Learning a new, tough, technical topic doesn't have to be boring. Your brain will learn much more quickly if it's not.

Touch their emotions. We now know that your ability to remember something is largely dependent on its emotional content. You remember what you care about. You remember when you *feel* something. No, we're not talking heart-wrenching stories about a boy and his dog. We're talking emotions like surprise, curiosity, fun, "what the…?", and the feeling of "I rule!" that comes when you solve a puzzle, learn something everybody else thinks is hard, or realize you know something that "I'm more technical than thou" Bob from Engineering *doesn't*.

Metacognition: thinking about thinking

If you really want to learn, and you want to learn more quickly and more deeply, pay attention to how you pay attention. Think about how you think. Learn how you learn.

Most of us did not take courses on metacognition or learning theory when we were growing up. We were *expected* to learn, but rarely *taught* to learn.

But we assume that if you're holding this book, you really want to learn how to develop Android apps. And you probably don't want to spend a lot of time. If you want to use what you read in this book, you need to *remember* what you read. And for that, you've got to *understand* it. To get the most from this book, or *any* book or learning experience, take responsibility for your brain. Your brain on *this* content.

The trick is to get your brain to see the new material you're learning as Really Important. Crucial to your well-being. As important as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best to keep the new content from sticking.

So just how *DO* you get your brain to treat Android development like it was a hungry tiger?

There's the slow, tedious way, or the faster, more effective way. The slow way is about sheer repetition. You obviously know that you *are* able to learn and remember even the dullest of topics if you keep pounding the same thing into your brain. With enough repetition, your brain says, "This doesn't *feel* important to him, but he keeps looking at the same thing *over* and *over* and *over*, so I suppose it must be."

The faster way is to do ***anything that increases brain activity,*** especially different *types* of brain activity. The things on the previous page are a big part of the solution, and they're all things that have been proven to help your brain work in your favor. For example, studies show that putting words *within* the pictures they describe (as opposed to somewhere else in the page, like a caption or in the body text) causes your brain to try to makes sense of how the words and picture relate, and this causes more neurons to fire. More neurons firing = more chances for your brain to *get* that this is something worth paying attention to, and possibly recording.

A conversational style helps because people tend to pay more attention when they perceive that they're in a conversation, since they're expected to follow along and hold up their end. The amazing thing is, your brain doesn't necessarily *care* that the "conversation" is between you and a book! On the other hand, if the writing style is formal and dry, your brain perceives it the same way you experience being lectured to while sitting in a roomful of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning…

Here's what WE did

We used ***pictures***, because your brain is tuned for visuals, not text. As far as your brain's concerned, a picture really *is* worth a thousand words. And when text and pictures work together, we embedded the text *in* the pictures because your brain works more effectively when the text is *within* the thing it refers to, as opposed to in a caption or buried in the body text somewhere.

We used ***redundancy***, saying the same thing in *different* ways and with different media types, and *multiple senses*, to increase the chance that the content gets coded into more than one area of your brain.

We used concepts and pictures in ***unexpected*** ways because your brain is tuned for novelty, and we used pictures and ideas with at least *some* ***emotional*** *content*, because your brain is tuned to pay attention to the biochemistry of emotions. That which causes you to *feel* something is more likely to be remembered, even if that feeling is nothing more than a little ***humor***, ***surprise***, or ***interest.***

We used a personalized, ***conversational style***, because your brain is tuned to pay more attention when it believes you're in a conversation than if it thinks you're passively listening to a presentation. Your brain does this even when you're *reading*.

We included ***activities***, because your brain is tuned to learn and remember more when you ***do*** things than when you *read* about things. And we made the exercises challenging-yet-doable, because that's what most people prefer.

We used ***multiple learning styles***, because *you* might prefer step-by-step procedures, while someone else wants to understand the big picture first, and someone else just wants to see an example. But regardless of your own learning preference, *everyone* benefits from seeing the same content represented in multiple ways.

We include content for ***both sides of your brain***, because the more of your brain you engage, the more likely you are to learn and remember, and the longer you can stay focused. Since working one side of the brain often means giving the other side a chance to rest, you can be more productive at learning for a longer period of time.

And we included ***stories*** and exercises that present ***more than one point of view,*** because your brain is tuned to learn more deeply when it's forced to make evaluations and judgments.

We included ***challenges***, with exercises, and by asking ***questions*** that don't always have a straight answer, because your brain is tuned to learn and remember when it has to *work* at something. Think about it—you can't get your *body* in shape just by *watching* people at the gym. But we did our best to make sure that when you're working hard, it's on the *right* things. That ***you're not spending one extra dendrite*** processing a hard-to-understand example, or parsing difficult, jargon-laden, or overly terse text.

We used ***people***. In stories, examples, pictures, and the like, because, well, *you're* a person. And your brain pays more attention to *people* than it does to *things*.

Here's what YOU can do to bend your brain into submission

So, we did our part. The rest is up to you. These tips are a starting point; listen to your brain and figure out what works for you and what doesn't. Try new things.

Cut this out and stick it on your refrigerator.

1. **Slow down. The more you understand, the less you have to memorize.**
   Don't just *read*. Stop and think. When the book asks you a question, don't just skip to the answer. Imagine that someone really *is* asking the question. The more deeply you force your brain to think, the better chance you have of learning and remembering.

2. **Do the exercises. Write your own notes.**
   We put them in, but if we did them for you, that would be like having someone else do your workouts for you. And don't just *look* at the exercises. **Use a pencil.** There's plenty of evidence that physical activity *while* learning can increase the learning.

3. **Read "There Are No Dumb Questions."**
   That means all of them. They're not optional sidebars, ***they're part of the core content!*** Don't skip them.

4. **Make this the last thing you read before bed. Or at least the last challenging thing.**
   Part of the learning (especially the transfer to long-term memory) happens *after* you put the book down. Your brain needs time on its own, to do more processing. If you put in something new during that processing time, some of what you just learned will be lost.

5. **Talk about it. Out loud.**
   Speaking activates a different part of the brain. If you're trying to understand something, or increase your chance of remembering it later, say it out loud. Better still, try to explain it out loud to someone else. You'll learn more quickly, and you might uncover ideas you hadn't known were there when you were reading about it.

6. **Drink water. Lots of it.**
   Your brain works best in a nice bath of fluid. Dehydration (which can happen before you ever feel thirsty) decreases cognitive function.

7. **Listen to your brain.**
   Pay attention to whether your brain is getting overloaded. If you find yourself starting to skim the surface or forget what you just read, it's time for a break. Once you go past a certain point, you won't learn faster by trying to shove more in, and you might even hurt the process.

8. **Feel something.**
   Your brain needs to know that this *matters*. Get involved with the stories. Make up your own captions for the photos. Groaning over a bad joke is *still* better than feeling nothing at all.

9. **Write a lot of code!**
   There's only one way to learn to develop Android apps: **write a lot of code**. And that's what you're going to do throughout this book. Coding is a skill, and the only way to get good at it is to practice. We're going to give you a lot of practice: every chapter has exercises that pose a problem for you to solve. Don't just skip over them—a lot of the learning happens when you solve the exercises. We included a solution to each exercise—don't be afraid to **peek at the solution** if you get stuck! (It's easy to get snagged on something small.) But try to solve the problem before you look at the solution. And definitely get it working before you move on to the next part of the book.

Read me

This is a learning experience, not a reference book. We deliberately stripped out everything that might get in the way of learning whatever it is we're working on at that point in the book. And the first time through, you need to begin at the beginning, because the book makes assumptions about what you've already seen and learned.

We assume you're new to Android, but not to Java.

We're going to be building Android apps using a combination of Java and XML. We assume that you're familiar with the Java prorgamming language. If you've never done any Java programming *at all,* then you might want to read *Head First Java* before you start on this one.

We start off by building an app in the very first chapter.

Believe it or not, even if you've never developed for Android before, you can jump right in and start building apps. You'll also learn your way around Android Studio, the official IDE for Android development.

The examples are designed for learning.

As you work through the book, you'll build a number of different apps. Some of these are very small so you can focus on a specific part of Android. Other apps are larger so you can see how different components fit together. We won't complete every part of every app, but feel free to experiment and finish them yourself. It's all part of the learning experience. The source code for all the apps is here: *https://tinyurl.com/HeadFirstAndroid*.

The activities are NOT optional.

The exercises and activities are not add-ons; they're part of the core content of the book. Some of them are to help with memory, some are for understanding, and some will help you apply what you've learned. ***Don't skip the exercises.***

The redundancy is intentional and important.

One distinct difference in a Head First book is that we want you to *really* get it. And we want you to finish the book remembering what you've learned. Most reference books don't have retention and recall as a goal, but this book is about *learning*, so you'll see some of the same concepts come up more than once.

The Brain Power exercises don't have answers.

For some of them, there is no right answer, and for others, part of the learning experience of the Brain Power activities is for you to decide if and when your answers are right. In some of the Brain Power exercises, you will find hints to point you in the right direction.

The technical review team

Technical reviewers:

Andy Parker is currently working as a development manager, but has been a research physicist, teacher, designer, reviewer, and team leader at various points in his career. Through all of his roles, he has never lost his passion for creating top quality, well-designed, and well-engineered software. Nowadays, he spends most of his time managing great Agile teams and passing on his wide range of experience to the next generation of developers.

Jacqui Cope started coding to avoid school netball practice. Since then she has gathered 30 years' experience working with a variety of financial software systems, from coding in COBOL to test management. Recently she has gained her MSc in Computer Security and has moved into software Quality Assurance in the higher education sector.

In her spare time, Jacqui likes to cook, walk in the countryside, and watch *Doctor Who* from behind the sofa.

Acknowledgments

Our editor:

Heartfelt thanks to our wonderful editor **Dawn Schanafelt** for picking up the reins on the second edition. She has truly been amazing, and a delight to work with. She made us feel valued and supported every step of the way, and gave us invaluable feedback and insight exactly when we needed it. We've appreciated all the many times she told us our sentences had all the right words, but not necessarily in the right order.

Thanks also to **Bert Bates** for teaching us to throw away the old rulebook and for letting us into his brain.

The O'Reilly team:

A big thank you goes to **Mike Hendrickson** for having confidence in us and asking us to write the first edition of the book; **Heather Scherer** for her behind-the-scenes organization skills and herding; the **early release team** for making early versions of the book available for download; and the **design team** for all their extra help. Finally, thanks go to the **production team** for expertly steering the book through the production process and for working so hard behind the scenes.

Family, friends, and colleagues:

Writing a Head First book is a rollercoaster of a ride, even when it's a second edition, and this one's been no exception. We've truly valued the kindness and support of our family and friends along the way. Special thanks go to **Ian**, **Steve**, **Colin**, **Angela**, **Paul B**, **Chris**, **Michael**, **Mum**, **Dad**, **Carl**, **Rob**, and **Lorraine**.

The without-whom list:

Our technical review team did a great job of keeping us straight, and making sure that what we covered was spot on. We're also grateful to **Ingo Krotzky** for his valuable feedback on the early release of this book, and all the people who gave us feedback on the first edition. We think the book's much, much better as a result.

Finally, our thanks to **Kathy Sierra** and **Bert Bates** for creating this extraordinary series of books.

O'Reilly Safari®

Safari (formerly Safari Books Online) is a membership-based training and reference platform for enterprise, government, educators, and individuals.

Members have access to thousands of books, training videos, Learning Paths, interactive tutorials, and curated playlists from over 250 publishers, including O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, and Course Technology, among others.

For more information, please visit *http://oreilly.com/safari*.

1 getting started

Diving In

Android has taken the world by storm.

Everybody wants a smartphone or tablet, and Android devices are hugely popular. In this book, we'll teach you how to **develop your own apps**, and we'll start by getting you to build a basic app and run it on an Android Virtual Device. Along the way, you'll meet some of the basic components of all Android apps, such as **activities** and **layouts**. **All you need is a little Java know-how...**

Welcome to Androidville

Android is the world's most popular mobile platform. At the last count, there were over *two billion* active Android devices worldwide, and that number is growing rapidly.

Android is a comprehensive open source platform based on Linux and championed by Google. It's a powerful development framework that includes everything you need to build great apps using a mix of Java and XML. What's more, it enables you to deploy those apps to a wide variety of devices—phones, tablets, and more.

So what makes up a typical Android app?

We're going to build our Android apps using a mixture of Java and XML. We'll explain things along the way, but you'll need to have a fair understanding of Java to get the most out of this book.

Layouts define what each screen looks like

A typical Android app is composed of one or more screens. You define what each screen looks like using a **layout** to define its appearance. Layouts are usually defined in XML, and can include GUI components such as buttons, text fields, and labels.

Activities define what the app does

Layouts only define the *appearance* of the app. You define what the app *does* using one or more **activities**. An activity is a special Java class that decides which layout to use and tells the app how to respond to the user. As an example, if a layout includes a button, you need to write Java code in the activity to define what the button should do when you press it.

Sometimes extra resources are needed too

In addition to activities and layouts, Android apps often need extra resources such as image files and application data. You can add any extra files you need to the app.

Android apps are really just a bunch of files in particular directories. When you build your app, all of these files get bundled together, giving you an app you can run on your device.

The Android platform dissected

The Android platform is made up of a number of different components. It includes core applications such as Contacts, a set of APIs to help you control what your app looks like and how it behaves, and a whole load of supporting files and libraries. Here's a quick look at how they all fit together:

Don't worry if this seems like a lot to take in.

We're just giving you an overview of what's included in the Android platform. We'll explain the different components in more detail as and when we need to.

Android comes with a set of core applications such as Contacts, Phone, Calendar, and a browser.

When you build apps, you have access to the same APIs used by the core applications. You use these APIs to control what your app looks like and how it behaves.

Underneath the application framework lies a set of C and C++ libraries. These libraries get exposed to you through the framework APIs.

Underneath everything else lies the Linux kernel. Android relies on the kernel for drivers, and also core services such as security and memory management.

The Android runtime comes with a set of core libraries that implement most of the Java programming language. Each Android app runs in its own process.

The great news is that all of the powerful Android libraries are exposed through the APIs in the application framework, and it's these APIs that you use to create great Android apps. All you need to begin is some Java knowledge and a great idea for an app.

Here's what we're going to do

So let's dive in and create a basic Android app. There are just a few things we need to do:

1. **Set up a development environment.**
   We need to install Android Studio, which includes all the tools you need to develop Android apps.

2. **Build a basic app.**
   We'll build a simple app using Android Studio that will display some sample text on the screen.

3. **Run the app in the Android emulator.**
   We'll use the built-in emulator to see the app up and running.

4. **Change the app.**
   Finally, we'll make a few tweaks to the app we created in step 2, and run it again.

there are no Dumb Questions

Q: Are all Android apps developed in Java?

A: You can develop Android apps in other languages, too. Most developers use Java, so that's what we're covering in this book.

Q: How much Java do I need to know for Android app development?

A: You really need experience with Java SE (Standard Edition). If you're feeling rusty, we suggest getting a copy of *Head First Java* by Kathy Sierra and Bert Bates.

Q: Do I need to know about Swing and AWT?

A: Android doesn't use Swing or AWT, so don't worry if you don't have Java desktop GUI experience.

Your development environment

Java is the most popular language used to develop Android applications. Android devices don't run *.class* and *.jar* files. Instead, to improve speed and battery performance, Android devices use their own optimized formats for compiled code. That means that you can't use an ordinary Java development environment—you also need special tools to convert your compiled code into an Android format, to deploy them to an Android device, and to let you debug the app once it's running.

All of these come as part of the **Android SDK**. Let's take a look at what's included.

The Android SDK

The Android Software Development Kit contains the libraries and tools you need to develop Android apps. Here are some of the main points:

SDK Platform
There's one of these for each version of Android.

SDK Tools
Tools for debugging and testing, plus other useful utilities. The SDK also features a set of platform dependent tools.

Sample apps
If you want practical code examples to help you understand how to use some of the APIs, the sample apps might help you.

Documentation
So you can access the latest API documentation offline.

Android support
Extra APIs that aren't available in the standard platform.

Google Play Billing
Allows you to integrate billing services in your app.

Android Studio is a special version of IntelliJ IDEA

IntelliJ IDEA is one of the most popular IDEs for Java development. Android Studio is a version of IDEA that includes a version of the Android SDK and extra GUI tools to help you with your app development.

In addition to providing you with an editor and access to the tools and libraries in the Android SDK, Android Studio gives you templates you can use to help you create new apps and classes, and it makes it easy to do things such as package your apps and run them.

Install Android Studio

Before we go any further, you need to install Android Studio on your machine. We're not including the installation instructions in this book as they can get out of date pretty quickly, but you'll be fine if you follow the online instructions.

We're using Android Studio version 2.3. You'll need to use this version or above to get the most out of this book.

First, check the Android Studio system requirements here:

http://developer.android.com/sdk/index.html#Requirements

Google sometimes changes their URLs. If these URLs don't work, search for Android Studio and you should find them.

Then follow the Android Studio installation instructions here:

https://developer.android.com/sdk/installing/index.html?pkg=studio

Once you've installed Android Studio, open it and follow the instructions to add the latest SDK tools and Support Libraries.

When you're done, you should see the Android Studio welcome screen. You're now ready to build your first Android app.

This is the Android Studio welcome screen. It includes a set of options for things you can do.

there are no Dumb Questions

Q: You say we're going to use Android Studio to build the Android apps. Do I have to?

A: Strictly speaking, you don't *have* to use Android Studio to build Android apps. All you need is a tool that will let you write and compile Java code, plus a few other tools to convert the compiled code into a form that Android devices can run.

Android Studio is the official Android IDE, and the Android team recommends using it. But quite a lot of people use IntelliJ IDEA instead.

Q: Can I write Android apps without using an IDE?

A: It's possible, but it's more work. Most Android apps are now created using a build tool called *Gradle*. Gradle projects can be created and built using a text editor and a command line.

Q: A build tool? So is gradle like ANT?

A: It's similar, but Gradle is much more powerful than ANT. Gradle can compile and deploy code, just like ANT, but it also uses Maven to download any third-party libraries your code needs. Gradle also uses Groovy as a scripting language, which means you can easily create quite complex builds with Gradle.

Q: Most apps are built using Gradle? I thought you said most developers use Android Studio.

A: Android Studio provides a graphical interface to Gradle, and also to other tools for creating layouts, reading logs, and debugging.

You can find out more about Gradle in Appendix II.

Build a basic app

Now that you've set up your development environment, you're ready to create your first Android app. Here's what the app will look like:

This is the name of the application.

This is a very simple app, but that's all you need for your very first Android app.

There'll be a small piece of sample text right here that Android Studio will put in for us.

You've completed this step now, so we've checked it off.

- [x] Set up environment
- [ ] **Build app**
- [ ] Run app
- [ ] Change app

How to build the app

Whenever you create a new app, you need to create a new project for it. Make sure you have Android Studio open, and follow along with us.

1. Create a new project

The Android Studio welcome screen gives you a number of options. We want to create a new project, so click on the option for "Start a new Android Studio project."

Click on this option to start a new Android Studio project.

Any projects you create will appear here. This is our first project, so this area is currently empty.

How to build the app (continued)

2. Configure the project

You now need to configure the app by telling Android Studio what you want to call it, what company domain to use, and where you would like to store the files.

Android Studio uses the company domain and application name to form the name of the package that will be used for your app. As an example, if you give your app a name of "My First App" and use a company domain of "hfad.com", Android Studio will derive a package name of `com.hfad.myfirstapp`. The package name is really important in Android, as it's used by Android devices to uniquely identify your app.

Enter an application name of "My First App", enter a company domain of "hfad.com", uncheck the option to include C++ support, and accept the default project location. Then click on the Next button.

The package name must stay the same for the lifetime of your app.

It's a unique identifier for your app and used to manage multiple versions of the same app.

Some versions of Android Studio may have an extra option asking if you want to include Kotlin support. Uncheck this option if it's there.

The wizard forms the package name by combining the application name and the company domain.

How to build the app (continued)

Set up environment
Build app
Run app
Change app

3. Specify the minimum SDK

You now need to indicate the minimum SDK of Android your app will use. API levels increase with every new version of Android. Unless you only want your app to run on the very newest devices, you'll probably want to specify one of the older APIs.

Here, we're choosing a minimum SDK of API level 19, which means it will be able to run on most devices. Also, we're only going to create a version of our app to run on phones and tablets, so we'll leave the other options unchecked.

There's more about the different API levels on the next page.

When you've done this, click on the Next button.

The minimum required SDK is the lowest version your app will support. Your app will run on devices with this level API or higher. It won't run on devices with a lower API.

Android Versions Up Close

You've probably heard a lot of things about Android that sound tasty, like Jelly Bean, KitKat, Lollipop, and Nougat. So what's with all the confectionary?

Android versions have a version number and a codename. The version number gives the precise version of Android (e.g., 7.0), while the codename is a more generic "friendly" name that may cover several versions of Android (e.g., Nougat). The API level refers to the version of the APIs used by applications. As an example, the equivalent API level for Android version 7.1.1 is 25.

Version	Codename	API level
1.0		1
1.1		2
1.5	Cupcake	3
1.6	Donut	4
2.0–2.1	Eclair	5–7
2.2.x	Froyo	8
2.3–2.3.7	Gingerbread	9–10
3.0 - 3.2	Honeycomb	11–13
4.0–4.0.4	Ice Cream Sandwich	14–15
4.1 - 4.3	Jelly Bean	16–18
4.4	KitKat	19–20
5.0–5.1	Lollipop	21–22
6.0	Marshmallow	23
7.0	Nougat	24
7.1–7.1.2	Nougat	25

Hardly anyone uses these versions anymore.

Most devices use one of these APIs.

When you develop Android apps, you really need to consider which versions of Android you want your app to be compatible with. If you specify that your app is only compatible with the very latest version of the SDK, you might find that it can't be run on many devices. You can find out the percentage of devices running particular versions here: *https://developer.android.com/about/dashboards/index.html.*

Activities and layouts from 50,000 feet

Set up environment
Build app
Run app
Change app

The next thing you'll be prompted to do is add an activity to your project. Every Android app is a collection of screens, and each screen is composed of an activity and a layout.

An **activity** is **a single, defined thing that your user can do**. You might have an activity to compose an email, take a photo, or find a contact. Activities are usually associated with one screen, and they're written in Java.

A **layout** describes **the appearance of the screen.** Layouts are written as XML files and they tell Android how the different screen elements are arranged.

Let's look in more detail at how activities and layouts work together to create a user interface:

Layouts define how the user interface is presented.

Activities define actions.

1. The device launches your app and creates an activity object.
2. The activity object specifies a layout.
3. The activity tells Android to display the layout onscreen.
4. The user interacts with the layout that's displayed on the device.
5. The activity responds to these interactions by running application code.
6. The activity updates the display...
7. ...which the user sees on the device.

Now that you know a bit more about what activities and layouts are, let's go through the last couple of steps in the Create New Project wizard and get it to create an activity and layout.

How to build the app (continued)

4. Add an activity

The next screen lets you choose among a series of templates you can use to create an activity and layout. We're going to create an app with an empty activity and layout, so choose the Empty Activity option and click the Next button.

There are other types of activity you can choose from, but for this exercise make sure you select the Empty Activity option.

How to build the app (continued)

Set up environment
Build app
Run app
Change app

5. Customize the activity

You will now be asked what you want to call the screen's activity and layout. Enter an activity name of "MainActivity", make sure the option to generate a layout file is checked, enter a layout name of "activity_main", and then uncheck the Backwards Compatibility (AppCompat) option. The activity is a Java class, and the layout is an XML file, so the names we've given here will create a Java class file called *MainActivity.java* and an XML file called *activity_main.xml*.

When you click on the Finish button, Android Studio will build your app.

Name the activity "MainActivity" and the layout "activity_main". Also make sure the option to generate the layout is checked.

Uncheck the Backwards Compatibility (AppCompat) option. You'll find out more about this setting later in the book.

You've just created your first Android app

So what just happened?

- **The Create New Project wizard created a project for your app, configured to your specifications.**
  You defined which versions of Android the app should be compatible with, and the wizard created all of the files and folders needed for a basic valid app.

- **The wizard created an activity and layout with template code.**
  The template code includes layout XML and activity Java code, with sample "Hello World!" text in the layout.

When you finish creating your project by going through the wizard, Android Studio automatically displays the project for you.

Here's what our project looks like (don't worry if it looks complicated—we'll break it down over the next few pages):

This is the project in Android Studio.

Set up environment
Build app
Run app
Change app

Android Studio creates a complete folder structure for you

An Android app is really just a bunch of valid files in a particular folder structure, and Android Studio sets all of this up for you when you create a new app. The easiest way of looking at this folder structure is with the explorer in the leftmost column of Android Studio.

The explorer contains all of the projects that you currently have open. To expand or collapse folders, just click on the arrows to the left of the folder icons.

Click on the arrow here and choose the Project option to see the files and folders that make up your project.

This is the name of the project.

These files and folders are all included in your project.

Click on these arrows to expand or collapse the folders.

The folder structure includes different types of files

If you browse through the folder structure, you'll see that the wizard has created various types of files and folders for you:

- **Java and XML source files**
  These are the activity and layout files for your app.
- **Android-generated Java files**
  There are some extra Java files you don't need to touch that Android Studio generates for you automatically.
- **Resource files**
  These include default image files for icons, styles your app might use, and any common String values your app might want to look up.
- **Android libraries**
  In the wizard, you specified the minimum SDK version you want your app to be compatible with. Android Studio makes sure your app includes the relevant Android libraries for that version.
- **Configuration files**
  The configuration files tell Android what's actually in the app and how it should run.

Let's take a closer look at some of the key files and folders in Androidville.

Useful files in your project

Android Studio projects use the Gradle build system to compile and deploy apps. Gradle projects have a standard structure. Here are some of the key files and folders you'll be working with:

Edit code with the Android Studio editors

Set up environment
Build app
Run app
Change app

You view and edit files using the Android Studio editors. Double-click on the file you want to work with, and the file's contents will appear in the middle of the Android Studio window.

The code editor

Most files get displayed in the code editor, which is just like a text editor, but with extra features such as color coding and code checking.

Double-click on the file in the explorer and the file contents appear in the editor panel.

The design editor

If you're editing a layout, you have an extra option. Rather than edit the XML (such as that shown on the next page), you can use the design editor, which allows you to drag GUI components onto your layout, and arrange them how you want. The code editor and design editor give different views of the same file, so you can switch back and forth between the two.

You dictate which editor you're using with these tabs.

You can edit layouts using the visual editor by dragging and dropping components.

WHAT'S MY PURPOSE?

Here's the code from an example layout file (**not the one Android Studio generated for us**). We know you've not seen layout code before, but just see if you can match each of the descriptions at the bottom of the page to the correct lines of code. We've done one to get you started.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.myfirstapp.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

Add padding to the screen margins.

Display the String "Hello World!"

Include a `<TextView>` GUI component for displaying text.

Make the layout the same width and height as the screen size on the device.

Make the GUI component just large enough for its content.

WHAT'S MY PURPOSE? SOLUTION

Here's the code from an example layout file (**not the one Android Studio generated for us**). We know you've not seen layout code before, but just see if you can match each of the descriptions at the bottom of the page to the correct lines of code. We've done one to get you started.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.myfirstapp.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

Add padding to the screen margins.

Display the String "Hello World!"

Include a `<TextView>` GUI component for displaying text.

Make the layout the same width and height as the screen size on the device.

Make the GUI component just large enough for its content.

WHAT'S MY PURPOSE?

Now let's see if you can do the same thing for some activity code. **This is example code, and not necessarily the code that Android Studio will have generated for you**. Match the descriptions below to the correct lines of code.

MainActivity.java

```
package com.hfad.myfirstapp;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

This is the package name.

Implement the `onCreate()` method from the `Activity` class. This method is called when the activity is first created.

These are Android classes used in `MainActivity`.

`MainActivity` extends the Android class `android.app.Activity`.

Specify which layout to use.

WHAT'S MY PURPOSE? SOLUTION

Now let's see if you can do the same thing for some activity code. **This is example code, and not necessarily the code that Android Studio will have generated for you**. Match the descriptions below to the correct lines of code.

MainActivity.java

```
package com.hfad.myfirstapp;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
)
```

This is the package name.

These are Android classes used in `MainActivity`.

Specify which layout to use.

Implement the `onCreate()` method from the `Activity` class. This method is called when the activity is first created.

`MainActivity` extends the Android class `android.app.Activity`.

Run the app in the Android emulator

So far you've seen what your Android app looks like in Android Studio and got a feel for how it hangs together. But what you *really* want to do is see it running, right?

You have a couple of options when it comes to running your apps. The first option is to run them on a physical device. But what if you don't have one with you, or you want to see how your app looks on a type of device you don't have?

In that case, you can use the **Android emulator** that's built into the Android SDK. The emulator enables you to set up one or more **Android virtual devices** (AVDs) and then run your app in the emulator *as though it's running on a physical device*.

The Android emulator allows you to run your app on an Android virtual device (AVD), which behaves just like a physical Android device. You can set up numerous AVDs, each emulating a different type of device.

So what does the emulator look like?

Here's an AVD running in the Android emulator. It looks just like a phone running on your computer.

The emulator recreates the exact hardware environment of an Android device: from its CPU and memory through to the sound chips and the video display. The emulator is built on an existing emulator called QEMU (pronounced "queue em you"), which is similar to other virtual machine applications you may have used, like VirtualBox or VMWare.

The exact appearance and behavior of the AVD depends on how you've set up the AVD in the first place. The AVD here is set up to mimic a Nexus 5X, so it will look and behave just like a Nexus 5X on your computer.

Let's set up an AVD so that you can see your app running in the emulator.

Once you've set up an AVD, you'll be able to see your app running on it. Android Studio launches the emulator for you.

Create an Android Virtual Device

Set up environment
Build app
Run app
Change app

There are a few steps you need to go through in order to set up an AVD within Android Studio. We'll set up a Nexus 5X AVD running API level 25 so that you can see how your app looks and behaves running on this type of device. The steps are pretty much identical no matter what type of virtual device you want to set up.

Open the Android Virtual Device Manager

The AVD Manager allows you to set up new AVDs, and view and edit ones you've already created. Open it by selecting Android on the Tools menu and choosing AVD Manager.

If you have no AVDs set up already, you'll be presented with a screen prompting you to create one. Click on the "Create Virtual Device" button.

Click on this button to create an AVD.

Select the hardware

On the next screen, you'll be prompted to choose a device definition. This is the type of device your AVD will emulate. You can choose a variety of phone, tablet, wear, or TV devices.

We're going to see what our app looks like running on a Nexus 5X phone. Choose Phone from the Category menu and Nexus 5X from the list. Then click the Next button.

Creating an AVD (continued)

Select a system image

Next, you need to select a system image. The system image gives you an installed version of the Android operating system. You can choose the version of Android you want to be on your AVD.

You need to choose a system image for an API level that's compatible with the app you're building. As an example, if you want your app to work on a minimum of API level 19, choose a system image for *at least* API level 19. We want our AVD to run API level 25, so choose the system image with a release name of Nougat and a target of Android 7.1.1 (API level 25). Then click on the Next button.

If you don't have this system image installed, you'll be given the option to download it.

We'll continue setting up the AVD on the next page.

Creating an AVD (continued)

- Set up environment
- Build app
- **Run app**
- Change app

Verify the AVD configuration

On the next screen, you'll be asked to verify the AVD configuration. This screen summarizes the options you chose over the last few screens, and gives you the option of changing them. Accept the options, and click on the Finish button.

The AVD Manager will create the AVD for you, and when it's done, display it in the AVD Manager list of devices. You may now close the AVD Manager.

Set up environment
Build app
Run app
Change app

Run the app in the emulator

Now that you've set up your AVD, let's run the app on it. To do this, choose the "Run 'app'" command from the Run menu. When you're asked to choose a device, select the Nexus 5X AVD you just created. Then click OK.

The AVD can take a few minutes to appear, so while we wait, let's take a look at what happens when you choose Run.

This is the AVD we just created.

Compile, package, deploy, and run

The Run command doesn't just run your app. It also handles all the preliminary tasks that are needed for the app to run:

An APK file is an Android application package. It's basically a JAR or ZIP file for Android applications.

1. **The Java source files get compiled to bytecode.**
2. **An Android application package, or APK file, gets created.**
   The APK file includes the compiled Java files, along with any libraries and resources needed by your app.
3. **Assuming there's not one already running, the emulator gets launched and then runs the AVD.**
4. **Once the emulator has been launched and the AVD is active, the APK file is uploaded to the AVD and installed.**
5. **The AVD starts the main activity associated with the app.**
   Your app gets displayed on the AVD screen, and it's all ready for you to test out.

You can watch progress in the console

Set up environment
Build app
Run app
Change app

It can sometimes take quite a while for the emulator to launch with your AVD—often *several minutes*. If you like, you can watch what's happening using the Android Studio console. The console gives you a blow-by-blow account of what the build system is doing, and if it encounters any errors, you'll see them highlighted in the text.

We suggest finding something else to do while waiting for the emulator to start. Like quilting, or cooking a small meal.

You can find the console at the bottom of the Android Studio screen (click on the Run option at the bottom of the screen if it doesn't appear automatically):

Here's the output from our console window when we ran our app:

```
03/13 10:45:41: Launching app
$ adb install-multiple -r /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/intermediates/
split-apk/debug/dep/dependencies.apk /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/
intermediates/split-apk/debug/slices/slice_1.apk /Users/dawng/AndroidStudioProjects/MyFirstApp/
app/build/intermediates/split-apk/debug/slices/slice_2.apk /Users/dawng/AndroidStudioProjects/
MyFirstApp/app/build/intermediates/split-apk/debug/slices/slice_0.apk /Users/dawng/
AndroidStudioProjects/MyFirstApp/app/build/intermediates/split-apk/debug/slices/slice_3.apk /Users/
dawng/AndroidStudioProjects/MyFirstApp/app/build/intermediates/split-apk/debug/slices/slice_6.apk /
Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/intermediates/split-apk/debug/slices/slice_4.
apk /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/intermediates/split-apk/debug/slices/
slice_5.apk /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/intermediates/split-apk/debug/
slices/slice_7.apk /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/intermediates/split-apk/
debug/slices/slice_8.apk /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/intermediates/
split-apk/debug/slices/slice_9.apk /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/outputs/
apk/app-debug.apk
Split APKs installed
$ adb shell am startservice com.hfad.myfirstapp/com.android.tools.fd.runtime.InstantRunService
$ adb shell am start -n "com.hfad.myfirstapp/com.hfad.myfirstapp.MainActivity" -a android.intent.
action.MAIN -c android.intent.category.LAUNCHER
Connected to process 2685 on device Nexus_5X_API_25 [emulator-5554]
```

Install the app.

Android Studio has finished launching the AVD we just set up.

The emulator launches our app by starting the main activity for it. This is the activity the wizard created for us.

- Set up environment
- Build app
- **Run app**
- Change app

Test drive

So let's look at what actually happens onscreen when you run your app.

First, the emulator fires up in a separate window. The emulator takes a while to load the AVD, but then you see what looks like an actual Android device.

The emulator launches...

...and here's the AVD home screen. It looks and behaves just like a real Nexus 5X device.

This is the name of the app.

Wait a bit longer, and you'll see the app you just created. The application name appears at the top of the screen, and the default sample text "Hello World!" is displayed in the middle of the screen.

Hello World!

Android Studio created the sample text "Hello World!" without us telling it to.

The wizard created sample text for us.

Here's the app running on the AVD.

What just happened?

Set up environment
Build app
Run app
Change app

Let's break down what happens when you run the app:

1. **Android Studio launches the emulator, loads the AVD, and installs the app.**
2. **When the app gets launched, an activity object is created from MainActivity.java.**
3. **The activity specifies that it uses the layout activity_main.xml.**
4. **The activity tells Android to display the layout on the screen.** The text "Hello World!" gets displayed.

In this particular instance, we're using a virtual device.

there are no Dumb Questions

Q: You mentioned that when you create an APK file, the Java source code gets compiled into bytecode and added to the APK. Presumably you mean it gets compiled into Java bytecode, right?

A: It does, but that's not the end of the story. Things work a little differently on Android.

The big difference with Android is that your code doesn't actually run inside an ordinary Java VM. It runs on the Android runtime (ART) instead, and on older devices it runs in a predecessor to ART called Dalvik. This means that you write your Java source code and compile it into *.class* files using the Java compiler, and then the *.class* files get stitched into one or more files in DEX format, which is smaller, more efficient bytecode. ART then runs the DEX code. You can find more details about this in Appendix III.

Q: That sounds complicated. Why not just use the normal Java VM?

A: ART can convert the DEX bytecode into native code that can run directly on the CPU of the Android device. This makes the app run a lot faster, and use a lot less battery power.

Q: Is a Java virtual machine really that much overhead?

A: Yes. Because on Android, each app runs inside its own process. If it used ordinary JVMs, it would need a lot more memory.

Q: Do I need to create a new AVD every time I create a new app?

A: No, once you've created the AVD you can use it for any of your apps. You may find it useful to create multiple AVDs in order to test your apps in different situations. As an example, in addition to a phone AVD you might want to create a tablet AVD so you can see how your app looks and behaves on larger devices.

Set up environment
Build app
Run app
Change app

Refine the app

Over the past several pages, you've built a basic Android app and seen it running in the emulator. Next, we're going to refine the app.

At the moment, the app displays the sample text "Hello World!" that the wizard put in as a placeholder. You're going to change that text to say something else instead. So what do we need to change in order to achieve that? To answer that, let's take a step back and look at how the app is currently built.

The app has one activity and one layout

When we built the app, we told Android Studio how to configure it, and the wizard did the rest. The wizard created an activity for us, and also a default layout.

The activity controls what the app does

Android Studio created an activity for us called *MainActivity.java*. The activity specifies what the app **does** and how it should respond to the user.

The layout controls the app's appearance

MainActivity.java specifies that it uses the layout Android Studio created for us called *activity_main.xml*. The layout specifies what the app **looks like**.

We want to change the appearance of the app by changing the text that's displayed. This means that we need to deal with the Android component that controls what the app looks like, so we need to take a closer look at the *layout*.

What's in the layout?

We want to change the sample "Hello World!" text that Android Studio created for us, so let's start with the layout file *activity_main.xml*. If it isn't already open in an editor, open it now by finding the file in the *app/src/main/res/layout* folder in the explorer and double-clicking on it.

If you can't see the folder structure in the explorer, try switching to Project view.

Click on this arrow to change how the files and folders are shown.

The design editor

As you learned earlier, there are two ways of viewing and editing layout files in Android Studio: through the **design editor** and through the **code editor**.

When you choose the design option, you can see that the sample text "Hello World!" appears in the layout as you might expect. But what's in the underlying XML?

Let's see by switching to the code editor.

The design editor

Here's the sample text.

You can see the design editor by choosing "Design" here.

The code editor

When you choose the code editor option, the content of *activity_main.xml* is displayed. Let's take a closer look at it.

The code editor

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android=
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.hfad.myfirstapp.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Design Text

To see the code editor, click on "Text" in the bottom tab.

Set up environment
Build app
Run app
Change app

activity_main.xml has two elements

Below is the code from *activity_main.xml* that Android Studio generated for us. We've left out some of the details you don't need to think about just yet; we'll cover them in more detail through the rest of the book.

Here's our code:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    ... >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        ... />
</android.support.constraint.ConstraintLayout>
```

This element determines how components should be displayed, in this case the "Hello World!" text.

Android Studio gave us more XML here, but you don't need to think about that yet.

This is the <TextView> element.

We've left out some of the <TextView> XML too.

As you can see, the code contains two elements.

The first is an `<android.support.constraint.ConstraintLayout>` element. This is a type of layout element that tells Android how to display components on the device screen. There are various types of layout element available for you to use, and you'll find out more about these later in the book.

The most important element for now is the second element, the `<TextView>`. This element is used to display text to the user, in our case the sample text "Hello World!"

The key part of the code within the `<TextView>` element is the line starting with `android:text`. This is a text property describing the text that should be displayed:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    ... />
```

The <TextView> element describes the text in the layout.

This is the text that's being displayed.

Don't worry if your layout code looks different from ours.

Android Studio may give you slightly different XML depending on which version you're using. You don't need to worry about this, because from the next chapter onward you'll learn how to roll your own layout code, and replace a lot of what Android Studio gives you.

Let's change the text to something else.

Update the text displayed in the layout

- Set up environment
- Build app
- Run app
- **Change app**

The key part of the `<TextView>` element is this line:

```
android:text="Hello World!" />
```

`android:text` means that this is the `text` property of the `<TextView>` element, so it specifies what text should be displayed in the layout. In this case, the text that's being displayed is "Hello World!"

Display the text...

...“Hello World!”

```
android:text="Hello World!" />
```

To update the text that's displayed in the layout, simply change the value of the `text` property from `"Hello World!"` to `"Sup doge"`. The new code for the `<TextView>` should look like this:

We've left out some of the code, as all we're doing for now is changing the text that's displayed.

```
...
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World! Sup doge"
    ... />
...
```

Change the text here from "Hello World!" to "Sup doge".

Once you've updated the file, go to the File menu and choose the Save All option to save your change.

there are no Dumb Questions

Q: My layout code looks different from yours. Is that OK?

A: Yes, that's fine. Android Studio may generate slightly different code if you're using a different version than us, but that doesn't really matter. From now on you'll be learning how to create your own layout code, and you'll replace a lot of what Android Studio gives you.

Q: Am I right in thinking we're hardcoding the text that's displayed?

A: Yes, purely so that you can see how to update text in the layout. There's a better way of displaying text values than hardcoding them in your layouts, but you'll have to wait for the next chapter to learn what it is.

Q: The folders in my project explorer pane look different from yours. Why's that?

A: Android Studio lets you choose alternate views for how to display the folder hierarchy, and it defaults to the "Android" view. We prefer the "Project" view, as it reflects the underlying folder structure. You can change your explorer to the "Project" view by clicking on the arrow at the top of the explorer pane, and selecting the "Project" option.

We're using the Project view.

Click on this arrow to change the explorer view.

Take the app for a test drive

- Set up environment
- Build app
- Run app
- **Change app**

Once you've edited the file, try running your app in the emulator again by choosing the "Run 'app'" command from the Run menu. You should see that your app now says "Sup doge" instead of "Hello World!"

Here's the updated version of our app.

You've now built and updated your first Android app.

Your Android Toolbox

You've got Chapter 1 under your belt and now you've added Android basic concepts to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Versions of Android have a version number, API level, and code name.
- Android Studio is a special version of IntelliJ IDEA that interfaces with the Android Software Development Kit (SDK) and the Gradle build system.
- A typical Android app is composed of activities, layouts, and resource files.
- Layouts describe what your app looks like. They're held in the *app/src/main/res/layout* folder.
- Activities describe what your app does, and how it interacts with the user. The activities you write are held in the *app/src/main/java* folder.
- *AndroidManifest.xml* contains information about the app itself. It lives in the *app/src/main* folder.
- An AVD is an Android Virtual Device. It runs in the Android emulator and mimics a physical Android device.
- An APK is an Android application package. It's like a JAR file for Android apps, and contains your app's bytecode, libraries, and resources. You install an app on a device by installing the APK.
- Android apps run in separate processes using the Android runtime (ART).
- The `<TextView>` element is used for displaying text.

2 building interactive apps

Apps That Do Something

Most apps need to respond to the user in some way.

In this chapter, you'll see how you can make your apps **a bit more interactive**. You'll learn how to get your app to ***do*** something in response to the user, and **how to get your activity and layout talking to each other** like best buddies. Along the way, we'll take you a bit **deeper into how Android actually works** by introducing you to **R**, the hidden gem that glues everything together.

Let's build a Beer Adviser app

In Chapter 1, you saw how to create an app using the Android Studio New Project wizard, and how to change the text displayed in the layout. But when you create an Android app, you're usually going to want the app to *do* something.

In this chapter, we're going to show you how to create an app that the user can interact with: a Beer Adviser app. In the app, users can select the types of beer they enjoy, click a button, and get back a list of tasty beers to try out.

Here's how the app will be structured:

1. **The layout specifies what the app will look like.**
   It includes three GUI components:
   - A drop-down list of values called a spinner, which allows the user to choose which type of beer they want.
   - A button that when pressed will return a selection of beer types.
   - A text field that displays the types of beer.

2. **The file strings.xml includes any String resources needed by the layout—for example, the label of the button specified in the layout and the types of beer.**

3. **The activity specifies how the app should interact with the user.**
   It takes the type of beer the user chooses, and uses this to display a list of beers the user might be interested in. It achieves this with the help of a custom Java class.

4. **The custom Java class contains the application logic for the app.**
   It includes a method that takes a type of beer as a parameter, and returns a list of beers of this type. The activity calls the method, passes it the type of beer, and uses the response.

Here's what we're going to do

So let's get to work. There are a few steps you need to go through to build the Beer Adviser app (we'll tackle these throughout the rest of the chapter):

1. **Create a project.**
   You're creating a brand-new app, so you'll need to create a new project. Just like before, you'll need to create an empty activity with a layout.

   ← We'll show you the details of how to do this on the next page.

2. **Update the layout.**
   Once you have the app set up, you need to amend the layout so that it includes all the GUI components your app needs.

3. **Connect the layout to the activity.**
   The layout only creates the visuals. To add smarts to your app, you need to connect the layout to the Java code in your activity.

4. **Write the application logic.**
   You'll add a Java custom class to the app, and use it to make sure users get the right beer based on their selection.

Create the project

- **Create project**
- Update layout
- Connect activity
- Write logic

Let's begin by creating the new app (the steps are similar to those we used in the previous chapter):

1. Open Android Studio and choose "Start a new Android Studio project" from the welcome screen. This starts the wizard you saw in Chapter 1.
2. When prompted, enter an application name of "Beer Adviser" and a company domain of "hfad.com", making your package name `com.hfad.beeradviser`. Make sure you uncheck the option to include C++ support.
3. We want the app to work on most phones and tablets, so choose a minimum SDK of API 19, and make sure the option for "Phone and Tablet" is selected. This means that any phone or tablet that runs the app must have API 19 installed on it as a minimum. Most Android devices meet this criterion.
4. Choose an empty activity for your default activity. Call the activity "FindBeerActivity" and the accompanying layout "activity_find_beer". Make sure the option to generate the layout is selected and you uncheck the Backwards Compatibility (AppCompat) option.

If your version of Android Studio has an option to include Kotlin support, uncheck this option too.

The wizard will take you through these steps, just like before. Call your application "Beer Adviser," make sure it uses a minimum SDK of API 19, and then tell it to create an empty activity called "FindBeerActivity" and a layout called "activity_find_beer".

Make sure you choose the Empty Activity option.

Make sure you UNCHECK the Backwards Compatibility (AppCompat) option.

We've created a default activity and layout

When you click on the Finish button, Android Studio creates a new project containing an activity called *FindBeerActivity.java* and a layout called *activity_find_beer.xml*.

Let's start by changing the layout file. To do this, switch to the Project view of Android Studio's explorer, go to the *app/src/main/res/layout* folder, and open the file *activity_find_beer.xml*. Then switch to the text version of the code to open the code editor, and replace the code in *activity_find_beer.xml* with the following (we've bolded all the new code):

Click on the Text tab to open the code editor.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.beeradviser.FindBeerActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a text view" />
</LinearLayout>
```

We're replacing the code Android Studio generated for us.

These elements relate to the layout as a whole. They determine the layout width and height, any padding in the layout margins, and whether components should be laid out vertically or horizontally.

This is used to display text.

We've just changed the code Android Studio gave us so that it uses a `<LinearLayout>`. This is used to display GUI components next to each other, either vertically or horizontally. If it's vertically, they're displayed in a single column, and if it's horizontally, they're displayed in a single row. You'll find out more about how this works as we go through the chapter.

Any changes you make to a layout's XML are reflected in Android Studio's design editor, which you can see by clicking on the Design tab. We'll look at this in more detail on the next page.

Click on the Design tab to open the design editor.

A closer look at the design editor

Create project
Update layout
Connect activity
Write logic

The design editor presents you with a more visual way of editing your layout code than editing XML. It features two different views of the layouts design. One shows you how the layout will look on an actual device, and the other shows you a blueprint of its structure:

If Android Studio doesn't show you both views of the layout, click on the "Show Design + Blueprint" icon in the design editor's toolbar.

This view of the design gives you an idea of how your layout will look on an actual device.

The text view in our layout's XML code appears in both views of the design editor.

This is the blueprint view, which focuses more on the layout's structure.

To the left of the design editor, there's a palette that contains components you can drag to your layout. We'll use this next.

This list shows you the different categories of component you can add to your layout. You can click on them to filter the components displayed in the palette.

These are the components You'll find out more about them later in the book.

You can increase the size of the palette by clicking on this area and dragging it downward.

Add a button using the design editor

We're going to add a button to our layout using the design editor. Find the `Button` component in the palette, click on it, and then drag it into the design editor so that it's positioned above the text view. The button appears in the layout's design:

Changes in the design editor are reflected in the XML

Dragging GUI components to the layout like this is a convenient way of updating the layout. If you switch to the code editor, you'll see that adding the button via the design editor has added some lines of code to the file:

```
...
<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is a text view" />
...
```

There's a new <Button> element that describes the new button you've dragged to the layout. We'll look at this in more detail over the next few pages.

The code the design editor adds depends on where you place the button, so don't worry if your code looks different from ours.

activity_find_beer.xml has a new button

The editor added a new `<Button>` element to *activity_find_beer.xml*:

```
<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button" />
```

A button in Androidville is a pushbutton that the user can press to trigger an action. The `<Button>` element includes properties controlling its size and appearance. These properties aren't unique to buttons—other GUI components including text views have them too.

Buttons and text views are subclasses of the same Android View class

There's a very good reason why buttons and text views have properties in common—they both inherit from the same Android **`View`** class. You'll find out more about this later in the book, but for now, here are some of the more common properties.

android:id

This gives the component an identifying name. The `id` property enables you to control what components do via activity code:

```
android:id="@+id/button"
```

android:layout_width, android:layout_height

These properties specify the width and height of the component. `"wrap_content"` means it should be just big enough for the content, and `"match_parent"` means it should be as wide as the layout containing it:

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

android:text

This tells Android what text the component should display. In the case of `<Button>`, it's the text that appears on the button:

```
android:text="Button"
```

Create project
Update layout
Connect activity
Write logic

A closer look at the layout code

Let's take a closer look at the layout code, and break it down so that you can see what it's actually doing (don't worry if your code looks a little different, just follow along with us):

The <LinearLayout> element

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.beeradviser.FindBeerActivity">
```

This is the button.

```
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />
```

This is the text view.

```
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a text view" />
```

```
</LinearLayout>
```

This closes the <LinearLayout> element.

The LinearLayout element

The first element in the layout code is `<LinearLayout>`. The `<LinearLayout>` element tells Android that the different GUI components in the layout should be displayed next to each other in a single row or column.

There are other ways of laying out your GUI components too. You'll find out more about these later in the book.

You specify the orientation using the `android:orientation` attribute. In this example we're using:

```
android:orientation="vertical"
```

so the GUI components are displayed in a single vertical column.

A closer look at the layout code (continued)

Create project
Update layout
Connect activity
Write logic

The `<LinearLayout>` contains two elements: a `<Button>` and a `<TextView>`.

The Button element

The first element is the `<Button>`:

```
...
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />
...
```

As this is the first element inside the `<LinearLayout>`, it appears first in the layout at the top of the screen. It has a `layout_width` of `"match_parent"`, which means that it should be as wide as its parent element, the `<LinearLayout>`. Its `layout_height` has been set to `"wrap_content"`, which means it should be tall enough to display its text.

Using a linear layout means that GUI components are displayed in a single row or column.

The TextView element

The final element inside the `<LinearLayout>` is the `<TextView>`:

```
...
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a text view" />
...
```

As this is the second element and we've set the linear layout's orientation to `"vertical"`, it's displayed underneath the button (the first element). Its `layout_width` and `layout_height` are set to `"wrap_content"` so that it takes up just enough space to contain its text.

The button is displayed at the top as it's the first element in the XML.

The text view is displayed underneath the button as it comes after it in the XML.

Changes to the XML...

You've seen how adding components to the design editor adds them to the layout XML. The opposite applies too—any changes you make to the layout XML are applied to the design.

Try this now. Update your *activity_find_beer.xml* code with the following changes (highlighted in bold):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.beeradviser.FindBeerActivity">

    <Spinner
        android:id="@+id/color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:layout_gravity="center"
        android:layout_margin="16dp" />

    <Button
        android:id="@+id/button find_beer"
        android:layout_width="match_parent wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="16dp"
        android:text="Button" />

    <TextView
        android:id="@+id/textView brands"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="16dp"
        android:text="This is a text view" />
</LinearLayout>
```

A spinner is the Android name for a drop-down list of values. It allows you to choose a single value from a selection.

This element displays a spinner in the layout.

Change the button's ID to "find_beer". We'll use this later.

Change the button's width so it's as wide as its content.

Center the button horizontally and give it a margin.

Change the text view's ID to "brands".

Center the text view and apply a margin.

Do this!

Update the contents of *activity_find_beer.xml* with the changes shown here.

Create project
Update layout
Connect activity
Write logic

...are reflected in the design editor

Once you've changed the layout XML, switch to the design editor. Instead of a layout containing a button with a text view underneath it, you should now see a spinner, button, and text view centered in a single column.

A **spinner** is the Android term for a drop-down list of values. When you press it, it expands to show you the list so that you can pick a single value.

We've shown you how to add GUI components to the layout with the aid of the design editor, and also by adding them through XML. In general, you're more likely to hack the XML for simple layouts to get the results you want without using the design editor. This is because editing the XML directly gives you more direct control over the layout.

A spinner provides a drop-down list of values. It allows you to choose a single value from a set of values.

GUI components such as buttons, spinners, and text views have very similar attributes, as they are all types of View. Behind the scenes, they all inherit from the same Android View class.

- Create project
- **Update layout**
- Connect activity
- Write logic

Let's take the app for a test drive

We still have more work to do on the app, but let's see how it's looking so far. Save the changes you've made by choosing File→Save All, then choose the "Run 'app'" command from the Run menu. When prompted, select the option to launch the emulator.

Wait patiently for the app to load, and eventually it should appear.

Try pressing the spinner. It's not immediately obvious, but when you press it, the spinner presents you with a drop-down list of values—it's just at this point we haven't added any values to it.

Here's what we've done so far

Here's a quick recap of what we've done so far:

1. **We've created a layout that specifies what the app looks like.**
   It includes a spinner, a button, and a text view.

2. **The activity specifies how the app should interact with the user.**
   Android Studio has created an activity for us, but we haven't done anything with it yet.

The next thing we'll do is look at replacing the hardcoded String values for the text view and button text.

there are no Dumb Questions

Q: My layout looks slightly different in the AVD compared with how it looks in the design editor. Why's that?

A: The design editor does its best to show you how the layout will look on a device, but it's not always accurate depending on what version of Android Studio you're using. How the layout looks in the AVD reflects how the layout will look on a physical device.

Create project
Update layout
Connect activity
Write logic

Hardcoding text makes localization hard

So far, we've hardcoded the text we want to appear in our text views and buttons using the `android:text` property:

Display the text...

...“Hello World!”

```
android:text="Hello World!" />
```

While this is fine when you're just learning, hardcoding text isn't the best approach.

Suppose you've created an app that's a big hit on your local Google Play Store. You don't want to limit yourself to just one country or language—you want to make it available internationally and for different languages. But if you've hardcoded all of the text in your layout files, sending your app international will be difficult.

It also makes it much harder to make global changes to the text. Imagine your boss asks you to change the wording in the app because the company's changed its name. If you've hardcoded all of the text, this means that you need to edit a whole host of files in order to change the text.

Put the text in a String resource file

A better approach is to put your text values into a String resource file called *strings.xml*.

Having a String resource file makes it much easier to internationalize your app. Rather than having to change hardcoded text values in a whole host of different activity and layout files, you can simply replace the *strings.xml* file with an internationalized version.

This approach also makes it much easier to make global changes to text across your whole application as you only need to edit one file. If you need to make changes to the text in your app, you only need to edit *strings.xml*.

Put String values in strings.xml rather than hardcoding them. strings.xml is a resource file used to hold name/value pairs of Strings. Layouts and activities can look up String values using their names.

How do you use String resources?

In order to use a String resource in your layout, there are two things you need to do:

1. **Create the String resource by adding it to strings.xml.**
2. **Use the String resource in your layout.**

Let's see how this is done.

Create project
Update layout
Connect activity
Write logic

Create the String resource

We're going to create two String resources, one for the text that appears on the button, and another for the default text that appears in the text view.

To do this, use Android Studio's explorer to find the file *strings.xml* in the *app/src/main/res/values* folder. Then open it by double-clicking on it.

The file should look something like this:

```
<resources>
    <string name="app_name">Beer Adviser</string>
</resources>
```

strings.xml contains one string resource named `"app_name"`, which has a value of `Beer Adviser`. Android Studio created this String resource for us automatically when we created the project.

This indicates that this is a String resource.

```
<string name="app_name">Beer Adviser</string>
```

This String resource has a name of "app_name", and a value of "Beer Adviser".

We're first going to add a new resource called `"find_beer"` that has a value of `Find Beer!` To do this, edit *strings.xml* so that you add it as a new line like this:

```
<resources>
    <string name="app_name">Beer Adviser</string>
    <string name="find_beer">Find Beer!</string>
</resources>
```

This adds a new String resource called "find_beer".

Then add a new resource named `"brands"` with a value of `No beers selected`:

```
<resources>
    <string name="app_name">Beer Adviser</string>
    <string name="find_beer">Find Beer!</string>
    <string name="brands">No beers selected</string>
</resources>
```

This will be the default text in the text view.

Once you've updated the file, go to the File menu and choose the Save All option to save your changes. Next, we'll use the String resources in our layout.

Use the String resource in your layout

Create project
Update layout
Connect activity
Write logic

You use String resources in your layout using code like this:

```
android:text="@string/find_beer" />
```

You've seen the `android:text` part of the code before; it specifies what text should be displayed. But what does `"@string/find_beer"` mean?

Let's start with the first part, `@string`. This is just a way of telling Android to look up a text value from a String resource file. In our case, this is the file *strings.xml* that you just edited.

The second part, `find_beer`, tells Android to **look up the value of a resource with the name `find_beer`**. So `"@string/find_beer"` means "look up the String resource with the name `find_beer`, and use the associated text value."

Display the text...

...for the String resource find_beer.

```
android:text="@string/find_beer" />
```

We want to change the button and text view elements in our layout XML so that they use the two String resources we've just added.

Go back to the layout file *activity_find_beer.xml* file, and make the following code changes:

- Change the line:

  ```
  android:text="Button"
  ```

  to:

  ```
  android:text="@string/find_beer"
  ```

- Change the line:

  ```
  android:text="TextView"
  ```

  to:

  ```
  android:text="@string/brands"
  ```

You can see the code on the next page.

Android Studio sometimes displays the values of references in the code editor in place of actual code.

As an example, it may display the text `"Find Beer!"` *instead of the real code* `"@string/find_beer"`. *Any such substitutions should be highlighted in the code editor. If you click on them, or hover over them with your mouse, the true code will be revealed.*

```
<TextView
    android:text="Hello world!"
    android:text="@string/hello_world"
    android:layout_height="wrap_content" />
```

The code for activity_find_beer.xml

Here's the updated code for *activity_find_beer.xml* (changes are in bold); update your version of the file to match ours.

```
...
    <Spinner
        android:id="@+id/color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:layout_gravity="center"
        android:layout_margin="16dp" />

    <Button
        android:id="@+id/find_beer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="16dp"
        android:text="Button@string/find_beer" />

    <TextView
        android:id="@+id/brands"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="16dp"
        android:text="This is a text view@string/brands" />

</LinearLayout>
```

We didn't need to change the spinner. We'll look at how you add values to it over the next few pages.

This will display the value of the find_beer String resource on the button.

Delete the hardcoded text.

This will display the value of the brands String resource in the text view.

Delete this hardcoded text too.

When you're done, save your changes.

We've put a summary of adding and using String resources on the next page.

String Resource Files Up Close

strings.xml is the default resource file used to hold name/value pairs of Strings so that they can be referenced throughout your app. It has the following format:

The <resources> element identifies the contents of the file as resources.

The <string> element identifies the name/value pairs as Strings.

```
<resources>
    <string name="app_name">Beer Adviser</string>
    <string name="find_beer">Find Beer!</string>
    <string name="brands">No beer selected</string>
</resources>
```

There are two things that allow Android to recognize *strings.xml* as being a String resource file:

- **The file is held in the folder *app/src/main/res/values*.**
  XML files held in this folder contain simple values, such as Strings and colors.
- **The file has a <resources> element, which contains one or more <string> elements.**
  The format of the file itself indicates that it's a resource file containing Strings. The `<resources>` element tells Android that the file contains resources, and the `<string>` element identifies each String resource.

This means that you don't need to call your String resource file *strings.xml*; if you want, you can call it something else, or split your Strings into multiple files.

Each name/value pair takes the form:

```
<string name="string_name">string_value</string>
```

where `string_name` is the identifier of the String, and `string_value` is the String value itself.

A layout can retrieve the value of the String using:

```
"@string/string_name"
```

This is the name of the String whose value we want to return.

"@string" tells Android to look for a String resource of this name.

Time for a test drive

Let's see how the app's looking now. Save the changes you've made, then choose the "Run 'app'" command from the Run menu. When prompted, select the option to launch the emulator.

This time when we run the app, the text for the button and the text view has changed to the String values we added to *strings.xml*. The button says "Find Beer!" and the text view says "No beers selected."

Here's what we've done so far

Here's a quick recap of where we've got to:

1. **We've created a layout that specifies what the app looks like.**
   It includes a spinner, a button, and a text view.
2. **The file strings.xml includes the String resources we need.**
   We've added a label for the button, and default text for the list of suggested beer brands to try.
3. **The activity specifies how the app should interact with the user.**
   Android Studio has created an activity for us, but we haven't done anything with it yet.

Next we'll look at how you add a list of beers to the spinner.

there are no Dumb Questions

Q: Do I absolutely have to put my text values in a String resource file such as *strings.xml*?

A: It's not mandatory, but Android gives you warning messages if you hardcode text values. Using a String resource file might seem like a lot of effort at first, but it makes things like localization much easier. It's also easier to use String resources to start off with, rather than patching them in afterward.

Q: How does separating out the String values help with localization?

A: Suppose you want your application to be in English by default, but in French if the device language is set to French. Rather than hardcode different languages into your app, you can have one String resource file for English text, and another resource file for French text.

Q: How does the app know which String resource file to use?

A: Put your default English Strings resource file in the *app/src/main/res/values* folder as normal, and your French resource file in a new folder called *app/src/main/res/values-fr*. If the device is set to French, it will use the Strings in the *app/src/main/res/values-fr* folder. If the device is set to any other language, it will use the Strings in *app/src/main/res/values*.

Add values to the spinner

Create project
Update layout
Connect activity
Write logic

At the moment, the layout includes a spinner, but it doesn't have anything in it. Whenever you use a spinner, you need to get it to display a list of values so that the user can choose the value they want.

We can give the spinner a list of values in pretty much the same way that we set the text on the button and the text view: by using a **resource**. So far, we've used *strings.xml* to specify individual String values. For the spinner, all we need to do is specify an *array* of String values, and get the spinner to reference it.

Resources are noncode assets, such as images or Strings, used by your app.

Adding an array resource is similar to adding a String

As you already know, you can add a String resource to *strings.xml* using:

```
<string name="string_name">string_value</string>
```

where `string_name` is the identifier of the String, and `string_value` is the String value itself.

To add an array of Strings, you use the following syntax:

```
<string-array name="string_array_name">
    <item>string_value1</item>
    <item>string_value2</item>
    <item>string_value3</item>
    ...
</string-array>
```

This is the name of the array.

These are the values in the array. You can add as many as you need.

where `string_array_name` is the name of the array, and `string_value1`, `string_value2`, `string_value3` are the individual String values that make up the array.

Let's add a `string-array` resource to our app that can be used by the spinner.

Create project
Update layout
Connect activity
Write logic

Add the string-array to strings.xml

To add the `string-array`, open up *strings.xml*, and add the array like this:

```
...
    <string name="brands">No beer selected </string>
    <string-array name="beer_colors">
        <item>light</item>
        <item>amber</item>
        <item>brown</item>
        <item>dark</item>
    </string-array>
</resources>
```

Add this string-array to strings.xml. It defines an array of Strings called beer_colors with array items of light, amber, brown, and dark.

Get the spinner to reference a string-array

A layout can reference a `string-array` using similar syntax to how it would retrieve the value of a String. Rather than use:

```
"@string/string_name"
```

you use the syntax:

```
"@array/array_name"
```

Use @string to reference a String, and @array to reference an array.

where `array_name` is the name of the array.

Let's use this in the layout. Go to the layout file *activity_find_beer.xml* and add an `entries` attribute to the spinner like this:

```
...
    <Spinner
        android:id="@+id/color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:layout_gravity="center"
        android:layout_margin="16dp"
        android:entries="@array/beer_colors" />
...
```

This means "the entries for the spinner come from array beer_colors".

Those are all the changes you need in order to get the spinner to display a list of values. Let's see what it looks like.

Test drive the spinner

- Create project
- **Update layout**
- Connect activity
- Write logic

So let's see what impact these changes have had on our app. Save your changes, then run the app. You should get something like this:

Where we've got to

Here's a reminder of what we've done so far:

1. **We've created a layout that specifies what the app looks like.**
   It includes a spinner, a button, and a text view.

2. **The file strings.xml includes the String resources we need.**
   We've added a label for the button, default text for the suggested beer brands, and an array of values for the spinner.

3. **The activity specifies how the app should interact with the user.**
   Android Studio has created an activity for us, but we haven't done anything with it yet.

So what's next?

- Create project
- Update layout
- **Connect activity**
- Write logic

We need to make the button do something

What we need to do next is make the app react to the value we select in the spinner when the Find Beer button is clicked. We want our app to behave something like this:

1. **The user chooses a type of beer from the spinner.**
2. **The user clicks the Find Beer button, and the layout specifies which method to call in the activity.**
3. **The method in the activity retrieves the value of the selected beer in the spinner and passes it to the getBrands() method in a Java custom class called BeerExpert.**
4. **BeerExpert's getBrands() method finds matching brands for the type of beer and returns them to the activity as an ArrayList of Strings.**
5. **The activity gets a reference to the layout text view and sets its text value to the list of matching beers.**

After all those steps are completed, the list is displayed on the device.

Let's start by getting the button to call a method.

Create project
Update layout
Connect activity
Write logic

Make the button call a method

Whenever you add a button to a layout, it's likely you'll want it to do something when the user clicks on it. To make this happen, you need to get the button to call a method in your activity.

To get our button to call a method in the activity when it's clicked, we need to make changes to two files:

- **Change the layout file activity_find_beer.xml.** We'll specify which method in the activity will get called when the button is clicked.
- **Change the activity file FindBeerActivity.java.** We need to write the method that gets called.

Let's start with the layout.

Use onClick to say which method the button calls

It only takes one line of XML to tell Android which method a button should call when it's clicked. All you need to do is add an `android:onClick` attribute to the `<button>` element, and tell it the name of the method you want to call:

```
android:onClick="method_name"
```

This means "when the component is clicked, call the method in the activity called method_name".

Let's try this now. Go to the layout file *activity_find_beer.xml*, and add a new line of XML to the `<button>` element to say that the method `onClickFindBeer()` should be called when the button is clicked:

```
...
    <Button
        android:id="@+id/find_beer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="16dp"
        android:text="@string/find_beer"
        android:onClick="onClickFindBeer" />
...
```

When the button is clicked, call the method onClickFindBeer() in the activity. We'll create the method in the activity over the next few pages.

Once you've made these changes, save the file.

Now that the layout knows which method to call in the activity, we need to write the method. Let's take a look at the activity.

What activity code looks like

When we first created a project for our app, we asked the wizard to create an empty activity called `FindBeerActivity`. The code for this activity is held in a file called *FindBeerActivity.java*. Open this file by going to the *app/src/main/java* folder and double-clicking on it.

When you open the file, you'll see that Android Studio has generated some Java code for you. Rather than taking you through all the code that Android Studio may (or may not) have created, we want you to replace the code that's currently in *FindBeerActivity.java* with the code shown here:

```
package com.hfad.beeradviser;

import android.app.Activity;
import android.os.Bundle;

public class FindBeerActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }
}
```

Make sure class extends the Android Activity class.

This is the onCreate() method. It's called when the activity is first created.

setContentView() tells Android which layout the activity uses. In this case, it's activity_find_beer.

The above code is all you need to create a basic activity. As you can see, it's a class that extends the `android.app.Activity` class, and implements an `onCreate()` method.

All activities (not just this one) have to extend the `Activity` class or one of its subclasses. The `Activity` class contains a bunch of methods that transform your Java class from a plain old Java class into a full-fledged, card-carrying Android activity.

All activities also need to implement the `onCreate()` method. This method gets called when the activity object gets created, and it's used to perform basic setup such as what layout the activity is associated with. This is done via the `setContentView()` method. In the example above, `setContentView(R.layout.activity_find_beer)` tells Android that this activity uses `activity_find_beer` as its layout.

On the previous page, we added an `onClick` attribute to the button in our layout and gave it a value of `onClickFindBeer`. We need to add this method to our activity so it will be called when the button gets clicked. This will enable the activity to respond when the user touches the button in the user interface.

Do this!

Replace the code in your version of *FindBeerActivity.java* with the code shown on this page.

Add an onClickFindBeer() method to the activity

- Create project
- Update layout
- **Connect activity**
- Write logic

The `onClickFindBeer()` method needs to have a particular signature, or otherwise it won't get called when the button specified in the layout gets clicked. The method needs to take the following form:

```
public void onClickFindBeer(View view) {
}
```

The method must be public.

The method must have a void return value.

The method must have a single parameter of type View.

If the method doesn't take this form, then it won't respond when the user presses the button. This is because behind the scenes, Android looks for a public method with a void return value, with a method name that matches the method specified in the layout XML.

The `View` parameter in the method may seem unusual at first glance, but there's a good reason for it being there. The parameter refers to the GUI component that triggers the method (in this case, the button). As we mentioned earlier, GUI components such as buttons and text views are all types of `View`.

So let's update our activity code. Add the `onClickFindBeer()` method below to your activity code (*FindBeerActivity.java*):

If you want a method to respond to a button click, it must be public, have a void return type, and take a single View parameter.

We're using this class, so we need to import it.

```
...
import android.view.View;

public class FindBeerActivity extends Activity {
...
    //Called when the user clicks the button
    public void onClickFindBeer(View view){
    }
}
```

Add the onClickFindBeer() method to FindBeerActivity.java.

- Create project
- Update layout
- Connect activity
- **Write logic**

onClickFindBeer() needs to do something

Now that we've created the `onClickFindBeer()` method in our activity, the next thing we need to do is get the method to do something when it runs. Specifically, we need to get our app to display a selection of different beers that match the beer type the user has selected.

In order to achieve this, we first need to get a reference to both the spinner and text view GUI components in the layout. This will allow us to retrieve the value of the chosen beer type from the spinner, and display text in the text view.

Use findViewById() to get a reference to a view

We can get references for our two GUI components using a method called `findViewById()`. This method takes the ID of the GUI component as a parameter, and returns a `View` object. You then cast the return value to the correct type of GUI component (for example, a `TextView` or a `Button`).

Here's how you'd use `findViewById()` to get a reference to the text view with an ID of `brands`:

We want the view with an ID of brands.

```
TextView brands = (TextView) findViewById(R.id.brands);
```

brands is a TextView, so we have to cast it as one.

Take a closer look at how we specified the ID of the text view. Rather than pass in the name of the text view, we passed in an ID of the form `R.id.brands`. So what does this mean? What's R?

R.java is a special Java file that gets generated by Android Studio whenever you create or build your app. It lives within the *app/build/generated/source/r/debug* folder in your project in a package with the same name as the package of your app. Android uses *R.java* to keep track of the resources used within the app, and among other things it enables you to get references to GUI components from within your activity code.

If you open up *R.java*, you'll see that it contains a series of inner classes, one for each type of resource. Each resource of that type is referenced within the inner class. As an example, *R.java* includes an inner class called `id`, and the inner class includes a `static final brands` value. Android added this code to *R.java* when we used the code `"@+id/brands"` in our layout. The line of code:

```
(TextView) findViewById(R.id.brands);
```

uses the value of `brands` to get a reference to the `brands` text view.

R is a special Java class that enables you to retrieve references to resources in your app.

***R.java* gets generated for you.**

You never change any of the code within this file, but it's useful to know it's there.

Once you have a view, you can access its methods

The `findViewById()` method provides you with a Java version of your GUI component. This means that you can get and set properties in the GUI component using the methods exposed by the Java class. Let's take a closer look.

Setting the text in a text view

As you've seen, you can get a reference to a text view in Java using:

```
TextView brands = (TextView) findViewById(R.id.brands);
```

When this line of code gets called, it creates a `TextView` object called `brands`. You are then able to call methods on this `TextView` object.

Let's say you wanted to set the text displayed in the `brands` text view to "Gottle of geer". The `TextView` class includes a method called `setText()` that you can use to change the text property. You use it like this:

```
brands.setText("Gottle of geer");
```

Set the text on the brands TextView to "Gottle of geer".

Retrieving the selected value in a spinner

You can get a reference to a spinner in a similar way to how you get a reference to a text view. You use the `findViewById()` method as before, but this time you cast the result as a spinner:

```
Spinner color = (Spinner) findViewById(R.id.color);
```

This gives you a `Spinner` object whose methods you can now access. As an example, here's how you retrieve the currently selected item in the spinner, and convert it to a String:

```
String.valueOf(color.getSelectedItem())
```

This gets the selected item in the spinner and converts it to a String.

The code:

```
color.getSelectedItem()
```

actually returns a generic Java object. This is because spinner values can be something other than Strings, such as images. In our case, we know the values are all Strings, so we can use `String.valueOf()` to convert the selected item from an `Object` to a `String`.

Update the activity code

You now know enough to write some code in the `onClickFindBeer()` method. Rather than write all the code we need in one go, let's start by reading the selected value from the spinner, and displaying it in the text view.

Activity Magnets

Somebody used fridge magnets to write a new **`onClickFindBeer()`** method for us to slot into our activity. Unfortunately, a freak kitchen whirlwind has dislodged the magnets. Can you piece the code back together again?

The code needs to retrieve the type of beer selected in the spinner, and then display the type of beer in the text view.

```
//Called when the button gets clicked
public void onClickFindBeer(                view) {

    //Get a reference to the TextView
                brands =                    (                    );

    //Get a reference to the Spinner
    Spinner             =                   (                    );

    //Get the selected item in the Spinner
    String              = String.valueOf(color.                  );

    //Display the selected item
    brands.             (beerType);
}
```

TextView | color | findViewById | setText | R.id.color | R.view.brands | R.id.brands | (TextView) | findView | Button | findView | View | R.view.color | findViewById | (Spinner) | getSelectedItem() | beerType

You won't need to use all of the magnets.

Activity Magnets Solution

Somebody used fridge magnets to write a new **`onClickFindBeer()`** method for us to slot into our activity. Unfortunately, a freak kitchen whirlwind has dislodged the magnets. Can you piece the code back together again?

The code needs to retrieve the type of beer selected in the spinner, and then display the type of beer in the text view.

```
//Called when the button gets clicked
public void onClickFindBeer( View view) {

    //Get a reference to the TextView
    TextView brands = (TextView) findViewById( R.id.brands );

    //Get a reference to the Spinner
    Spinner color = (Spinner) findViewById( R.id.color );

    //Get the selected item in the Spinner
    String beerType = String.valueOf(color. getSelectedItem() );

    //Display the selected item
    brands. setText (beerType);
}
```

`R.view.brands`

`findView`

`Button`

`R.view.color`

`findView`

You didn't need to use these magnets.

- Create project
- Update layout
- Connect activity
- **Write logic**

The first version of the activity

Our cunning plan is to build the activity in stages and test it as we go along. In the end, the activity will take the selected value from the spinner, call a method in a custom Java class, and then display matching types of beer. For this first version, our goal is just to make sure that we correctly retrieve the selected item from the spinner.

Here is our activity code, including the method you pieced together on the previous page. Apply these changes to *FindBeerActivity.java*, then save them:

```
package com.hfad.beeradviser;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;

public class FindBeerActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }

    //Called when the button gets clicked
    public void onClickFindBeer(View view) {
        //Get a reference to the TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Get a reference to the Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Get the selected item in the Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Display the selected item
        brands.setText(beerType);
    }
}
```

We're using these extra classes so we need to import them.

We've not changed this method.

findViewById returns a View. You need to cast it to the right type of View.

getSelectedItem returns an Object. You need to turn it into a String.

What the code does

Create project
Update layout
Connect activity
Write logic

Before we take the app for a test drive, let's look at what the code actually does.

1. **The user chooses a type of beer from the spinner and clicks on the Find Beer button. This calls the public void onClickFindBeer(View) method in the activity.**
   The layout specifies which method in the activity should be called when the button is clicked via the button's `android:onClick` property.

2. **The activity gets references to the Spinner and TextView GUI components using calls to the findViewById() method.**

3. **The activity retrieves the currently selected value of the spinner (in this case amber), and converts it to a String.**

4. **The activity then sets the text property of the TextView to reflect the currently selected item in the spinner.**

Test drive the changes

Make the changes to the activity file, save it, and then run your app. This time when we click on the Find Beer button, it displays the value of the selected item in the spinner.

there are no Dumb Questions

Q: I added a String to my *strings.xml* file, but I can't see it in *R.java*. Why isn't it there?

A: Android Studio, generates *R.java* when you save any changes you've made. If you've added a resource but can't see it in *R.java*, check that your changes have been saved.

R.java also gets updated when the app gets built. The app builds when you run the app, so running the app will also update *R.java*.

Q: The values in the spinner look like they're static as they're set to the values in the `string-array`. Can I change these values programmatically?

A: You can, but that approach is more complicated than just using static values. We'll show you later in the book how you can have complete control over the values displayed in components such as spinners.

Q: What type of object is returned by `getSelectedItem()`?

A: It's declared as type `Object`. Because we used a `string-array` for the values, the actual value returned in this case is a String.

Q: What do you mean "in this case"—isn't it always?

A: You can do more complicated things with spinners than just display text. As an example, the spinner might display an icon next to each value. As `getSelectedItem()` returns an object, it gives you a bit more flexibility than just returning a String.

Q: Does the name of `onClickFindBeer` matter?

A: All that matters is that the name of the method in the activity code matches the name used in the button's `onClick` attribute in the layout.

Q: Why did we have to replace the activity code that Android Studio created for us?

A: IDEs such as Android Studio include functions and utilities that can save you a lot of time. They generate a lot of code for you, and sometimes this can be useful. But when you're learning a new language or development area such as Android, we think it's best to learn about the fundamentals of the language rather than what the IDE generates for you. This way you'll develop a greater understanding of the language.

Build the custom Java class

As we said at the beginning of the chapter, the Beer Adviser app decides which beers to recommend with the help of a custom Java class. This Java class is written in plain old Java, with no knowledge of the fact it's being used by an Android app.

Custom Java class spec

The custom Java class should meet the following requirements:

- The package name should be `com.hfad.beeradviser`.
- The class should be called `BeerExpert`.
- It should expose one method, `getBrands()`, that takes a preferred beer color (as a String), and return a `List<String>` of recommended beers.

Build and test the Java class

Java classes can be extremely complicated and involve calls to complex application logic. You can either build and test your own version of the class, or use our sophisticated version of the class shown here:

```
package com.hfad.beeradviser;

import java.util.ArrayList;
import java.util.List;

public class BeerExpert {
    List<String> getBrands(String color) {
        List<String> brands = new ArrayList<>();
        if (color.equals("amber")) {
            brands.add("Jack Amber");
            brands.add("Red Moose");
        } else {
            brands.add("Jail Pale Ale");
            brands.add("Gout Stout");
        }
        return brands;
    }
}
```

This is pure Java code; nothing Androidy about it.

Do this!

Add the BeerExpert class to your project. Select the *com.hfad.beeradviser* package in the *app/src/main/java* folder, go to File→New...→Java Class, name the file "BeerExpert", and make sure the package name is "com.hfad.beeradviser". This creates the *BeerExpert.java* file.

- Create project
- Update layout
- Connect activity
- **Write logic**

Enhance the activity to call the custom Java class so that we can get REAL advice

In version two of the activity we need to enhance the `onClickFindBeer()` method to call the `BeerExpert` class for beer recommendations. The code changes needed are plain old Java. You can try to write the code and run the app on your own, or you can follow along with us. But before we show you the code changes, try the exercise below; it'll help you create some of the activity code you'll need.

Sharpen your pencil

Enhance the activity so that it calls the `BeerExpert` `getBrands()` method and displays the results in the text view.

```
package com.hfad.beeradviser;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
...
    //Called when the button gets clicked
    public void onClickFindBeer(View view) {
        //Get a reference to the TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Get a reference to the Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Get the selected item in the Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Get recommendations from the BeerExpert class

        }
}
```

We added this line for you.

You'll need to use the BeerExpert class to get the beer recommendations, so we added this line for you too.

You need to update the onClickFindBeer() method.

Sharpen your pencil Solution

Enhance the activity so that it calls the `BeerExpert getBrands()` method and displays the results in the text view.

```
package com.hfad.beeradviser;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
...
    //Called when the button gets clicked
    public void onClickFindBeer(View view) {
        //Get a reference to the TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Get a reference to the Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Get the selected item in the Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Get recommendations from the BeerExpert class

        List<String> brandsList = expert.getBrands(beerType);
        StringBuilder brandsFormatted = new StringBuilder();
        for (String brand : brandsList) {
            brandsFormatted.append(brand).append('\n');
        }
        //Display the beers
        brands.setText(brandsFormatted);

    }
}
```

Get a List of brands.

Build a String using the values in the List.

Display each brand on a new line.

Display the results in the text view.

Using the BeerExpert requires pure Java code, so don't worry if your code looks a little different than ours.

- Create project
- Update layout
- Connect activity
- **Write logic**

Activity code version 2

Here's our full version of the activity code. Apply the changes shown here to your version of *FindBeerActivity.java*, make sure you've added the `BeerExpert` class to your project, and then save your changes:

```
package com.hfad.beeradviser;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }

    //Called when the button gets clicked
    public void onClickFindBeer(View view) {
        //Get a reference to the TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Get a reference to the Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Get the selected item in the Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Get recommendations from the BeerExpert class
        List<String> brandsList = expert.getBrands(beerType);
        StringBuilder brandsFormatted = new StringBuilder();
        for (String brand : brandsList) {
            brandsFormatted.append(brand).append('\n');
        }
        //Display the beers
        brands.setText(brandsFormatted);
        brands.setText(beerType);
    }
}
```

We're using this extra class so we need to import it.

Add an instance of BeerExpert as a private variable.

Use the BeerExpert class to get a List of brands.

Build a String, displaying each brand on a new line.

Display the String in the TextView.

Delete this line.

What happens when you run the code

1. **When the user clicks on the Find Beer button, the onClickFindBeer() method in the activity gets called.**
The method creates a reference to the spinner and text view, and gets the currently selected value from the spinner.

2. **onClickFindBeer() calls the getBrands() method in the BeerExpert class, passing in the type of beer selected in the spinner.**
The `getBrands()` method returns a list of brands.

3. **The onClickFindBeer() method formats the list of brands and uses it to set the text property in the text view.**

Test drive your app

Once you've made the changes to your app, go ahead and run it. Try selecting different types of beer and clicking on the Find Beer button.

When you choose different types of beer and click on the Find Beer button, the app uses the `BeerExpert` class to provide you with a selection of suitable beers.

Your Android Toolbox

You've got Chapter 2 under your belt and now you've added building interactive Android apps to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- The `<Button>` element is used to add a button.
- The `<Spinner>` element is used to add a spinner, which is a drop-down list of values.
- All GUI components are types of view. They inherit from the Android `View` class.
- *strings.xml* is a String resource file. It's used to separate out text values from the layouts and activities, and supports localization.
- Add a String to *strings.xml* using:

```
<string name="name">Value</string>
```

- Reference a String in the layout using:

```
"@string/name"
```

- Add an array of String values to *strings.xml* using:

```
<string-array name="array">
    <item>string1</item>
    ...
</string-array>
```

- Reference a `string-array` in the layout using:

```
"@array/array_name"
```

- Make a button call a method when clicked by adding the following to the layout:

```
android:onClick="clickMethod"
```

There needs to be a corresponding method in the activity:

```
public void clickMethod(View view){
}
```

- *R.java* is generated for you. It enables you to get references for layouts, GUI components, Strings, and other resources in your Java code.
- Use `findViewById()` to get a reference to a view.
- Use `setText()` to set the text in a view.
- Use `getSelectedItem()` to get the selected item in a spinner.
- Add a custom class to an Android project by going to File menu→New...→Java Class.

3 multiple activities and intents

State Your Intent

Most apps need more than one activity.

So far we've just looked at single-activity apps, which is fine for simple apps. But when things get more complicated, just having the one activity won't cut it. We're going to show you **how to build apps with multiple activities**, and how you can get your activities talking to each other using ***intents***. We'll also look at how you can use intents to **go beyond the boundaries of your app** and **make activities in other apps on your device perform *actions***. Things are about to get a whole lot more powerful...

Apps can contain more than one activity

Earlier in the book, we said that an activity is a single, defined thing that your user can do, such as displaying a list of recipes. If your app is simple, this may be all that's needed.

But a lot of the time, you'll want users to do *more* than just one thing—for example, adding recipes as well as displaying a list of them. If this is the case, you'll need to use multiple activities: one for displaying the list of recipes and another for adding a single recipe.

The best way of understanding how this works is to see it in action. You're going to build an app containing two activities. The first activity will allow you to type a message. When you click on a button in the first activity, it will launch the second activity and pass it the message. The second activity will then display the message.

An activity is a single focused thing your user can do. If you chain multiple activities together to do something more complex, it's called a task.

Here's what we're going to do in this chapter

1. Create an app with a single activity and layout.
2. Add a second activity and layout.
3. Get the first activity to call the second activity.
4. Get the first activity to pass data to the second activity.

Here's the app structure

The app contains two activities and two layouts.

1. **When the app gets launched, it starts activity CreateMessageActivity.**
   This activity uses the layout *activity_create_message.xml*.

2. **When the user clicks a button in CreateMessageActivity, ReceiveMessageActivity is launched.**
   This activity uses layout *activity_receive_message.xml*.

Get started: create the project

Create 1st activity
Create 2nd activity
Call 2nd activity
Pass data

You create a project for the app in exactly the same way you did in previous chapters. Create a new Android Studio project for an application named "My Messenger" with a company domain of "hfad.com", making the package name `com.hfad.mymessenger`. The minimum SDK should be API 19 so that it will work on most devices. You'll need an empty activity named "CreateMessageActivity" with a layout named "activity_create_message" so that your code matches ours. **Make sure that you untick the Backwards Compatibility (AppCompat) option when you create the activity.**

On the next page, we'll update the activity's layout.

Update the layout

Here's the XML for the *activity_create_message.xml* file. We're using a `<LinearLayout>` to display components in a single column, and we've added `<Button>` and `<EditText>` elements to it. The `<EditText>` element gives you an editable text field you can use to enter data.

Change your *activity_create_message.xml* file to match the XML here:

This is the editable text field. If it's empty, it gives the user a hint about what text they should enter in it.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.mymessenger.CreateMessageActivity">

    <EditText
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:hint="@string/hint"
        android:ems="10" />

    <Button
        android:id="@+id/send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:onClick="onSendMessage"
        android:text="@string/send" />

</LinearLayout>
```

We're using a linear layout with a vertical orientation.

This creates an editable text field.

The hint attribute gives the user a hint of what text they should type into the text field. We need to add it as a String resource.

This describes how wide the <EditText> should be. It should be wide enough to accommodate 10 letter Ms.

This is a String resource we need to create.

Clicking on the button runs the onSendMessage() method in the activity.

The <EditText> element defines an editable text field for entering text. It inherits from the same Android View class as the other GUI components we've seen so far.

Create 1st activity
Create 2nd activity
Call 2nd activity
Pass data

Update strings.xml...

We used two String resources in our layout on the previous page. The button has a text value of `@string/send` that appears on the button, and the editable text field has a hint value of `@string/hint` that tells the user what to enter in the field. This means we need to add Strings called `"send"` and `"hint"` to *strings.xml* and give them values. Do this now:

```
<resources>
    ...
    <string name="send">Send Message</string>
    <string name="hint">Enter a message</string>
</resources>
```

The text "Send Message" will appear on the button.

The text "Enter a message" will appear as a hint in the text field if it's empty.

...and add the method to the activity

This line in the `<Button>` element:

```
android:onClick="onSendMessage"
```

means that the `onSendMessage()` method in the activity will fire when the button is clicked. Let's add this method to the activity now.

Open up the *CreateMessageActivity.java* file and replace the code Android Studio created for you with the following:

```
package com.hfad.mymessenger;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
    }
}
```

Make sure your activity extends the Activity class.

The onCreate() method gets called when the activity is created.

This method will get called when the button's clicked. We'll complete the method body as we work our way through the rest of the chapter.

Now that you've created the first activity, let's move on to the second.

Create the second activity and layout

- Create 1st activity
- **Create 2nd activity**
- Call 2nd activity
- Pass data

Android Studio has a wizard that lets you add extra activities and layouts to your apps. It's like a scaled-down version of the wizard you use to create an app, and you use it whenever you want to create a new activity.

To create the new activity, switch to the Project view of Android Studio's explorer, click on the *com.hfad.mymessenger* package in the *app/src/main/java* folder, choose File → New → Activity, and choose the option for Empty Activity. You will be presented with a new screen where you can choose options for your new activity.

Every time you create a new activity and layout, you need to name them. Name the new activity "ReceiveMessageActivity" and the layout "activity_receive_message". Make sure that the option to generate a layout is checked, and the Launcher Activity and Backwards Compatibility (AppCompat) options are unchecked. Finally, confirm that the package name is `com.hfad.mymessenger`, and when you're done, click on the Finish button.

Some versions of Android Studio may ask you what the source language of your activity should be. If prompted, select the option for Java.

Name the activity "ReceiveMessageActivity", and the layout "activity_receive_message."

Make sure the option to generate the layout is checked.

Uncheck the Launcher Activity and Backwards Compatibility (AppCompat) options.

- ✓ Create 1st activity
- ➔ **Create 2nd activity**
- Call 2nd activity
- Pass data

What just happened?

When you clicked on the Finish button, Android Studio created a shiny new activity file for you, along with a new layout. If you look in the explorer, you should see that a new file called *ReceiveMessageActivity.java* has appeared in the *app/src/main/java* folder, and a file called *activity_receive_message.xml* has appeared under *app/src/main/res/layout*.

Each activity uses a different layout. `CreateMessageActivity` uses the layout *activity_create_message.xml*, and `ReceiveMessageActivity` uses the layout *activity_receive_message.xml*:

Your explorer window may look different than ours because we've switched to the Project view.

Behind the scenes, Android Studio also made a configuration change to the app in a file called *AndroidManifest.xml*. Let's take a closer look.

Welcome to the Android manifest file

Every Android app must include a file called *AndroidManifest.xml*. You can find it in the *app/src/main* folder of your project. The *AndroidManifest.xml* file contains essential information about your app, such as what activities it contains, required libraries, and other declarations. Android creates the file for you when you create the app. If you think back to the settings you chose when you created the project, some of the file contents should look familiar.

Here's what our copy of *AndroidManifest.xml* looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.mymessenger">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".CreateMessageActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".ReceiveMessageActivity"></activity>

    </application>

</manifest>
```

This is the package name we specified.

Android Studio gave our app default icons.

The theme affects the appearance of the app. We'll look at this later.

This is the first activity, Create Message Activity.

This bit specifies that it's the main activity of the app.

This says the activity can be used to launch the app.

This is the second activity, ReceiveMessageActivity. Android Studio added this code when we added the second activity.

Watch it!

If you develop Android apps without an IDE, you'll need to create this file manually.

Every activity needs to be declared

All activities need to be declared in *AndroidManifest.xml*. If an activity isn't declared in the file, the system won't know it exists. And if the system doesn't know it exists, the activity will never run.

You declare an activity in the manifest by including an `<activity>` element inside the `<application>` element. In fact, *every* activity in your app needs a corresponding `<activity>` element. Here's the general format:

```
<application
    ...
    ...>
    <activity
        android:name=".MyActivityClassName"
        ...
        ...>
    ...
    </activity>
    ...
</application>
```

Each activity needs to be declared inside the <application> element.

This line is mandatory; just replace MyActivityClassName with the name of your activity.

The activity may have other properties too.

The following line is mandatory and is used to specify the class name of the activity, in this example `"MyActivityClassName"`:

```
android:name=".MyActivityClassName"
```

`MyActivityClassName` is the name of the class. It's prefixed with a "." because Android combines the class name with the name of the package to derive the *fully qualified* class name.

The activity declaration may include other properties too, such as security permissions, and whether it can be used by activities in other apps.

The second activity in our app was automatically declared because we added it using the Android Studio wizard.

If you add extra activities manually, you'll need to edit AndroidManifest.xml *yourself. The same may be true if you use another IDE besides Android Studio.*

An intent is a type of message

- Create 1st activity
- Create 2nd activity
- **Call 2nd activity**
- Pass data

You start an activity by creating an intent and using it in the startActivity() method.

So far we've created an app with two activities in it, and each activity has its own layout. When the app is launched, our first activity, `CreateMessageActivity`, will run. What we need to do next is get `CreateMessageActivity` to call `ReceiveMessageActivity` when the user clicks the Send Message button.

Whenever you want an activity to start a second activity, you use an **intent**. You can think of an intent as an "intent to do something." It's a type of message that allows you to bind separate objects (such as activities) together at runtime. If one activity wants to start a second activity, it does it by sending an intent to Android. Android will then start the second activity and pass it the intent.

You can create and send an intent using just a couple of lines of code. You start by creating the intent like this:

```
Intent intent = new Intent(this, Target.class);
```

The intent specifies the activity you want to receive it. It's like putting an address on an envelope.

The first parameter tells Android which object the intent is from: you can use the word `this` to refer to the current activity. The second parameter is the class name of the activity that needs to receive the intent.

Once you've created the intent, you pass it to Android like this:

```
startActivity(intent);
```

startActivity() starts the activity specified in the intent..

This tells Android to start the activity specified by the intent.

Once Android receives the intent, it checks that everything's OK and tells the activity to start. If it can't find the activity, it throws an **`ActivityNotFoundException`**.

- ✓ Create 1st activity
- ✓ Create 2nd activity
- → Call 2nd activity
- Pass data

Use an intent to start the second activity

Let's put this into practice and use an intent to call `ReceiveMessageActivity`. We want to launch the activity when the user clicks on the Send Message button, so we'll add the two lines of code we discussed on the previous page to our `onSendMessage()` method.

Make the changes highlighted below:

```
package com.hfad.mymessenger;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        startActivity(intent);
    }

}
```

We need to import the Intent class android.content.Intent as we're using it in onSendMessage().

Start activity ReceiveMessageActivity.

So what happens now when we run the app?

What happens when you run the app

Before we take the app out for a test drive, let's go over how the app we've developed so far will function:

1. **When the app gets launched, the main activity, CreateMessageActivity, starts.** When it starts, the activity specifies that it uses layout *activity_create_message.xml*. This layout gets displayed in a new window.

2. **The user types in a message and then clicks on the button.** The `onSendMessage()` method in `CreateMessageActivity` responds to the click.

3. **The onSendMessage() method uses an intent to tell Android to start activity ReceiveMessageActivity.** Android checks that the intent is valid, and then it tells `ReceiveMessageActivity` to start.

The story continues...

4. When ReceiveMessageActivity starts, it specifies that it uses layout *activity_receive_message.xml* and this layout gets displayed in a new window.

Test drive the app

Save your changes, and then run the app. `CreateMessageActivity` starts, and when you click on the Send Message button, it launches `ReceiveMessageActivity`.

- [x] Create 1st activity
- [x] Create 2nd activity
- [x] Call 2nd activity
- [ ] Pass data

Pass text to a second activity

- Create 1st activity
- Create 2nd activity
- Call 2nd activity
- **Pass data**

So far we've coded `CreateMessageActivity` to start `ReceiveMessageActivity` when the Send Message button is pressed. Next, we'll get `CreateMessageActivity` to pass text to `ReceiveMessageActivity` so that `ReceiveMessageActivity` can display it. In order to accomplish this, we'll do three things:

1. Tweak the layout *activity_receive_message.xml* so that it can display the text. At the moment it's simply the default layout the wizard gave us.
2. Update *CreateMessageActivity.java* so that it gets the text the user inputs, and then adds the text to the intent before it sends it.
3. Update *ReceiveMessageActivity.java* so that it displays the text sent in the intent.

Let's start with the layout

We'll begin by changing the *activity_receive_message.xml* code Android Studio created for us so that it uses a `<LinearLayout>`. Update your version of the code so that it matches ours:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.mymessenger.ReceiveMessageActivity">

</LinearLayout>
```

We're going to use a linear layout with a vertical orientation as we did in activity_create_message.xml.

Exercise

We need to change the layout so that it includes a text view. The text view needs to have an ID of "message" so that we can reference it in our activity code. How should we change the layout's code? Think about this before looking at the next page.

- [x] Create 1st activity
- [x] Create 2nd activity
- [x] Call 2nd activity
- [ ] **Pass data**

Update the text view properties

We need to add a `<TextView>` element to the layout, and give it an ID of "message." This is because you have to add an ID to any GUI components you need to reference in your activity code, and we need to reference the text view so that we can update the text it displays.

We've updated our code so that it includes a new text view. Update your *activity_receive_message.xml* code so that it reflects ours (we've bolded our changes):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.mymessenger.ReceiveMessageActivity">

    <TextView
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

This adds the text view.

This line gives the text view an ID of "message".

We've not specified default text for the text view, as the only text we'll ever want to display in the text view is the message passed to it by `CreateMessageActivity`.

Now that we've updated the layout, we can get to work on the activities. Let's start by looking at how we can use an intent to pass a message to `ReceiveMessageActivity`.

there are no Dumb Questions

Q: Do I have to use intents? Can't I just construct an instance of the second activity in the code for my first activity?

A: That's a good question, but no, that's not the "Android way" of doing things. One of the reasons is that passing intents to Android tells Android the sequence in which activities are started. This means that when you click on the Back button on your device, Android knows exactly where to take you back to.

putExtra() puts extra info in an intent

You've seen how you can create a new intent using:

```
Intent intent = new Intent(this, Target.class);
```

You can add extra information to this intent that can be picked up by the activity you're targeting so it can react in some way. To do this, you use the `putExtra()` method like so:

```
intent.putExtra("message", value);
```

putExtra() lets you put extra information in the message you're sending.

where `message` is a String name for the value you're passing in, and `value` is the value. The `putExtra()` method is overloaded so `value` has many possible types. As an example, it can be a primitive such as a boolean or int, an array of primitives, or a String. You can use `putExtra()` repeatedly to add numerous extra data to the intent. If you do this, make sure you give each one a unique name.

There are many different options for the type of value. You can see them all in the Google Android documentation. Android Studio will also give you a list as you type code in.

How to retrieve extra information from an intent

The story doesn't end there. When Android tells `ReceiveMessageActivity` to start, `ReceiveMessageActivity` needs some way of retrieving the extra information that `CreateMessageActivity` sent to Android in the intent.

There are a couple of useful methods that can help with this. The first of these is:

```
getIntent();
```

`getIntent()` returns the intent that started the activity, and you can use this to retrieve any extra information that was sent along with it. How you do this depends on the type of information that was sent. As an example, if you know the intent includes a String value named `"message"`, you would use the following:

```
Intent intent = getIntent();
String string = intent.getStringExtra("message");
```

Get the intent.

Get the String passed along with the intent that has a name of "message".

You're not just limited to retrieving String values. As an example, you can use:

```
int intNum = intent.getIntExtra("name", default_value);
```

to retrieve an int with a name of `name`. `default_value` specifies what int value you should use as a default.

Pool Puzzle

Your **job** is to take code snippets from the pool and place them into the blank lines in *CreateMessageActivity.java*. You may **not** use the same code snippet more than once, and you won't need to use all the code snippets. Your **goal** is to make the activity retrieve text from the message `<EditText>` and add it to the intent.

```
package com.hfad.mymessenger;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
.......................................................................

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        ...................................................................
        ...................................................................
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        ...................................................................
        startActivity(intent);
    }
}
```

Note: each thing from the pool can only be used once!

Pool Puzzle Solution

Your **job** is to take code snippets from the pool and place them into the blank lines in *CreateMessageActivity.java*. You may **not** use the same code snippet more than once, and you won't need to use all the code snippets. Your **goal** is to make the activity retrieve text from the message `<EditText>` and add it to the intent.

```
package com.hfad.mymessenger;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        EditText messageView = (EditText) findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra("message", messageText);
        startActivity(intent);
    }
}
```

You need to import the EditText class.

Get the text from the editable text field with an ID of "message."

Add the text to the intent, giving it a name of "message".

- ✓ Create 1st activity
- ✓ Create 2nd activity
- ✓ Call 2nd activity
- → Pass data

Update the CreateMessageActivity code

We updated our code for *CreateMessageActivity.java* so that it takes the text the user enters on the screen and adds it to the intent. Here's the full code (make sure you update your code to include these changes, shown in bold):

```
package com.hfad.mymessenger;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        startActivity(intent);
    }
}
```

You need to import the EditText class android.widget.EditText as you're using it in your activity code.

Get the text that's in the EditText.

Create an intent, then add the text to the intent. We're using a constant for the name of the extra information so that we know CreateMessageActivity and ReceiveMessageActivity are using the same String. We'll add this constant to ReceiveMessageActivity on the next page, so don't worry if Android Studio says it doesn't exist.

Start ReceiveMessageActivity with the intent.

Now that `CreateMessageActivity` has added extra information to the intent, we need to retrieve that information and use it.

Get ReceiveMessageActivity to use the information in the intent

Now that we've changed `CreateMessageActivity` to add text to the intent, we'll update `ReceiveMessageActivity` so that it uses that text.

We're going to get `ReceiveMessageActivity` to display the message in its text view when the activity gets created. Because the activity's `onCreate()` method gets called as soon as the activity is created, we'll add the code to this method.

To get the message from the intent, we'll first get the intent using the `getIntent()` method, then get the value of the message using `getStringExtra()`.

Here's the full code for *ReceiveMessageActivity.java* (replace the code that Android Studio generated for you with this code, and then save all your changes):

```
package com.hfad.mymessenger;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.widget.TextView;

public class ReceiveMessageActivity extends Activity {

    public static final String EXTRA_MESSAGE = "message";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receive_message);
        Intent intent = getIntent();
        String messageText = intent.getStringExtra(EXTRA_MESSAGE);
        TextView messageView = (TextView)findViewById(R.id.message);
        messageView.setText(messageText);
    }
}
```

We need to import these classes.

Make sure your activity extends the Activity class.

This is the name of the extra value we're passing in the intent.

Get the intent, and get the message from it using getStringExtra().

Add the text to the message text view.

Before we take the app for a test drive, let's run through what the current code does.

What happens when the user clicks the Send Message button

1. **When the user clicks on the button, the onSendMessage() method is called.**
   Code within the `onSendMessage()` method creates an intent to start activity `ReceiveMessageActivity`, adds a message to the intent, and passes it to Android with an instruction to start the activity.

2. **Android checks that the intent is OK, and then tells ReceiveMessageActivity to start.**

3. **When ReceiveMessageActivity starts, it specifies that it uses layout activity_receive_message.xml, and this gets displayed on the device.**
   The activity also updates the layout so that it displays the extra text included in the intent.

Test drive the app

- Create 1st activity
- Create 2nd activity
- Call 2nd activity
- **Pass data**

Make sure you've updated the two activities, save your changes, and then run the app. `CreateMessageActivity` starts, and when you enter some text and then click on the Send Message button, it launches `ReceiveMessageActivity`. The text you entered is displayed in the text view.

We can change the app to send messages to other people

Now that we have an app that sends a message to another activity, we can change it so that it can send messages to other people. We can do this by integrating with the message sending apps already on the device. Depending on what apps the user has, we can get our app to send messages via Gmail, Google+, Facebook, Twitter...

Hey, hold it right there! That sounds like a freaky amount of work to get our app working with all those other apps. And how the heck do I know what apps people have on their devices anyway?

It's not as hard as it sounds thanks to the way Android is designed to work.

Remember right at the beginning of the chapter when we said that tasks are multiple activities chained together? Well, **you're not just limited to using the activities within your app**. You can go beyond the boundaries of your app to use activities within *other* apps as well.

- Create 1st activity
- Create 2nd activity
- Call 2nd activity
- **Pass data**

How Android apps work

As you've seen, all Android apps are composed of one or more activities, along with other components such as layouts. Each activity is a single defined focused thing the user can do. As an example, apps such as Gmail, Google+, Facebook, and Twitter all have activities that enable you to send messages, even though they may achieve this in different ways.

Each app is composed of a bunch of activities. (There are other components too, but we're just focusing on the activities right now.)

Intents can start activities in other apps

You've already seen how you can use an intent to start a second activity within the same app. The first activity passes an intent to Android, Android checks it, and then Android tells the second activity to start.

The same principle applies to activities in other apps. You get an activity in your app to pass an intent to Android, Android checks it, and then Android tells the second activity to start *even though it's in another app*. As an example, we can use an intent to start the activity in Gmail that sends messages, and pass it the text we want to send. That means that instead of writing our own activities to send emails, we can use the existing Gmail app.

You can create an intent to start another activity even if the activity is within another app.

This is the app you've been working on throughout the chapter.

This means that you can build apps that perform powerful tasks by chaining together activities across the device.

But we don't know what apps are on the user's device

There are three questions we need answers to before we can call activities in other apps:

- How do we know which activities are available on the user's device?
- How do we know which of these activities are appropriate for what we want to do?
- How do we know how to use these activities?

The great news is that we can solve all of these problems using **actions**. Actions are a way of telling Android what standard operations activities can perfom. As an example, Android knows that all activities registered for a send action are capable of sending messages.

Let's explore how to create intents that use actions to return a set of activities that you can use in a standard way—for example, to send messages.

Here's what you're going to do

1. **Create an intent that specifies an action.**
   The intent will tell Android you want to use an activity that can send a message. The intent will include the text of the message.

2. **Allow the user to choose which app to use.**
   Chances are, there'll be more than one app on the user's device capable of sending messages, so the user will need to pick one. We want the user to be able to choose one every time they click on the Send Message button.

Create an intent that specifies an action

So far you've seen how to create an intent that launches a specific activity using:

```
Intent intent = new Intent(this, ReceiveMessageActivity.class);
```

We've told the intent which class it's intended for, but what if we don't know?

This intent is an example of an **explicit intent**; you explicitly tell Android which class you want it to run.

If there's an action you want done but you don't care which activity does it, you create an **implicit intent**. You tell Android what sort of action you want it to perform, and you leave the details of which activity performs it to Android.

How to create the intent

You create an intent that specifies an action using the following syntax:

```
Intent intent = new Intent(action);
```

where `action` is the type of activity action you want to perform. Android provides you with a number of standard actions you can use. As an example, you can use `Intent.ACTION_DIAL` to dial a number, `Intent.ACTION_WEB_SEARCH` to perform a web search, and `Intent.ACTION_SEND` to send a message. So, if you want to create an intent that specifies you want to send a message, you use:

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

You can find out more about the sorts of activity actions you can use and the extra information they support in the Android developer reference material: *http://tinyurl.com/n57qb5.*

Adding extra information

Once you've specified the action you want to use, you can add extra information to it. We want to pass some text with the intent that will form the body of the message we're sending. To do this, you use the following lines of code:

```
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, messageText);
```

These attributes relate to Intent.ACTION_SEND. They're not relevant for all actions.

where `messageText` is the text you want to send. This tells Android that you want the activity to be able to handle data with a MIME data-type of `text/plain`, and also tells it what the text is.

You can make extra calls to the `putExtra()` method if there's additional information you want to add. As an example, if you want to specify the subject of the message, you can use:

```
intent.putExtra(Intent.EXTRA_SUBJECT, subject);
```

If subject isn't relevant to a particular app, it will just ignore this information. Any apps that know how to use it will do so.

where `subject` is the subject of the message.

Specify action
Create chooser

Change the intent to use an action

We'll update *CreateMessageActivity.java* so that we create an implicit intent that uses a send action. Make the changes highlighted below, and save your work:

```
package com.hfad.mymessenger;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, messageText);
        startActivity(intent);
    }
}
```

Remove these two lines.

Instead of creating an intent that's explicitly for ReceiveMessageActivity, we're creating an intent that uses a send action.

Now that you've updated your code, let's break down what happens when the user clicks on the Send Message button.

What happens when the code runs

1. **When the onSendMessage() method is called, an intent gets created. The startActivity() method passes this intent to Android.**
   The intent specifies an action of `ACTION_SEND`, and a MIME type of `text/plain`.

2. **Android sees that the intent can only be passed to activities able to handle ACTION_SEND and text/plain data. Android checks all the activities on the user's device, looking for ones that are able to receive the intent.**
   If no actions are able to handle the intent, an `ActivityNotFoundException` is thrown.

3a. **If just one activity is able to receive the intent, Android tells that activity to start and passes it the intent.**

The story continues...

Specify action
Create chooser

3b **If more than one activity is able to receive the intent, Android displays an activity chooser dialog and asks the user which one to use.**

---

4 **When the user chooses the activity she wants to use, Android tells the activity to start and passes it the intent.** The activity displays the extra text contained in the intent in the body of a new message.

In order to pass the intent to an activity, Android must first know which activities are capable of receiving the intent. On the next couple of pages we'll look at how it does this.

Specify action

Create chooser

The intent filter tells Android which activities can handle which actions

When Android is given an intent, it has to figure out which activity, or activities, can handle it. This process is known as **intent resolution**.

When you use an *explicit* intent, intent resolution is straightforward. The intent explicitly says which component the intent is directed at, so Android has clear instructions about what to do. As an example, the following code explicitly tells Android to start `ReceiveMessageActivity`:

```
Intent intent = new Intent(this, ReceiveMessageActivity.class);
startActivity(intent);
```

When you use an *implicit* intent, Android uses the information in the intent to figure out which components are able to receive it. It does this by checking the intent filters in every app's copy of *AndroidManifest.xml*.

An **intent filter** specifies what types of intent each component can receive. As an example, here's the entry for an activity that can handle an action of `ACTION_SEND`. The activity is able to accept data with MIME types of `text/plain` or `image`:

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

This is just an example; there's no activity called "ShareActivity" in our project.

This tells Android the activity can handle ACTION_SEND.

The intent filter must include a category of DEFAULT or it won't be able to receive implicit intents.

These are the types of data the activity can handle.

The intent filter also specifies a **category**. The category supplies extra information about the activity such as whether it can be started by a web browser, or whether it's the main entry point of the app. An intent filter ***must*** include a category of `android.intent.category.DEFAULT` if it's to receive implicit intents. If an activity has no intent filter, or it doesn't include a category name of `android.intent.category.DEFAULT`, it means that the activity can't be started with an implicit intent. It can only be started with an *explicit* intent using the full name (including the package) of the component.

Specify action
Create chooser

How Android uses the intent filter

When you use an implicit intent, Android compares the information given in the intent with the information given in the intent filters specified in every app's *AndroidManifest.xml* file.

Android first considers intent filters that include a category of `android.intent.category.DEFAULT`:

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT"/>
    ...
</intent-filter>
```

Intent filters without this category will be omitted, as they can't receive implicit intents.

Android then matches intents to intent filters by comparing the action and MIME type contained in the intent with those of the intent filters. As an example, if an intent specifies an action of `Intent.ACTION_SEND` using:

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

It will also look at the category of the intent filter if one is supplied by the intent. However, this feature isn't used very often, so we don't cover how to add categories to intents.

Android will only consider activities that specify an intent filter with an action of `android.intent.action.SEND` like this:

```
<intent-filter>
    <action android:name="android.intent.action.SEND"/>
    ...
</intent-filter>
```

Similarly, if the intent MIME type is set to `text/plain` using:

```
intent.setType("text/plain");
```

Android will only consider activities that can accommodate this type of data:

```
<intent-filter>
    <data android:mimeType="text/plain"/>
    ...
</intent-filter>
```

If the MIME type is left out of the intent, Android tries to infer the type based on the data the intent contains.

Once Android has finished comparing the intent to the component's intent filters, it sees how many matches it finds. If Android finds a single match, it starts the component (in our case, the activity) and passes it the intent. If it finds multiple matches, it asks the user to pick one.

BE the Intent

Your job is to play like you're the intent on the right and say which of the activities described below are compatible with your action and data. Say why, or why not, for each one.

Here's the intent.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="*/*"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.MAIN"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.MAIN"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

BE the Intent Solution

Your job is to play like you're the intent on the right and say which of the activities described below are compatible with your action and data. Say why, or why not, for each one.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="*/*"/>
    </intent-filter>
</activity>
```

✓ This activity accepts ACTION_SEND and can handle data of any MIME type, so it can respond to the intent.

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.MAIN"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

✗ This activity doesn't have a category of DEFAULT so can't receive the intent.

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.MAIN"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

✗ This activity can't accept ACTION_SEND intents, only ACTION_SENDTO (which allows you to send a message to someone specified in the intent's data).

You need to run your app on a REAL device

So far we've been running our apps using the emulator. The emulator only includes a small number of apps, and there may well be just one app that can handle `ACTION_SEND`. In order to test our app properly, we need to run it on a physical device where we know there'll be more than one app that can support our action—for example, an app that can send emails and an app that can send text messages.

Here's how you go about getting your app to run on a physical device.

1. Enable USB debugging on your device

On your device, open "Developer options" (in Android 4.0 onward, this is hidden by default). To enable "Developer options," go to Settings → About Phone and tap the build number seven times. When you return to the previous screen, you should now be able to see "Developer options."

Yep, seriously.

Within "Developer options," turn on USB debugging.

You need to enable USB debugging.

2. Set up your system to detect your device

If you're using a Mac, you can skip this step.

If you're using Windows, you need to install a USB driver. You can find the latest instructions here:

http://developer.android.com/tools/extras/oem-usb.html

If you're using Ubuntu Linux, you need to create a `udev` rules file. You can find the latest instructions on how to do this here:

http://developer.android.com/tools/device.html#setting-up

3. Plug your device into your computer with a USB cable

Your device may ask you if you want to accept an RSA key that allows USB debugging with your computer. If it does, you can check the "Always allow from this computer" option and choose OK to enable this.

Specify action
Create chooser

Running your app on a real device (continued)

4. Stop your app running on the emulator

Before you can run your app on a different device, you need to stop it running on the current one (in this case the emulator). To do this, choose Run → "Stop 'app'", or click on the "Stop 'app'" button in the toolbar.

Click on this button in Android Studio's toolbar to stop the app running on the current device.

5. Run your app on the physical device

Run the app by choosing Run → "Run 'app'". Android Studio will ask you to choose which device you want to run your app on, so select your device from the list of available devices and click OK. Android Studio will install the app on your device and launch it.

And here's the app running on the physical device

You should find that your app looks about the same as when you ran it through the emulator. You'll probably find that your app installs and runs quicker too.

Now that you know how to run the apps you create on your own device, you're all set to test the latest changes to your app.

Test drive the app

Try running the app using the emulator, and then using your own device. The results you get will depend on how many activities you have on each that support using the Send action with text data.

If you have one activity

Clicking on the Send Message button will take you straight to that app.

We only have one activity available on the emulator that can send messages with text data, so when we click on the Send Message button, Android starts that activity.

If you have more than one activity

Android displays a chooser and asks you to pick which one you want to use. It also asks you whether you want to use this activity just once or always. If you choose always, the next time you click on the Send Message button it uses the same activity by default.

We have lots of suitable activities available on our physical device. We decided to use the built-in Messages app. We selected the "always" option—great if we always want to use the same app, not so great if we want to use a different one each time.

What if you ALWAYS want your users to choose an activity?

You've just seen that if there's more than one activity on your device that's capable of receiving your intent, Android automatically asks you to choose which activity you want to use. It even asks you whether you want to use this activity all the time or just on this occasion.

There's just one problem with this default behavior: what if you want to *guarantee* that users can choose an activity every time they click on the Send Message button? If they've chosen the option to always use Gmail, for instance, they won't be asked if they want to use Twitter next time.

Fortunately, there's a way around this. You can create a chooser that asks users to pick an activity without giving them the option to always use that activity.

createChooser() allows you to specify a title for the chooser dialog, and doesn't give the user the option of selecting an activity to use by default. It also lets the user know if there are no matching activities by displaying a message.

Intent.createChooser() displays a chooser dialog

You can achieve this using the `Intent.createChooser()` method, which takes the intent you've already created and wraps it in a chooser dialog. When you use this method, the user isn't given the option of choosing a default activity—they get asked to choose one every time.

You call the `createChooser()` method like this:

This is the intent you created earlier.

```
Intent chosenIntent = Intent.createChooser(intent, "Send message via...");
```

You can pass in a title for the chooser that gets displayed at the top of the screen.

The method takes two parameters: an intent and an optional `String` title for the chooser dialog window. The `Intent` parameter needs to describe the types of activity you want the chooser to display. You can use the same intent we created earlier, as this specifies that we want to use `ACTION_SEND` with textual data.

The `createChooser()` method returns a brand-new `Intent`. This is a new explicit intent that's targeted at the activity chosen by the user. It includes any extra information supplied by the original intent, including any text.

To start the activity the user chose, you need to call:

```
startActivity(chosenIntent);
```

Over the next couple of pages we'll take a closer look at what happens when you call the `createChooser()` method.

What happens when you call createChooser()

Here's what happens when you run the following two lines of code:

```
Intent chosenIntent = Intent.createChooser(intent, "Send message via...");
startActivity(chosenIntent);
```

1. **The createChooser() method gets called.**
   The method includes an intent that specifies the action and MIME type that's required.

2. **Android checks which activities are able to receive the intent by looking at their intent filters.**
   It matches on the actions, type of data, and categories they can support.

3. **If more than one activity is able to receive the intent, Android displays an activity chooser dialog and asks the user which one to use.**
   It doesn't give the user the option of always using a particular activity, and it displays "Send message via..." in the title.

   If no activities are found, Android still displays the chooser but shows a message telling the user there are no apps that can perform the action.

The story continues...

Specify action
Create chooser

4 **When the user chooses which activity she wants to use, Android returns a new explicit intent describing the chosen activity.**
The new intent includes any extra information that was included in the original intent, such as any text.

5 **The activity asks Android to start the activity specified in the intent.**

6 **Android starts the activity specified by the intent, and then passes it the intent.**

Change the code to create a chooser

Let's change the code so that the user gets asked which activity they want to use to send a message every time they click on the Send Message button. We'll add a String resource to *strings.xml* for the chooser dialog title, and we'll update the `onSendMessage()` method in *CreateMessageActivity.java* so that it calls the `createChooser()` method.

Update strings.xml...

We want the chooser dialog to have a title of "Send message via...". Add a String called `"chooser"` to *strings.xml*, and give it the value `Send message via...` (make sure to save your changes):

```
...
<string name="chooser">Send message via...</string>
...
```

This will be displayed in the chooser dialog.

...and update the onSendMessage() method

We need to change the `onSendMessage()` method so that it retrieves the value of the chooser String resource in *strings.xml*, calls the `createChooser()` method, and then starts the activity the user chooses. Update your code as follows:

```
...
    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, messageText);
        String chooserTitle = getString(R.string.chooser);
        Intent chosenIntent = Intent.createChooser(intent, chooserTitle);
        startActivity(intent);
        startActivity(chosenIntent);
    }
...
```

Get the chooser title.

Display the chooser dialog.

Delete this line.

Start the activity that the user selected.

The `getString()` method is used to get the value of a String resource. It takes one parameter, the ID of the resource (in our case, this is `R.string.chooser`):

```
getString(R.string.chooser);
```

If you look in R.java, you'll find chooser in the inner class called string.

Now that we've updated the app, let's run the app to see our chooser in action.

Test drive the app

Save your changes, then try running the app again on the device or the emulator.

If you have one activity

Clicking on the Send Message button will take you straight to that activity just like before.

There's no change here—Android continues to take you straight to the activity.

If you have more than one activity

Android displays a chooser, but this time it doesn't ask us if we always want to use the same activity. It also displays the value of the chooser String resource in the title.

Here's the chooser we created with createChooser(). It no longer gives us the option of using a particular activity every time.

If you have NO matching activities

If you have no activities on your device that are capable of sending messages, the `createChooser()` method lets you know by displaying a message.

If you want to replicate this for yourself, try running the app in the emulator, and disable the built-in Messenger app that's on there.

there are no Dumb Questions

Q: So I can run my apps in the emulator or on a physical device. Which is best?

A: Each one has its pros and cons.

If you run apps on your physical device, they tend to load a lot quicker than using the emulator. This approach is also useful if you're writing code that interacts with the device hardware.

The emulator allows you to run apps against many different versions of Android, screen resolutions, and device specifications. It saves you from buying lots of different devices. The key thing is to make sure you test your apps thoroughly using a mixture of the emulator and physical devices before releasing them to a wider audience.

Q: Should I use implicit or explicit intents?

A: It comes down to whether you need Android to use a specific activity to perform your action, or whether you just want the action done. As an example, suppose you wanted to send an email. If you don't care which email app the user uses to send it, just as long as the email gets sent, you'd use an implicit intent. On the other hand, if you needed to pass an intent to a particular activity in your app, you'd need to use an explicit intent to explicitly say which activity needs to receive the intent.

Q: You mentioned that an activity's intent filter can specify a category as well as an action. What's the difference between the two?

A: An action specifies what an activity can do, and the category gives extra detail. We've not gone into details about adding categories because you don't often need to specify a category when you create an intent.

Q: You say that the `createChooser()` method displays a message in the chooser if there are no activities that can handle the intent. What if I'd instead used the default Android chooser and passed an implicit intent to `startActivity()`?

A: If the `startActivity()` method is given an intent where there are no matching activities, an `ActivityNotFoundException` is thrown. You can check whether any activities on the device are able to receive the intent by calling the intent's `resolveActivity()` method and checking its return value. If its return value is null, no activities on the device are able to receive the intent, so you shouldn't call `startActivity()`.

Your Android Toolbox

You've got Chapter 3 under your belt and now you've added multi-activity apps and intents to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- A **task** is two or more activities chained together.
- The `<EditText>` element defines an editable text field for entering text. It inherits from the Android `View` class.
- You can add a new activity in Android Studio by choosing File → New... → Activity.
- Each activity you create must have an entry in *AndroidManifest.xml*.
- An **intent** is a type of message that Android components use to communicate with one another.
- An explicit intent specifies the component the intent is targeted at. You create an explicit intent using `Intent intent = new Intent(this, Target.class);`.
- To start an activity, call `startActivity(intent)`. If no activities are found, it throws an `ActivityNotFoundException`.
- Use the `putExtra()` method to add extra information to an intent.
- Use the `getIntent()` method to retrieve the intent that started the activity.
- Use the `get*Extra()` methods to retrieve extra information associated with the intent. `getStringExtra()` retrieves a String, `getIntExtra()` retrieves an int, and so on.
- An activity action describes a standard operational action an activity can perform. For example, to send a message, use `Intent.ACTION_SEND`.
- To create an implicit intent that specifies an action, use `Intent intent = new Intent(action);`.
- To describe the type of data in an intent, use the `setType()` method.
- Android resolves intents based on the named component, action, type of data, and categories specified in the intent. It compares the contents of the intent with the intent filters in each app's *AndroidManifest.xml*. An activity must have a category of `DEFAULT` if it is to receive an implicit intent.
- The `createChooser()` method allows you to override the default Android activity chooser dialog. It lets you specify a title for the dialog, and doesn't give the user the option of setting a default activity. If no activities can receive the intent it is passed, it displays a message. The `createChooser()` method returns an `Intent`.
- You retrieve the value of a String resource using `getString(R.string.stringname);`.

4 the activity lifecycle

Being an Activity

Activities form the foundation of every Android app.

So far you've seen how to create activities, and made one activity start another using an intent. But ***what's really going on beneath the hood?*** In this chapter, we're going to dig a little deeper into **the activity lifecycle**. What happens when an activity is **created** and **destroyed**? Which methods get called when an activity is **made visible and appears in the foreground**, and which get called when the activity **loses the focus and is hidden**? And **how do you save and restore your activity's state**? Read on to find out.

How do activities really work?

So far you've seen how to create apps that interact with the user, and apps that use multiple activities to perform tasks. Now that you have these core skills under your belt, it's time to take a deeper look at how activities *actually work*. Here's a recap of what you know so far, with a few extra details thrown in.

An app is a collection of activities, layouts, and other resources.
One of these activities is the main activity for the app.

By default, each app runs within its own process.
This helps keep your apps safe and secure. You can read more about this in Appendix III (which covers the Android runtime, a.k.a. ART) at the back of this book.

Your app can start an activity in another application by passing an intent with startActivity().
The Android system knows about all the device's installed apps and their activities, and uses the intent to start the correct activity.

When an activity needs to start, Android checks whether there's already a process for that app.
If one exists, Android runs the activity in that process. If one doesn't exist, Android creates one.

When Android starts an activity, it calls its onCreate() method.
`onCreate()` is always run whenever an activity gets created.

But there are still lots of things we don't yet know about how activities function. How long does an activity live for? What happens when your activity disappears from the screen? Is it still running? Is it still in memory? And what happens if your app gets interrupted by an incoming phone call? We want to be able to control the behavior of our activities in a *whole range of different circumstances*, but how?

The Stopwatch app

In this chapter, we're going to take a closer look at how activities work under the hood, common ways in which your apps can break, and how you can fix them using the activity lifecycle methods. We're going to explore the lifecycle methods using a simple Stopwatch app as an example.

The Stopwatch app consists of a single activity and a single layout. The layout includes a text view showing you how much time has passed, a Start button that starts the stopwatch, a Stop button that stops it, and a Reset button that resets the timer value to 0.

Create a new project for the Stopwatch app

You have enough experience under your belt to build the app without much guidance from us. We're going to give you just enough code so you can build the app yourself, and then you can see what happens when you try to run it.

Start off by creating a new Android project for an application named "Stopwatch" with a company domain of "hfad.com", making the package name `com.hfad.stopwatch`. The minimum SDK should be API 19 so it can run on most devices. You'll need an empty activity called "StopwatchActivity" and a layout called "activity_stopwatch". **Make sure you uncheck the Backwards Compatibility (AppCompat) checkbox.**

Add String resources

We're going to use three String values in our stopwatch layout, one for the text value of each button. These values are String resources, so they need to be added to *strings.xml*. Add the String values below to your version of *strings.xml*:

```
...
    <string name="start">Start</string>
    <string name="stop">Stop</string>
    <string name="reset">Reset</string>
...
```

We'll use these String resources in our layout.

Next, let's update the code for our layout.

Update the stopwatch layout code

Here's the XML for the layout. It describes a single text view that's used to display the timer, and three buttons to control the stopwatch. Replace the XML currently in *activity_stopwatch.xml* with the XML shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".StopwatchActivity">

    <TextView
        android:id="@+id/time_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textAppearance="@android:style/TextAppearance.Large"
        android:textSize="56sp" />
```

We'll use this text view to display the number of seconds.

These attributes make the stopwatch timer nice and big.

The layout code continues over the page.

The layout code (continued)

```
    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="8dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />

    <Button
        android:id="@+id/reset_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="8dp"
        android:onClick="onClickReset"
        android:text="@string/reset" />
</LinearLayout>
```

This code is for the Start button.

When it gets clicked, the Start button calls the onClickStart() method.

This is for the Stop button.

When it gets clicked, the Stop button calls the onClickStop() method.

This is for the Reset button.

When it gets clicked, the Reset button calls the onClickReset() method.

The layout is now done! Next, let's move on to the activity.

Do this!

Make sure you update the layout and *strings.xml* in your app before continuing.

How the activity code will work

The layout defines three buttons that we'll use to control the stopwatch. Each button uses its `onClick` attribute to specify which method in the activity should run when the button is clicked. When the Start button is clicked, the `onClickStart()` method gets called, when the Stop button is clicked the `onClickStop()` method gets called, and when the Reset button is clicked the `onClickReset()` method gets called. We'll use these methods to start, stop, and reset the stopwatch.

When you click this button, the onClickStart() method is called.

When you click this button, the onClickStop() method is called.

When you click this button, the onClickReset() method is called.

We'll update the stopwatch using a method we'll create called `runTimer()`. The `runTimer()` method will run code every second to check whether the stopwatch is running, and, if it is, increment the number of seconds and display the number of seconds in the text view.

To help us with this, we'll use two private variables to record the state of the stopwatch. We'll use an `int` called `seconds` to track how many seconds have passed since the stopwatch started running, and a `boolean` called `running` to record whether the stopwatch is currently running.

We'll start by writing the code for the buttons, and then we'll look at the `runTimer()` method.

Add code for the buttons

When the user clicks on the Start button, we'll set the `running` variable to `true` so that the stopwatch will start. When the user clicks on the Stop button, we'll set `running` to `false` so that the stopwatch stops running. If the user clicks on the Reset button, we'll set `running` to `false` and `seconds` to 0 so that the stopwatch is reset and stops running.

To do all that, replace the contents of *StopwatchActivity.java* with the code below:

```
package com.hfad.stopwatch;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class StopwatchActivity extends Activity {

    private int seconds = 0;
    private boolean running;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
    }


    //Start the stopwatch running when the Start button is clicked.
    public void onClickStart(View view) {
        running = true;
    }

    //Stop the stopwatch running when the Stop button is clicked.
    public void onClickStop(View view) {
        running = false;
    }

    //Reset the stopwatch when the Reset button is clicked.
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }
}
```

Make sure your activity extends the Activity class.

Use the seconds and running variables to record the number of seconds passed and whether the stopwatch is running.

Start the stopwatch.

This gets called when the Start button is clicked.

Stop the stopwatch.

This gets called when the Stop button is clicked.

Stop the stopwatch and set the seconds to 0.

This gets called when the Reset button is clicked.

The runTimer() method

The next thing we need to do is create the `runTimer()` method. This method will get a reference to the text view in the layout; format the contents of the `seconds` variable into hours, minutes, and seconds; and then display the results in the text view. If the `running` variable is set to `true`, it will increment the `seconds` variable.

The code for the `runTimer()` method is below. We'll add it to *StopwatchActivity.java* in a few pages:

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    ...
    int hours = seconds/3600;
    int minutes = (seconds%3600)/60;
    int secs = seconds%60;
    String time = String.format(Locale.getDefault(),
                "%d:%02d:%02d", hours, minutes, secs);
    timeView.setText(time);
    if (running) {
        seconds++;
    }
    ...
}
```

Get the text view.

We've left out a bit of code here. We'll look at it on the next page.

Format the seconds into hours, minutes, and seconds. This is plain Java code.

Set the text view text.

If running is true, increment the seconds variable.

We need this code to keep looping so that it increments the `seconds` variable and updates the text view every second. We need to do this in such a way that we don't block the main Android thread.

In non-Android Java programs, you can perform tasks like this using a background thread. In Androidville, that approach won't work—only the main Android thread can update the user interface, and if any other thread tries to do so, you get a `CalledFromWrongThreadException`.

The solution is to use a `Handler`. We'll look at this technique on the next page.

Handlers allow you to schedule code

A `Handler` is an Android class you can use to schedule code that should be run at some point in the future. You can also use it to post code that needs to run on a different thread than the main Android thread. In our case, we're going to use a `Handler` to schedule the stopwatch code to run every second.

To use the `Handler`, you wrap the code you wish to schedule in a `Runnable` object, and then use the `Handler` `post()` and `postDelayed()` methods to specify when you want the code to run. Let's take a closer look at these mehods.

The post() method

The `post()` method posts code that needs to be run as soon as possible (which is usually almost immediately). This method takes one parameter, an object of type `Runnable`. A `Runnable` object in Androidville is just like a `Runnable` in plain old Java: a job you want to run. You put the code you want to run in the `Runnable`'s `run()` method, and the `Handler` will make sure the code is run as soon as possible. Here's what the method looks like:

```
final Handler handler = new Handler();
handler.post(Runnable);
```

← You put the code you want to run in the Runnable's run() method.

The postDelayed() method

The `postDelayed()` method works in a similar way to the `post()` method except that you use it to post code that should be run in the future. The `postDelayed()` method takes two parameters: a `Runnable` and a `long`. The `Runnable` contains the code you want to run in its `run()` method, and the `long` specifies the number of milliseconds you wish to delay the code by. The code will run as soon as possible after the delay. Here's what the method looks like:

```
final Handler handler = new Handler();
handler.postDelayed(Runnable, long);
```

← Use this method to delay running code by a specified number of milliseconds.

On the next page, we'll use these methods to update the stopwatch every second.

The full runTimer() code

To update the stopwatch, we're going to repeatedly schedule code using the `Handler` with a delay of 1 second each time. Each time the code runs, we'll increment the `seconds` variable and update the text view.

Here's the full code for the `runTimer()` method, which we'll add to *StopwatchActivity.java* in a couple of pages:

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds/3600;
            int minutes = (seconds%3600)/60;
            int secs = seconds%60;
            String time = String.format(Locale.getDefault(),
                        "%d:%02d:%02d", hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
```

Create a new Handler.

Call the post() method, passing in a new Runnable. The post() method processes code without a delay, so the code in the Runnable will run almost immediately.

The Runnable run() method contains the code you want to run—in our case, the code to update the text view.

Post the code in the Runnable to be run again after a delay of 1,000 milliseconds. As this line of code is included in the Runnable run() method, it will keep getting called.

Using the `post()` and `postDelayed()` methods in this way means that the code will run as soon as possible after the required delay, which in practice means almost immediately. While this means the code will lag slightly over time, it's accurate enough for the purposes of exploring the lifecycle methods in this chapter.

We want the `runTimer()` method to start running when `StopwatchActivity` gets created, so we'll call it in the activity `onCreate()` method:

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    runTimer();
}
```

We'll show you the full code for `StopwatchActivity` on the next page.

The full StopwatchActivity code

Here's the full code for *StopwatchActivity.java*. Update your code to match our changes below.

```
package com.hfad.stopwatch;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import java.util.Locale;
import android.os.Handler;
import android.widget.TextView;

public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        runTimer();
    }

    //Start the stopwatch running when the Start button is clicked.
    public void onClickStart(View view) {
        running = true;
    }

    //Stop the stopwatch running when the Stop button is clicked.
    public void onClickStop(View view) {
        running = false;
    }
```

We're using these extra classes so we need to import them.

Use the seconds and running variables to record the number of seconds passed and whether the stopwatch is running.

We're using a separate method to update the stopwatch. We're starting it when the activity is created.

This gets called when the Start button is clicked.

Start the stopwatch.

This gets called when the Stop button is clicked.

Stop the stopwatch.

The activity code (continued)

```
    //Reset the stopwatch when the Reset button is clicked.
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }

    //Sets the number of seconds on the timer.
    private void runTimer() {
        final TextView timeView = (TextView)findViewById(R.id.time_view);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                int hours = seconds/3600;
                int minutes = (seconds%3600)/60;
                int secs = seconds%60;
                String time = String.format(Locale.getDefault(),
                            "%d:%02d:%02d", hours, minutes, secs);
                timeView.setText(time);
                if (running) {
                    seconds++;
                }
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

This gets called when the Reset button is clicked.

Stop the stopwatch and set the seconds to 0.

Get the text view.

Use a Handler to post code.

Format the seconds into hours, minutes, and seconds.

Set the text view text.

If running is true, increment the seconds variable.

Post the code again with a delay of 1 second.

Let's look at what happens when the code runs.

Do this!

Make sure you update your activity code to reflect these changes.

What happens when you run the app

1. **The user decides she wants to run the app.**
On her device, she clicks on the app's icon.

2. **An intent is constructed to start this activity using startActivity(intent).**
The *AndroidManifest.xml* file for the app specifies which activity to use as the launch activity.

3. **Android checks to see whether there's already a process running for the app, and if not, creates a new process.**
It then creates a new activity object—in this case, for `StopwatchActivity`.

The story continues

4. **The onCreate() method in the activity gets called.**
The method includes a call to `setContentView()`, specifying a layout, and then starts the stopwatch with `runTimer()`.

5. **When the onCreate() method finishes, the layout gets displayed on the device.**
The `runTimer()` method uses the `seconds` variable to determine what text to display in the text view, and uses the `running` variable to determine whether to increment the number of seconds. As `running` is initially `false`, the number of seconds isn't incremented.

there are no Dumb Questions

Q: Why does Android run each app inside a separate process?

A: For security and stability. This approach prevents one app from accessing the data of another. It also means if one app crashes, it won't take others down with it.

Q: Why have an `onCreate()` method in our activity? Why not just put that code inside a constructor?

A: Android needs to set up the environment for the activity after it's constructed. Once the environment is ready, Android calls `onCreate()`. That's why code to set up the screen goes inside `onCreate()` instead of a constructor.

Q: Couldn't I just write a loop in `onCreate()` to keep updating the timer?

A: No, `onCreate()` needs to finish before the screen will appear. An endless loop would prevent that from happening.

Q: `runTimer()` looks really complicated. Do I really need to do all this?

A: It's a little complex, but whenever you need to post code that runs in a loop, the code will look similar to `runTimer()`.

Test drive the app

When we run the app in the emulator, the app works great. We can start, stop, and reset the stopwatch without any problems at all—the app works just as you'd expect.

These buttons work as you'd expect. The Start button starts the stopwatch, the Stop button stops it, and the Reset button sets the stopwatch back to 0.

But there's just one problem...

When we ran the app on a physical device, the app worked OK until someone rotated the device. When the device was rotated, the stopwatch set itself back to 0.

The stopwatch was running, but reset to 0 when the device was rotated.

In Androidville, it's surprisingly common for apps to break when you rotate the device. Before we fix the problem, let's take a closer look at what caused it.

What just happened?

So why did the app break when the user rotated the screen?
Let's take a closer look at what really happened.

1. **The user starts the app, and clicks on the Start button to set the stopwatch going.**
The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view using the `seconds` and `running` variables.

2. **The user rotates the device.**
Android sees that the screen orientation and screen size has changed, and it destroys the activity, including any variables used by the `runTimer()` method.

3. **StopwatchActivity is then recreated.**
The `onCreate()` method runs again, and the `runTimer()` method gets called. As the activity has been recreated, the `seconds` and `running` variables are set to their default values.

Rotating the screen changes the device configuration

When Android runs your app and starts an activity, it takes into account the **device configuration**. By this we mean the configuration of the physical device (such as the screen size, screen orientation, and whether there's a keyboard attached) and also configuration options specified by the user (such as the locale).

Android needs to know what the device configuration is when it starts an activity because the configuration can impact what resources are needed for the application. A different layout might need to be used if the device screen is landscape rather than portrait, for instance, and a different set of String values might need to be used if the locale is France.

Android apps can contain multiple resource files in the app/src/main/res folder. For instance, if the device locale is set to France, Android will use the strings.xml file in the values-fr folder.

The device configuration includes options specified by the user (such as the locale), and options relating to the physical device (such as the orientation and screen size). A change to any of these options results in the activity being destroyed and then recreated.

When the device configuration changes, anything that displays a user interface needs to be updated to match the new configuration. If you rotate your device, Android spots that the screen orientation and screen size have changed, and classes this as a change to the device configuration. It destroys the current activity, and then recreates it so that resources appropriate to the new configuration get picked up.

The states of an activity

When Android creates and destroys an activity, the activity moves from being launched to running to being destroyed.

The main state of an activity is when it's ***running*** or ***active***. An activity is running when it's in the foreground of the screen, it has the focus, and the user can interact with it. The activity spends most of its life in this state. An activity starts running after it has been launched, and at the end of its life, the activity is ***destroyed***.

An activity is running when it's in the foreground of the screen.

When an activity moves from being launched to being destroyed, it triggers key activity lifecycle methods: the `onCreate()` and `onDestroy()` methods. These are lifecycle methods that your activity inherits, and which you can override if necessary.

The `onCreate()` method gets called immediately after your activity is launched. This method is where you do all your normal activity setup such as calling `setContentView()`. You should always override this method. If you *don't* override it, you won't be able to tell Android what layout your activity should use.

onCreate() gets called when the activity is first created, and it's where you do your normal activity setup.

The `onDestroy()` method is the final call you get before the activity is destroyed. There are a number of situations in which an activity can get destroyed—for example, if it's been told to finish, if the activity is being recreated due to a change in device configuration, or if Android has decided to destroy the activity in order to save space.

onDestroy() gets called just before your activity gets destroyed.

We'll take a closer look at how these methods fit into the activity states on the next page.

The activity lifecycle: from create to destroy

Here's an overview of the activity lifecycle from birth to death. As you'll see later in the chapter, we've left out some of the details, but at this point we're just focusing on the `onCreate()` and `onDestroy()` methods.

1. **The activity gets launched.**
   The activity object is created and its constructor is run.

2. **The onCreate() method runs immediately after the activity is launched.**
   The `onCreate()` method is where any initialization code should go, as this method always gets called after the activity has launched but before it starts running.

3. **An activity is running when it's visible in the foreground and the user can interact with it.**
   This is where an activity spends most of its life.

4. **The onDestroy() method runs immediately before the activity is destroyed.**
   The `onDestroy()` method enables you to perform any final cleanup such as freeing up resources.

5. **After the onDestroy() method has run, the activity is destroyed.**
   The activity ceases to exist.

If your device is extremely low on memory, onDestroy() might not get called before the activity is destroyed.

The `onCreate()` and `onDestroy()` methods are two of the activity lifecycle methods. So where do these methods come from?

Your activity inherits the lifecycle methods

As you saw earlier in the book, your activity extends the `android.app.Activity` class. It's this class that gives your activity access to the Android lifecycle methods. Here's a diagram showing the class hierarchy:

Context abstract class
(android.content.Context)
An interface to global information about the application environment. Allows access to application resources, classes, and operations.

ContextWrapper class
(android.content.ContextWrapper)
A proxy implementation for the `Context`.

ContextThemeWrapper class
(android.view.ContextThemeWrapper)
Allows you to modify the theme from what's in the `ContextWrapper`.

Activity class
(android.app.Activity)
The `Activity` class implements default versions of the lifecycle methods. It also defines methods such as `findViewById(Int)` and `setContentView(View)`.

These are the activity lifecycle methods. You'll find out more about them through the rest of the chapter.

These aren't lifecycle methods, but they're still very useful. You've already used most of them in earlier chapters.

YourActivity class
(com.hfad.foo)
Most of the behavior of your activity is handled by superclass methods your activity inherits. All you do is override the methods you need.

Now that you know more about the activity lifecycle methods, let's see how you deal with device configuration changes.

Save the current state...

As you saw, our app went wrong when the user rotated the screen. The activity was destroyed and recreated, which meant that local variables used by the activity were lost. So how do we get around this issue?

The best way of dealing with configuration changes is to save the current state of the activity, and then reinstate it in the `onCreate()` method of the activity.

To save the activity's current state, you need to implement the `onSaveInstanceState()` method. This method gets called before the activity gets destroyed, which means you get an opportunity to save any values you want to retain before they get lost.

The `onSaveInstanceState()` method takes one parameter, a `Bundle`. A `Bundle` allows you to gather together different types of data into a single object:

```
public void onSaveInstanceState(Bundle savedInstanceState) {
}
```

The onSaveInstanceState() method gets called before onDestroy(). It gives you a chance to save your activity's state before the activity is destroyed.

The `onCreate()` method gets passed the `Bundle` as a parameter. This means that if you add the values of the `running` and `seconds` variables to the `Bundle`, the `onCreate()` method will be able to pick them up when the activity gets recreated. To do this, you use `Bundle` methods to add name/value pairs to the `Bundle`. These methods take the form:

```
bundle.put*("name", value)
```

where `bundle` is the name of the `Bundle`, `*` is the type of value you want to save, and `name` and `value` are the name and value of the data. As an example, to add the `seconds` `int` value to the `Bundle`, you'd use:

```
bundle.putInt("seconds", seconds);
```

You can save multiple name/value pairs of data to the `Bundle`.

Here's our `onSaveInstanceState()` method in full (we'll add it to *StopwatchActivity.java* a couple of pages ahead):

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
}
```

Save the values of the seconds and running variables to the Bundle.

Once you've saved variable values to the `Bundle`, you can use them in our `onCreate()` method.

...then restore the state in onCreate()

As we said earlier, the `onCreate()` method takes one parameter, a `Bundle`. If the activity's being created from scratch, this parameter will be null. If, however, the activity's being recreated and there's been a prior call to `onSaveInstanceState()`, the `Bundle` object used by `onSaveInstanceState()` will get passed to the activity:

```
protected void onCreate(Bundle savedInstanceState) {
    ...
}
```

You can get values from `Bundle` by using methods of the form:

```
bundle.get*("name");
```

Instead of *, use Int, String, and so on, to specify the type of data you want to get.

where `bundle` is the name of the `Bundle`, `*` is the type of value you want to get, and `name` is the name of the name/value pair you specified on the previous page. As an example, to get the `seconds` value from the `Bundle`, you'd use:

```
int seconds = bundle.getInt("seconds");
```

Putting all of this together, here's what our `onCreate()` method now looks like (we'll add this to *StopwatchActivity.java* on the next page):

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_stopwatch);
    if (savedInstanceState != null) {
        seconds = savedInstanceState.getInt("seconds");
        running = savedInstanceState.getBoolean("running");
    }
    runTimer();
}
```

Retrieve the values of the seconds and running variables from the Bundle.

We'll look at the full code to save and restore `StopwatchActivity`'s state on the next page.

The updated StopwatchActivity code

We've updated our `StopwatchActivity` code so that if the user rotates the device, its state gets saved via the `onSaveInstanceState()` method, and restored via the `onCreate()` method. Update your version of *StopwatchActivity.java* to include our changes (below in bold):

```
...

public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
        }
        runTimer();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
    }
...
```

Restore the activity's state by getting values from the Bundle.

Save the state of the variables in the activity's onSaveInstanceState() method.

We've left out some of the activity code, as we don't need to change it.

So how does this work in practice?

What happens when you run the app

1. **The user starts the app, and clicks on the Start button to set the stopwatch going.**
   The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view.

2. **The user rotates the device.**
   Android views this as a configuration change, and gets ready to destroy the activity. Before the activity is destroyed, `onSaveInstanceState()` gets called. The `onSaveInstanceState()` method saves the `seconds` and `running` values to a `Bundle`.

The story continues

3. **Android destroys the activity, and then recreates it.**
The `onCreate()` method gets called, and the `Bundle` gets passed to it.

4. **The Bundle contains the values of the seconds and running variables as they were before the activity was destroyed.**
Code in the `onCreate()` method sets the current variables to the values in the `Bundle`.

5. **The runTimer() method gets called, and the timer picks up where it left off.**
The running stopwatch gets displayed on the device and continues to increment.

Test drive the app

Make the changes to your activity code, then run the app. When you click on the Start button, the timer starts, and it continues when you rotate the device.

there are no Dumb Questions

Q: Why does Android want to recreate an activity just because I rotated the screen?

A: The `onCreate()` method is normally used to set up the screen. If your code in `onCreate()` depended upon the screen configuration (for example, if you had different layouts for landscape and portrait), then you would want `onCreate()` to be called every time the configuration changed. Also, if the user changed their locale, you might want to recreate the UI in the local language.

Q: Why doesn't Android automatically store every instance variable automatically? Why do I have to write all of that code myself?

A: You might not want every instance variable stored. For example, you might have a variable that stores the current screen width. You would want that variable to be recalculated the next time `onCreate()` is called.

Q: Is a `Bundle` some sort of Java map?

A: No, but it's designed to work like a `java.util.Map`. A `Bundle` has additional abilities compared to a `Map`. `Bundles` can be sent between processes, for example. That's really useful, because it allows the Android OS to stay in touch with the state of an activity.

There's more to an activity's life than create and destroy

So far we've looked at the create and destroy parts of the activity lifecycle (and a little bit in between), and you've seen how to deal with configuration changes such as screen orientation. But there are other events in an activity's life that you might want to deal with to get the app to behave in the way you want.

As an example, suppose the stopwatch is running and you get a phone call. Even though the stopwatch isn't visible, it will continue running. But what if you want the stopwatch to stop while it's hidden, and resume once the app is visible again?

Even if you don't really want your stopwatch to behave like this, just play along with us. It's a great excuse to look at more lifecycle methods.

Start, stop, and restart

Fortunately, it's easy to handle actions that relate to an activity's visibility if you use the right lifecycle methods. In addition to the `onCreate()` and `onDestroy()` methods, which deal with the overall lifecycle of the activity, there are other lifecycle methods that deal with an activity's visibility.

Specifically, there are three key lifecycle methods that deal with when an activity becomes visible or invisible to the user: `onStart()`, `onStop()`, and `onRestart()`. Just as with `onCreate()` and `onDestroy()`, your activity inherits them from the Android `Activity` class.

`onStart()` gets called when your activity becomes visible to the user.

`onStop()` gets called when your activity has stopped being visible to the user. This might be because it's completely hidden by another activity that's appeared on top of it, or because the activity is going to be destroyed. If `onStop()` is called because the activity's going to be destroyed, `onSaveInstanceState()` gets called before `onStop()`.

`onRestart()` gets called after your activity has been made invisible, before it gets made visible again.

We'll take a closer look at how these fit in with the `onCreate()` and `onDestroy()` methods on the next page.

An activity has a state of stopped if it's completely hidden by another activity and isn't visible to the user. The activity still exists in the background and maintains all state information.

The activity lifecycle: the visible lifetime

Let's build on the lifecycle diagram you saw earlier in the chapter, this time including the `onStart()`, `onStop()`, and `onRestart()` methods (the bits you need to focus on are in bold):

1. **The activity gets launched, and the onCreate() method runs.**
   Any activity initialization code in the `onCreate()` method runs. At this point, the activity isn't yet visible, as no call to `onStart()` has been made.

2. **The onStart() method runs. It gets called when the activity is about to become visible.**
   After the `onStart()` method has run, the user can see the activity on the screen.

3. **The onStop() method runs when the activity stops being visible to the user.**
   After the `onStop()` method has run, the activity is no longer visible.

4. **If the activity becomes visible to the user again, the onRestart() method gets called followed by onStart().**
   The activity may go through this cycle many times if the activity repeatedly becomes invisible and then visible again.

5. **Finally, the activity is destroyed.**
   The `onStop()` method will get called before `onDestroy()`.

We need to implement two more lifecycle methods

There are two things we need to do to update our Stopwatch app. First, we need to implement the activity's `onStop()` method so that the stopwatch stops running when the app isn't visible. Once we've done that, we need to implement the `onStart()` method so that the stopwatch starts again when the app is visible. Let's start with the `onStop()` method.

Implement onStop() to stop the timer

You override the `onStop()` method in the Android `Activity` class by adding the following method to your activity:

```
@Override
protected void onStop() {
    super.onStop();
    ...
}
```

This calls the onStop() method in the activity's superclass, android.app.Activity.

The line of code:

```
super.onStop();
```

calls the `onStop()` method in the `Activity` superclass. You need to add this line of code whenever you override the `onStop()` method to make sure that the activity gets to perform any other actions in the superclass `onStop()` method. If you bypass this step, Android will generate an exception. This applies to all of the lifecycle methods. If you override any of the `Activity` lifecycle methods in your activity, you must call the superclass method or Android will give you an exception.

When you override any activity lifecycle method in your activity, you need to call the Activity superclass method. If you don't, you'll get an exception.

We need to get the stopwatch to stop when the `onStop()` method is called. To do this, we need to set the value of the `running` boolean to `false`. Here's the complete method:

```
@Override
protected void onStop() {
    super.onStop();
    running = false;
}
```

So now the stopwatch stops when the activity is no longer visible. The next thing we need to do is get the stopwatch to start again when the activity becomes visible.

Sharpen your pencil

Now it's your turn. Change the activity code so that if the stopwatch was running before **onStop()** was called, it starts running again when the activity regains the focus. Hint: you may need to add a new variable.

```
public class StopwatchActivity extends Activity {
    private int seconds = 0;
    private boolean running;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
        }
        runTimer();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
    }

    @Override
    protected void onStop() {
        super.onStop();
        running = false;
    }
```

Here's the first part of the activity code. You'll need to implement the onStart() method and change other methods slightly too.

Sharpen your pencil Solution

Now it's your turn. Change the activity code so that if the stopwatch was running before **onStop()** was called, it starts running again when the activity regains the focus. Hint: you may need to add a new variable.

```
public class StopwatchActivity extends Activity {
    private int seconds = 0;
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
        runTimer();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
        savedInstanceState.putBoolean("wasRunning", wasRunning);
    }

    @Override
    protected void onStop() {
        super.onStop();
        wasRunning = running;
        running = false;
    }

    @Override
    protected void onStart() {
        super.onStart();
        if (wasRunning) {
            running = true;
        }
    }
```

We added a new variable, wasRunning, to record whether the stopwatch was running before the onStop() method was called so that we know whether to set it running again when the activity becomes visible again.

We'll restore the state of the wasRunning variable if the activity is recreated.

Save the state of the wasRunning variable.

Record whether the stopwatch was running when the onStop() method was called.

Implement the onStart() method. If the stopwatch was running, set it running again.

The updated StopwatchActivity code

We've updated our activity code so that if the stopwatch was running before it lost the focus, it starts running again when it gets the focus back. Make the following changes (in bold) to your version of *StopwatchActivity.java*:

```
public class StopwatchActivity extends Activity {
    private int seconds = 0;
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
        runTimer();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
        savedInstanceState.putBoolean("wasRunning", wasRunning);
    }

    @Override
    protected void onStop() {
        super.onStop();
        wasRunning = running;
        running = false;
    }

    @Override
    protected void onStart() {
        super.onStart();
        if (wasRunning) {
            running = true;
        }
    }
...
```

A new variable, wasRunning, records whether the stopwatch was running before the onStop() method was called.

Restore the state of the wasRunning variable if the activity is recreated.

Save the state of the wasRunning variable.

Record whether the stopwatch was running when the onStop() method was called.

Implement the onStart() method. If the stopwatch was running, we'll set it running again.

We've left out some of the activity code as we don't need to change it.

What happens when you run the app

1. **The user starts the app, and clicks the Start button to set the stopwatch going.**
The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view.

2. **The user navigates to the device's home screen so the Stopwatch app is no longer visible.**
The `onStop()` method gets called, `wasRunning` is set to `true`, `running` is set to `false`, and the number of seconds stops incrementing.

3. **The user navigates back to the Stopwatch app.**
The `onStart()` method gets called, `running` is set to `true`, and the number of seconds starts incrementing again.

Test drive the app

Save the changes to your activity code, then run the app. When you click on the Start button the timer starts: it stops when the app is no longer visible, and it starts again when the app becomes visible again.

there are no Dumb Questions

Q: Could we have used the `onRestart()` method instead of `onStart()` to set the stopwatch running again?

A: `onRestart()` is used when you only want code to run when an app becomes visible after having previously been invisible. It doesn't run when the activity becomes visible for the first time. In our case, we wanted the app to still work when we rotated the device.

Q: Why should that make a difference?

A: When you rotate the device, the activity is destroyed and a new one is created in its place. If we'd put code to set the stopwatch running again in the `onRestart()` method instead of `onStart()`, it wouldn't have run when the activity was recreated. The `onStart()` method gets called in both situations.

What if an app is only partially visible?

So far you've seen what happens when an activity gets created and destroyed, and you've also seen what happens when an activity becomes visible, and when it becomes invisible. But there's one more situation we need to consider: when an activity is visible but doesn't have the focus.

When an activity is visible but doesn't have the focus, the activity is paused. This can happen if another activity appears on top of your activity that isn't full-size or that's transparent. The activity on top has the focus, but the one underneath is still visible and is therefore paused.

The stopwatch activity is still visible, but it's partially obscured and no longer has the focus. When this happens, it pauses.

This is an activity from another app that's appeared on top of the stopwatch.

An activity has a state of paused if it's lost the focus but is still visible to the user. The activity is still alive and maintains all its state information.

There are two lifecycle methods that handle when the activity is paused and when it becomes active again: `onPause()` and `onResume()`. `onPause()` gets called when your activity is visible but another activity has the focus. `onResume()` is called immediately before your activity is about to start interacting with the user. If you need your app to react in some way when your activity is paused, you need to implement these methods.

You'll see on the next page how these methods fit in with the rest of the lifecycle methods you've seen so far.

The activity lifecycle: the foreground lifetime

Let's build on the lifecycle diagram you saw earlier in the chapter, this time including the `onResume()` and `onPause()` methods (the new bits are in bold):

1. **The activity gets launched, and the onCreate() and onStart() methods run.**
   At this point, the activity is visible, but it doesn't have the focus.

2. **The onResume() method runs. It gets called when the activity is about to move into the foreground.**
   After the `onResume()` method has run, the activity has the focus and the user can interact with it.

3. **The onPause() method runs when the activity stops being in the foreground.**
   After the `onPause()` method has run, the activity is still visible but doesn't have the focus.

4. **If the activity moves into the foreground again, the onResume() method gets called.**
   The activity may go through this cycle many times if the activity repeatedly loses and then regains the focus.

5. **If the activity stops being visible to the user, the onStop() method gets called.**
   After the `onStop()` method has run, the activity is no longer visible.

6. **If the activity becomes visible to the user again, the onRestart() method gets called, followed by onStart() and onResume().**
   The activity may go through this cycle many times.

7. **Finally, the activity is destroyed.**
   As the activity moves from running to destroyed, the `onPause()` and `onStop()` methods get called before the activity is destroyed.

Earlier on you talked about how the activity is destroyed and a new one is created when the user rotates the device. What happens if the activity is paused when the device is rotated? Does the activity go through the same lifecycle methods?

That's a great question, so let's look at this in more detail before getting back to the Stopwatch app.

The original activity goes through all its lifecycle methods, from `onCreate()` to `onDestroy()`. A new activity is created when the original is destroyed. As this new activity isn't in the foreground, only the `onCreate()` and `onStart()` lifecycle methods get called. Here's what happens when the user rotates the device when the activity doesn't have the focus::

1. **The user launches the activity.**
   The activity lifecycle methods `onCreate()`, `onStart()`, and `onResume()` get called.

2. **Another activity appears in front of it.**
   The activity's `onPause()` method gets called.

3. **The user rotates the device.**
   Android sees this as a configuration change. The `onStop()` and `onDestroy()` methods get called, and Android destroys the activity. A new activity is created in its place.

4. **The activity is visible but not in the foreground.**
   The `onCreate()` and `onStart()` methods get called. As the activity is visible but doesn't have the focus, `onResume()` isn't called.

I see, the replacement activity doesn't reach a state of "running" because it's not in the foreground. But what if you navigate away from the activity completely so it's not even visible? If the activity's stopped, do onResume() and onPause() get called before onStop()?

Activities can go straight from onStart() to onStop() and bypass onPause() and onResume().

If you have an activity that's visible, but never in the foreground and never has the focus, the `onPause()` and `onResume()` methods ***never get called***.

The `onResume()` method gets called when the activity appears in the foreground and has the focus. If the activity is only visible behind other activities, the `onResume()` method doesn't get called.

Similarly, the `onPause()` method gets called only when the activity is no longer in the foreground. If the activity is never in the foreground, this method won't get called.

If an activity stops or gets destroyed before it appears in the foreground, the onStart() method is followed by the onStop() method. onResume() and onPause() are bypassed.

Stop the stopwatch if the activity's paused

Let's get back to the Stopwatch app.

So far we've made the stopwatch stop if the Stopwatch app isn't visible, and made it start again when the app becomes visible again. We did this by overriding the `onStop()` and `onStart()` methods like this:

```
@Override
protected void onStop() {
    super.onStop();
    wasRunning = running;
    running = false;
}

@Override
protected void onStart() {
    super.onStart();
    if (wasRunning) {
        running = true;
    }
}
```

Let's get the app to have the same behavior if the app is only partially visible. We'll get the stopwatch to stop if the activity is paused, and start again when the activity is resumed. So what changes do we need to make to the lifecycle methods?

We want the Stopwatch app to stop running when the activity is paused, and start it again (if it was running) when the activity is resumed. In other words, we want it to behave the same as when the activity is stopped or started. This means that instead of repeating the code we already have in multiple methods, we can use one method when the activity is paused or stopped, and another method when the activity is resumed or started.

Implement the onPause() and onResume() methods

We'll start with when the activity is resumed or started.

When the activity is resumed, the activity's `onResume()` lifecycle method is called. If the activity is started, the activity's `onResume()` method is called after calling `onStart()`. The `onResume()` method is called irrespective of whether the activity is resumed or started, which means that if we move our `onStart()` code to the `onResume()` method, our app will behave the same irrespective of whether the activity is resumed or started. This means we can remove our `onStart()` method, and replace it with the `onResume()` method like this:

```
@Override
protected void onStart() {
    super.onStart();
    if (wasRunning) {
        running = true;
    }
}
```

Delete the onStart() method.

```
@Override
protected void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}
```

Add the onResume() method.

The onResume() method is called when the activity is started or resumed. As we want the app to do the same thing irrespective of whether it's started or resumed, we only need to implement the onResume() method.

We can do something similar when the activity is paused or stopped.

When the activity is paused, the activity's `onPause()` lifecycle method is called. If the activity is stopped, the activity's `onPause()` method is called prior to calling `onStop()`. The `onPause()` method is called irrespective of whether the activity is paused or stopped, which means we can move our on`Stop()` code to the `onPause()` method:

```
@Override
protected void onStop() {
    super.onStop();
    wasRunning = running;
    running = false;
}
```

Delete the onStop() method.

```
@Override
protected void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}
```

Add the onPause() method.

The onPause() method is called when the activity is paused or stopped. This means we only need to implement the onPause() method.

The complete StopwatchActivity code

Here's the full *StopwatchActivity.java* code for the finished app (with changes in bold):

```
package com.hfad.stopwatch;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import java.util.Locale;
import android.os.Handler;
import android.widget.TextView;

public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
        runTimer();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
        savedInstanceState.putBoolean("wasRunning", wasRunning);
    }
```

Use seconds, running, and wasRunning respectively to record the number of seconds passed, whether the stopwatch is running, and whether the stopwatch was running before the activity was paused.

Get the previous state of the stopwatch if the activity's been destroyed and recreated.

Save the state of the stopwatch if it's about to be destroyed.

The activity code continues over the page.

The activity code (continued)

```
@Override
protected void onStop() {
    super.onStop();
    wasRunning = running;
    running = false;
}

@Override
protected void onStart() {
    super.onStart();
    if (wasRunning) {
        running = true;
    }
}
```

Delete these two methods.

```
@Override
protected void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}

@Override
protected void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}

//Start the stopwatch running when the Start button is clicked.
public void onClickStart(View view) {
    running = true;
}
```

If the activity's paused, stop the stopwatch.

If the activity's resumed, start the stopwatch again if it was running previously.

This gets called when the Start button is clicked.

The activity code continues over the page.

The activity code (continued)

```
    //Stop the stopwatch running when the Stop button is clicked.
    public void onClickStop(View view) {
        running = false;
    }

    //Reset the stopwatch when the Reset button is clicked.
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }

    //Sets the number of seconds on the timer.
    private void runTimer() {
        final TextView timeView = (TextView)findViewById(R.id.time_view);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                int hours = seconds/3600;
                int minutes = (seconds%3600)/60;
                int secs = seconds%60;
                String time = String.format(Locale.getDefault(),
                            "%d:%02d:%02d", hours, minutes, secs);
                timeView.setText(time);
                if (running) {
                    seconds++;
                }
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

This gets called when the Stop button is clicked.

This gets called when the Reset button is clicked.

The runTimer() method uses a Handler to increment the seconds and update the text view.

Let's go through what happens when the code runs.

What happens when you run the app

1. **The user starts the app, and clicks on the Start button to set the stopwatch going.** The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view.

2. **Another activity appears in the foreground, leaving StopwatchActivity partially visible.** The `onPause()` method gets called, `wasRunning` is set to `true`, `running` is set to `false`, and the number of seconds stops incrementing.

3. **When StopwatchActivity returns to the foreground, the onResume() method gets called, running is set to true, and the number of seconds starts incrementing again.**

Test drive the app

Save the changes to your activity code, then run the app. When you click on the Start button, the timer starts; it stops when the app is partially obscured by another activity; and it starts again when the app is back in the foreground.

BE the Activity

On the right, you'll see some activity code. Your job is to play like you're the activity and say which code will run in each of the situations below. We've labeled the code we want you to consider. We've done the first one to start you off.

User starts the activity and starts using it.

Code segments A, G, D. The activity is created, then made visible, then receives the focus.

User starts the activity, starts using it, then switches to another app.

This one's tough.

User starts the activity, starts using it, rotates the device, switches to another app, then goes back to the activity.

```
...
class MyActivity extends Activity{

    protected void onCreate(
              Bundle savedInstanceState) {
    A //Run code A
      ...
    }

    protected void onPause() {
    B //Run code B
      ...
    }

    protected void onRestart() {
    C //Run code C
      ...
    }

    protected void onResume() {
    D //Run code D
      ...
    }

    protected void onStop() {
    E //Run code E
      ...
    }

    protected void onRecreate() {
    F //Run code F
      ...
    }

    protected void onStart() {
    G //Run code G
      ...
    }

    protected void onDestroy() {
    H //Run code H
      ...
    }
}
```

BE the Activity Solution

On the right, you'll see some activity code. Your job is to play like you're the activity and say which code will run in each of the situations below. We've labeled the code we want you to consider. We've done the first one to start you off.

```
...
class MyActivity extends Activity{

    protected void onCreate(
              Bundle savedInstanceState) {
    A //Run code A
      ...
    }

    protected void onPause() {
    B //Run code B
      ...
    }

    protected void onRestart() {
    C //Run code C
      ...
    }

    protected void onResume() {
    D //Run code D
      ...
    }

    protected void onStop() {
    E //Run code E
      ...
    }

    protected void onRecreate() {
    F //Run code F
      ...
    }

    protected void onStart() {
    G //Run code G
      ...
    }

    protected void onDestroy() {
    H //Run code H
      ...
    }
}
```

There's no lifecycle method called onRecreate().

User starts the activity and starts using it.

Code segments A, G, D. The activity is created, then made visible, then receives the focus.

User starts the activity, starts using it, then switches to another app.

Code segments A, G, D, B, E. The activity is created, then made visible, then receives the focus. When the user switches to another app, it loses the focus and is no longer visible to the user.

User starts the activity, starts using it, rotates the device, switches to another app, then goes back to the activity.

Code segments A, G, D, B, E, H, A, G, D, B, E, C, G, D. First, the activity is created, made visible, and receives the focus. When the device is rotated, the activity loses the focus, stops being visible, and is destroyed. It's then created again, made visible, and receives the focus. When the user switches to another app and back again, the activity loses the focus, loses visibility, becomes visible again, and regains the focus.

Your handy guide to the lifecycle methods

Method	When it's called	Next method
onCreate()	When the activity is first created. Use it for normal static setup, such as creating views. It also gives you a `Bundle` that contains the previously saved state of the activity.	`onStart()`
onRestart()	When your activity has been stopped but just before it gets started again.	`onStart()`
onStart()	When your activity is becoming visible. It's followed by `onResume()` if the activity comes into the foreground, or `onStop()` if the activity is made invisible.	`onResume()` or `onStop()`
onResume()	When your activity is in the foreground.	`onPause()`
onPause()	When your activity is no longer in the foreground because another activity is resuming. The next activity isn't resumed until this method finishes, so any code in this method needs to be quick. It's followed by `onResume()` if the activity returns to the foreground, or `onStop()` if it becomes invisible.	`onResume()` or `onStop()`
onStop()	When the activity is no longer visible. This can be because another activity is covering it, or because this activity is being destroyed. It's followed by `onRestart()` if the activity becomes visible again, or `onDestroy()` if the activity is being destroyed.	`onRestart()` or `onDestroy()`
onDestroy()	When your activity is about to be destroyed or because the activity is finishing.	None

Your Android Toolbox

You've got Chapter 4 under your belt and now you've added the activity lifecycle to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Each app runs in its own process by default.
- Only the main thread can update the user interface.
- Use a `Handler` to schedule code or post code to a different thread.
- A device configuration change results in the activity being destroyed and recreated.
- Your activity inherits the lifecycle methods from the `android.app.Activity` class. If you override any of these methods, you need to call up to the method in the superclass.
- `onSaveInstanceState(Bundle)` enables your activity to save its state before the activity gets destroyed. You can use the `Bundle` to restore state in `onCreate()`.
- You add values to a `Bundle` using `bundle.put*("name", value)`. You retrieve values from the bundle using `bundle.get*("name")`.
- `onCreate()` and `onDestroy()` deal with the birth and death of the activity.
- `onRestart()`, `onStart()`, and `onStop()` deal with the visibility of the activity.
- `onResume()` and `onPause()` handle when the activity gains and loses the focus.

5 views and view groups

Enjoy the View

You've seen how to arrange GUI components using a linear layout, but so far we've only scratched the surface.

In this chapter we'll **look a little deeper** and show you how linear layouts *really work*. We'll introduce you to the **frame layout**, a simple layout used to stack views, and we'll also take a tour of the **main GUI components** and **how you use them**. By the end of the chapter, you'll see that even though they all look a little different, all layouts and GUI components have ***more in common than you might think***.

Your user interface is made up of layouts and GUI components

As you already know, a layout defines what a screen looks like, and you define it using XML. Layouts usually contain GUI components such as buttons and text fields. Your user interacts with these to make your app do something.

All the apps you've seen in the book so far have used linear layouts, where GUI components are arranged in a single column or row. In order to make the most out of them, however, we need to spend a bit of time looking at how they work, and how to use them effectively.

These are all examples of linear layouts.

In this chapter, we're going to take a closer look at linear layouts, introduce you to their close relative the frame layout, and show you other GUI components you can use to make your app more interactive.

Let's start with linear layouts.

LinearLayout displays views in a single row or column

As you already know, a linear layout displays its views next to each other, either vertically or horizontally. If it's vertically, the views are displayed in a single column. If it's horizontally, the views are displayed in a single row.

You define a linear layout using the `<LinearLayout>` element like this:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    ...>
    ...
</LinearLayout>
```

You use <LinearLayout> to define a linear layout.

The layout_width and layout_height specify what size you want the layout to be.

The orientation specifies whether you want to display views vertically or horizontally.

There may be other attributes too.

The `xmlns:android` attribute is used to specify the Android namespace, and you must always set it to `"http://schemas.android.com/apk/res/android"`.

You MUST set the layout's width and height

The `android:layout_width` and `android:layout_height` attributes specify how wide and high you want the layout to be. **These attributes are mandatory for all types of layout and view**.

You can set `android:layout_width` and `android:layout_height` to `"wrap_content"`, `"match_parent"` or a specific size such as 8dp—that's 8 density-independent pixels. `"wrap_content"` means that you want the layout to be just big enough to hold all of the views inside it, and `"match_parent"` means that you want the layout to be as big as its parent—in this case, as big as the device screen minus any padding (there's more about padding in a couple of pages). You will usually set the layout width and height to `"match_parent"`.

You may sometimes see `android:layout_width` and `android:layout_height` set to `"fill_parent"`. `"fill_parent"` was used in older versions of Android, and it's now replaced by `"match_parent"`. `"fill_parent"` is deprecated.

Geek Bits

What are density-independent pixels?

Some devices create very sharp images by using very tiny pixels. Other devices are cheaper to produce because they have fewer, larger pixels. You use density-independent pixels (dp) to avoid creating interfaces that are overly small on some devices, and overly large on others. A measurement in density-independent pixels is roughly the same size across all devices.

LinearLayout
FrameLayout

Orientation is vertical or horizontal

You specify the direction in which you wish to arrange views using the `android:orientation` attribute.

As you've seen in earlier chapters, you arrange views vertically using:

```
android:orientation="vertical"
```

This displays the views in a single column.

You arrange views horizontally in a single row using:

```
android:orientation="horizontal"
```

When the orientation is horizontal, views are displayed from left to right by default. This is great for languages that are read from left to right, but what if the user has set the language on their device to one that's read from right to left?

For apps where the minimum SDK is *at least* API 17, you can get views to rearrange themselves depending on the language setting on the device. If the user's language is read from right to left, you can get the views to arrange themselves starting from the right.

To do this, you declare that your app supports languages that are read from right to left in your *AndroidManifest.xml* file like this:

```
<manifest ...>
    <application
        ...
        android:supportsRtl="true">
        ...
    </application>
</manifest>
```

Android Studio may add this line of code for you. It must go inside the <application> tag.

`supportsRtl` means "supports right-to-left languages."

Padding adds space

If you want there to be a bit of space around the edge of the layout, you can set **padding** attributes. These attributes tell Android how much padding you want between each of the layout's sides and its parent. Here's how you would tell Android you want to add padding of 16dp to all edges of the layout:

```
<LinearLayout ...
    android:padding="16dp" >
    ...
</LinearLayout>
```

This adds the same padding to all edges of the layout.

If you want to add different amounts of padding to different edges, you can specify the edges individually. Here's how you would add padding of 32dp to the top of the layout, and 16dp to the other edges:

```
<LinearLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="32dp" >
    ...
</LinearLayout>
```

Add padding to the individual edges.

If your app supports right-to-left languages, you can use:

```
android:paddingStart="16dp"
```

and:

```
android:paddingEnd="16dp"
```

to add padding to the start and end edges of the layout instead of their left and right edges.

`android:PaddingStart` adds padding to the start edge of the layout. The start edge is on the left for languages that are read from left to right, and the right edge for languages that are read from right to left.

`android:PaddingEnd` adds padding to the end edge of the layout. The end edge is on the right for languages that are read from left to right, and the left edge for languages that are read from right to left.

You can only use start and end properties with API 17 or above.

If you want your app to work on older versions of Android, you must use the left and right properties instead.

LinearLayout
FrameLayout

Add a dimension resource file for consistent padding across layouts

In the example on the previous page, we hardcoded the padding and set it to 16dp. An alternative approach is to specify the padding in a *dimension resource file* instead. Doing so makes it easier to maintain the padding dimensions for all the layouts in your app.

To use a dimension resource file, you first need to add one to your project. To do this, first select the *app/src/main/res/values* folder in Android Studio, then go to File menu and choose New→Values resource file. When prompted, enter a name of "dimens" and click on the OK button. This creates a new resource file called *dimens.xml*.

Android Studio may have already added this file for you, but it depends on which version of Android Studio you're using.

Once you've created the dimension resource file, you add dimensions to it using the `<dimen>` element. As an example, here's how you would add dimensions for the horizontal and vertical margins to *dimens.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

This creates two dimension resources.

You use the dimensions you create by setting the padding attributes in your layout file to the name of a dimension resource like this:

```
<LinearLayout ...
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin">
```

The paddingLeft and paddingRight attributes are set to @dimen/activity_horizontal_margin.

The paddingTop and paddingBottom attributes are set to @dimen/activity_vertical_margin.

With that kind of setup, at runtime, Android looks up the values of the attributes in the dimension resource file and applies the values it finds there.

A linear layout displays views in the order they appear in the layout XML

When you define a linear layout, you add views to the layout in the order in which you want them to appear. So, if you want a text view to appear above a button in a linear layout, you *must* define the text view first:

```
<LinearLayout ... >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_view1" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me" />
</LinearLayout>
```

If you define the text view above the button in the XML, the text view will appear above the button when displayed.

You specify the width and height of any views using `android:layout_width` and `android:layout_height`. The code:

```
android:layout_width="wrap_content"
```

means that you want the view to be just wide enough for its content to fit inside it—for example, the text displayed on a button or in a text view. The code:

```
android:layout_width="match_parent"
```

means that you want the view to be as wide as the parent layout.

If you need to refer to a view elsewhere in your code, you need to give it an ID. As an example, you'd give the text view an ID of `"text_view"` using the code:

```
...
    <TextView
        android:id="@+id/text_view"
        ... />
...
```

android:layout_width and android:layout_height are mandatory attributes for all views, no matter which layout you use.

They can take the values wrap_content, match_parent, or a specific dimension value such as 16dp.

LinearLayout
FrameLayout

Use margins to add distance between views

When you position a view using a linear layout, the layout doesn't leave much of a gap between views. You can increase the size of the gap by adding one or more **margins** to the view.

As an example, suppose you wanted to put one view below another, but add 48dp of extra space between the two. To do that, you'd add a margin of 48dp to the top of the bottom view:

```
LinearLayout ... >
    <Button
        android:id="@+id/button_click_me"
        ... />

    <Button
        android:id="@+id/button_below"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="48dp"
        android:text="@string/button_below" />
</LinearLayout>
```

Adding a margin to the top of the bottom button adds extra space between the two views.

Here's a list of the margins you can use to give your views extra space. Add the attribute to the view, and set its value to the size of margin you want:

```
android:attribute="8dp"
```

Attribute	What it does	
layout_marginTop	Adds extra space to the top of the view.	
layout_marginBottom	Adds extra space to the bottom of the view.	
layout_marginLeft, layout_marginStart	Adds extra space to the left (or start) of the view.	
layout_marginRight, layout_marginEnd	Adds extra space to the right (or end) of the view.	
layout_margin	Adds equal space to each side of the view.	

Let's change up a basic linear layout

At first glance, a linear layout can seem basic and inflexible. After all, all it does is arrange views in a particular order. To give you more flexibility, you can tweak your layout's appearance using some more of its attributes. To show you how this works, we're going to transform a basic linear layout.

The layout is composed of two editable text fields and a button. To start with, these text fields are simply displayed vertically on the screen like this:

Each view takes up the least possible amount of vertical space.

We're going to change the layout so that the button is displayed in the end corner of the layout, and one of the editable text fields takes up any remaining space.

The Message editable text field has been given a lot more space.

The Send button appears at the bottom-right of the screen for languages that are read from left to right.

LinearLayout
FrameLayout

Here's the starting point for the linear layout

The linear layout contains two editable text fields and a button. The button is labeled "Send," and the editable text fields contain hint text values of "To" and "Message."

Hint text in an editable text field is text that's displayed when the field is empty. It's used to give users a hint as to what sort of text they should enter. You define hint text using the `android:hint` attribute:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/send" />
</LinearLayout>
```

The editable text fields are as wide as the parent layout.

android:hint displays a hint to the user as to what they should type in the editable text field.

The values of these Strings are defined in strings.xml as usual.

The text values displayed here come from String resources in strings.xml.

All of these views take up just as much vertical space in the layout as they need for their contents. So how do we make the Message text field taller?

Make a view streeeeetch by adding weight

All of the views in our basic layout take up just as much vertical space as they need for their content. But what we actually want is to make the Message text field stretch to take up any vertical space in the layout that's not being used by the other views.

We want to make the Message text field stretch vertically so that it fills any spare space in the layout.

In order to do this, we need to allocate some **weight** to the Message text field. Allocating weight to a view is a way of telling it to stretch to take up extra space in the layout.

You assign weight to a view using:

```
android:layout_weight="number"
```

where `number` is some number greater than 0.

When you allocate weight to a view, the layout first makes sure that each view has enough space for its content: each button has space for its text, each editable text field has space for its hint, and so on. Once it's done that, the layout takes any extra space, and divides it proportionally between the views with a weight of 1 or greater.

LinearLayout
FrameLayout

Adding weight to one view

We need the Message editable text field to take up any extra space in the layout. To do this, we'll set its `layout_weight` attribute to `1`. As this is the only view in the layout with a weight value, this will make the text field stretch vertically to fill the remainder of the screen. Here's the code:

```
<LinearLayout ... >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/send" />
</LinearLayout>
```

This <EditText> and the <Button> have no layout_weight attribute set. They'll take up as much room as their content needs, but no more.

This view is the only one with any weight. It will expand to fill the space that's not needed by any of the other views.

The height of the view will be determined by the linear layout based on the layout_weight. Setting the layout_height to "0dp" is more efficient than setting it to "wrap_content", as this way Android doesn't have to work out the value of "wrap_content".

Giving the Message editable text field a weight of `1` means that it takes up all of the extra space that's not used by the other views in the layout. This is because neither of the other two views has been allocated any weight in the layout XML.

By default, the Message hint text appears in the middle. We'll deal with that next.

The Message view has a weight of 1. As it's the only view with its weight attribute set, it expands to take up any extra vertical space in the layout.

Adding weight to multiple views

In this example, we only had one view with a weight attribute set. But what if we had *more* than one?

Suppose we gave the To text field a weight of `1`, and the Message text field a weight of `2`, like this:

```
<LinearLayout ... >
    ...
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:hint="@string/to" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:hint="@string/message" />
    ...
</LinearLayout>
```

To figure out how much extra space each view takes up, start by adding together the `layout_weight` attributes for each view. In our case, this is 1+2=3. The amount of extra space taken up by each view will be the view's weight divided by the total weight. The To view has a weight of `1`, so this means it will take up 1/3 of the remaining space in the layout. The Message view has a weight of `2`, so it will take up 2/3 of the remaining space.

This is just an example; we're not really going to change the layout so that it looks like this.

The To view has a weight of 1, so it takes up 1/3 of the extra space.

The Message view has a weight of 2, so it takes up 2/3 of the extra space.

LinearLayout
FrameLayout

Gravity controls the position of a view's contents

The next thing we need to do is move the hint text inside the Message text field. At the moment, it's centered vertically inside the view. We need to change it so that the text appears at the top of the text field. We can achieve this using the `android:gravity` attribute.

The `android:gravity` attribute lets you specify how you want to position the contents of a view inside the view—for example, how you want to position text inside a text field. If you want the text inside a view to appear at the top, the following code will do the trick:

```
android:gravity="top"
```

We'll add an `android:gravity` attribute to the Message text field so that the hint text moves to the top of the view:

```
<LinearLayout ... >
    ...
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    ...
</LinearLayout>
```

Displays the text inside the text field at the top of the text field

Test drive

Adding the `android:gravity` attribute to the Message text field moves the hint text to the top of the view, just like we want.

You'll find a list of the other values you can use with the `android:gravity` attribute on the next page.

Values you can use with the android:gravity attribute

Here are some more of the values you can use with the `android:gravity` attribute. Add the attribute to your view, and set its value to one of the values below:

```
android:gravity="value"
```

android:gravity lets you say where you want the view's contents to appear inside the view.

Value	What it does
top	Puts the view's contents at the top of the view.
bottom	Puts the view's contents at the bottom of the view.
left	Puts the view's contents at the left of the view.
right	Puts the view's contents at the right of the view.
start	Puts the view's contents at the start of the view.
end	Puts the view's contents at the end of the view.
center_vertical	Centers the view's contents vertically.
center_horizontal	Centers the view's contents horizontally.
center	Centers the view's contents vertically and horizontally.
fill_vertical	Makes the view's contents fill the view vertically.
fill_horizontal	Makes the view's contents fill the view horizontally.
fill	Makes the view's contents fill the view vertically and horizontally.

start and end are only available if you're using API 17 or above.

You can also apply multiple gravities to a view by separating each value with a "|". To sink a view's contents to the bottom-end corner, for example, use:

```
android:gravity="bottom|end"
```

LinearLayout
FrameLayout

layout_gravity controls the position of a view within a layout

There's one final change we need to make to our layout. The Send button currently appears in the bottom-left corner. We need to move it over to the end instead (the bottom-right corner for left-to-right languages). To do this, we'll use the `android:layout_gravity` attribute.

The `android:layout_gravity` attribute lets you specify where you want a view in a linear layout to appear in its enclosing space. You can use it to push a view to the right, for instance, or center the view horizontally. To move our button to the end, we'd need to add the following to the button's code:

```
android:layout_gravity="end"
```

Linear layouts have two attributes that sound similar to one another, gravity and layout_gravity.

A couple of pages ago, we used the `android:gravity` attribute to position the Message text inside a text view. This is because the `android:gravity` attribute lets you say where you want a view's **contents** to appear.

`android:layout_gravity` deals with the **placement of the view itself**, and lets you control where views appear in their available space. In our case, we want the view to move to the end of its available space, so we're using:

```
android:layout_gravity="end"
```

There's a list of some of the other values you can use with the `android:layout_gravity` attribute on the next page.

More values you can use with the android:layout_gravity attribute

Here are some of the values you can use with the `android:layout_gravity` attribute. Add the attribute to your view, and set its value to one of the values below:

```
android:layout_gravity="value"
```

You can apply multiple layout_gravity values by separating each one with a "|". As an example, use android:layout_gravity="bottomlend" to move a view to the bottom-end corner of its available space.

Value	What it does
top, bottom, left, right	Puts the view at the top, bottom, left, or right of its available space.
start, end	Puts the view at the start or end of its available space.
center_vertical, center_horizontal	Centers the view vertically or horizontally in its available space.
center	Centers the view vertically and horizontally in its available space.
fill_vertical, fill_horizontal	Grows the view so that it fills its available space vertically or horizontally.
fill	Grows the view so that it fills its available space vertically and horizontally.

android:layout_gravity lets you say where you want views to appear in their available space.

android:layout_gravity deals with the placement of the view itself, whereas android:gravity controls how the view's contents are displayed.

The full linear layout code

Here's the full code for the linear layout:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end"
        android:text="@string/send" />

</LinearLayout>
```

android:gravity is different from android:layout_gravity. android:gravity relates to the contents of the view, while android:layout_gravity relates to the view itself.

LinearLayout: a summary

Here's a summary of how you create linear layouts.

How you specify a linear layout

You specify a linear layout using `<LinearLayout>`. You must specify the layout's width, height, and orientation, but padding is optional:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    ...>
    ...
</LinearLayout>
```

Views get displayed in the order they're listed in the code

When you define a linear layout, you add views to the layout in the order in which you want them to appear.

Stretch views using weight

By default, all views take up just as much space as necessary for their content. If you want to make one or more of your views take up more space, you can use the `weight` attribute to make it stretch:

```
android:layout_weight="1"
```

Use gravity to specify where a view's contents appear within a view

The `android:gravity` attribute lets you specify how you want to position the contents of a view—for example, how you want to position text inside a text field.

Use layout_gravity to specify where a view appears in its available space

The `android:layout_gravity` attribute lets you specify where you want a view in a linear layout to appear in its parent layout. You can use it to push a view to the end, for instance, or center the view horizontally.

That's everything we've covered on linear layouts. Next we'll take a look at the **frame layout**.

LinearLayout
FrameLayout

Frame layouts stack their views

As you've already seen, linear layouts arrange their views in a single row or column. Each view is allocated its own space on the screen, and they don't overlap one another.

Sometimes, however, you *want* your views to overlap. As an example, suppose you want to display an image with some text overlaid on top of it. You wouldn't be able to achieve this just using a linear layout.

If you want a layout whose views can overlap, a simple option is to use a **frame layout**. Instead of displaying its views in a single row or column, it stacks them on top of each other. This allows you to, for example, display text on top of an image.

Frame layouts let your views overlap one another. This is useful for displaying text on top of images, for example.

How you define a frame layout

You define a frame layout using the `<FrameLayout>` element like this:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...>

    ...

</FrameLayout>
```

These are the same attributes we used for our linear layout.

You use <FrameLayout> to define a frame layout.

This is where you add any views you wish to stack in the frame layout.

Just like a linear layout, the `android:layout_width` and `android:layout_height` attributes are mandatory and specify the layout's width and height.

Create a new project

To see how frame layouts work, we're going to use one to overlay text on an image. Create a new Android Studio project for an application named "Duck" with a company name of "hfad.com", making the package name `com.hfad.duck`. The minimum SDK should be API 19 so that it will work on most devices. You'll need an empty activity called "MainActivity" with a layout called "activity_main" so that your code matches ours. **Make sure you uncheck the Backwards Compatibility (AppCompat) option when you create the activity.**

LinearLayout
FrameLayout

Add an image to your project

We're going to use an image called *duck.jpg* in our layout, so we need to add it to our project.

To do this, you first need to create a *drawable* resource folder (if Android Studio hasn't already created it for you). This is the default folder for storing image resources in your app. Switch to the Project view of Android Studio's explorer, select the *app/src/main/res* folder, go to the File menu, choose the New... option, then click on the option to create a new Android resource directory. When prompted, choose a resource type of "drawable", name the folder "drawable", and click on OK.

Once you've created the *drawable* folder, download the file *duck.jpg* from *https://git.io/v9oet*, then add it to the *app/src/main/res/drawable* folder.

duck.jpg

We're going to change *activity_main.xml* so that it uses a frame layout containing an image view (a view that displays an image) and a text view. To do this, replace the code in your version of *activity_main.xml* with ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context="com.hfad.duck.MainActivity">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/duck"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:textSize="20sp"
        android:text="It's a duck!" />
</FrameLayout>
```

We're using a frame layout.

This adds an image to the frame layout. You'll find out more about image views later in the chapter.

This crops the image's edges so it fits in the available space.

This line tells Android to use the image named "duck" located in the drawable folder.

This adds a text view to the frame layout.

We've increased the size of the text.

In a real-world duck app, you'd want to add this text as a String resource.

Then run your app, and we'll look at what the code does on the next page.

LinearLayout
FrameLayout

A frame layout stacks views in the order they appear in the layout XML

When you define a frame layout, you add views to the layout in the order in which you want them to be stacked. The first view is displayed first, the second is stacked on top of it, and so on. In our case, we've added an image view followed by a text view, so the text view appears on top of the image view:

```
<FrameLayout ...>
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/duck"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:textSize="20sp"
        android:text="It's a duck!" />
</FrameLayout>
```

This is the image view.

This is the text view.

Position views in the layout using layout_gravity

By default, any views you add to a frame layout appear in the top-left corner. You can change the position of these views using the `android:layout_gravity` attribute, just as you could with a linear layout. As an example, here's how you would move the text view to the bottom-end corner of the image:

```
...
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:layout_gravity="bottom|end"
        android:textSize="20sp"
        android:text="It's a duck!" />
</FrameLayout>
```

This sinks the text to the bottom-end corner.

You can nest layouts

One of the disadvantages of using a frame layout is that it's easy for views to overlap one another when you don't want them to. As an example, you may want to display two text views in the bottom-end corner, one above the other:

If you're not careful, views can overlap one another like this.

It's possible to solve this problem by adding margins or padding to the text views. A neater solution, however, is to add them to a linear layout, which you then nest inside the frame layout. Doing this allows you to arrange the two text views linearly, then position them as a group inside the frame layout:

Here, the frame layout contains an image view and a linear layout.

This is the linear layout. It contains two text views neatly arranged in a single column.

We'll show you the full code for how to do this on the next page.

The full code to nest a layout

Here's the full code to nest a linear layout in a frame layout. Update your version of *activity_main.xml* to include our changes, then run your app to see how it looks.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context="com.hfad.duck.MainActivity">

    <ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:scaleType="centerCrop"
    android:src="@drawable/duck"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_gravity="bottom|end"
        android:gravity="end"
        android:padding="16dp" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:text="It's a duck!"  />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:text="(not a real one)" />
    </LinearLayout>
</FrameLayout>
```

We're adding a linear layout that's just big enough to contain its text views.

Move each text view in the linear layout to the end of its available space.

This line sinks the linear layout to the bottom-end corner of the frame layout.

FrameLayout: a summary

Here's a summary of how you create frame layouts.

How you specify a frame layout

You specify a frame layout using `<FrameLayout>`. You must specify the layout's width and height:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...>

    ...

</FrameLayout>
```

Views are stacked in the order they appear

When you define a frame layout, you add views to the layout in the order in which you want them to be stacked. The first view you add is displayed on the bottom of the stack, the next view is stacked on top of it, and so on.

Use layout_gravity to specify where a view appears

The `android:layout_gravity` attribute lets you specify where you want a view in a frame layout to appear. You can use it to push a view to the end, for instance, or sink it to the bottom-end corner.

Now that you've seen how to use two simple Android layouts, a linear layout and a frame layout, have a go at the following exercise.

BE the Layout

Three of the five screens below were made from layouts on the next page. Your job is to match each of the three layouts to the screen that the layout would produce.

1

2

3

4

5

A

```
<LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="HELLO!" />
</LinearLayout>
```

B

```
<LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="HELLO!" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HI!" />
</LinearLayout>
```

C

```
<LinearLayout  xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HELLO!" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HI!" />
</LinearLayout>
```

BE the Layout Solution

Three of the five screens below were made from layouts on the next page. Your job is to match each of the three layouts to the screen that the layout would produce.

None of the layouts produce these screens.

1

A

```
<LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="HELLO!" />
</LinearLayout>
```

This has one button that fills the screen.

3

B

```
<LinearLayout xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="HELLO!" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HI!" />
</LinearLayout>
```

This button fills the screen, leaving space for another one underneath it.

4

C

```
<LinearLayout  xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.views.MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HELLO!" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="HI!" />
</LinearLayout>
```

Both buttons have their layout_width and layout_height properties set to "wrap_content", so they take up just enough space to display their contents.

Layouts and GUI components have a lot in common

You may have noticed that all layout types have attributes in common. Whichever type of layout you use, you must specify the layout's width and height using the `android:layout_width` and `android:layout_height` attributes. And this requirement isn't just limited to layouts—the `android:layout_width` and `android:layout_height` are mandatory for all GUI components too.

This is because **all layouts and GUI components are subclasses of the Android `View` class**. Let's look at this in more detail.

GUI components are a type of View

You've already seen that GUI components are all types of views—behind the scenes, they are all subclasses of the `android.view.View` class. This means that all of the GUI components in your user interface have attributes and behavior in common. They can all be displayed on the screen, for instance, and you get to say how tall or wide they should be. Each of the GUI components you use in your user interface takes this basic functionality and extends it.

Layouts are a type of View called a ViewGroup

It's not just the GUI components that are a type of view. Under the hood, a layout is a special type of view called a **view group**. All layouts are subclasses of the `android.view.ViewGroup` class. A view group is a type of view that can contain other views.

A GUI component is a type of view, an object that takes up space on the screen.

A layout is a type of view group, which is a special type of view that can contain other views.

What being a view buys you

A `View` object occupies rectangular space on the screen. It includes the functionality all views need in order to lead a happy helpful life in Androidville. Here are some of the qualities of views that we think are the most important:

Getting and setting properties

Each view is a Java object behind the scenes, and that means you can get and set its properties in your activity code. As an example, you can retrieve the value selected in a spinner or change the text in a text view. The exact properties and methods you can access depend on the type of view.

To help you get and set view properties, each view can have an ID associated with it so that you can refer to it in your code.

Size and position

You specify the width and height of views so that Android knows how big they need to be. You can also say whether any padding is needed around the view.

Once your view has been displayed, you can retrieve the position of the view, and its actual size on the screen.

Focus handling

Android handles how the focus moves depending on what the user does. This includes responding to any views that are hidden, removed, or made visible.

Event handling and listeners

Views can respond to events. You can also create listeners so that your app can react to things happening in the view. As an example, all views can react to getting or losing the focus, and a button (and all of its subclasses) can react to being clicked.

As a view group is also a type of view, this means that all layouts and GUI components share this common functionality.

Here are some of the View methods you can use in your activity code. As these are in the base View class, they're common to all views and view groups.

android.view.View
getId()
getHeight()
getWidth()
setVisibility(int)
findViewById(int)
isClickable()
isFocused()
requestFocus()
...

A layout is really a hierarchy of Views

The layout you define using XML gives you a *hierarchical tree of views and view groups*. As an example, here's a linear layout containing a button and an editable text field. The linear layout is a view group, and the button and text field are both views. The view group is the view's parent, and the views are the view group's children:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ... >

    <Button
        android:id="@+id/send"
        ... />

    <EditText
        android:id="@+id/message"
        ... />

</LinearLayout>
```

Behind the scenes, when you build your app, the layout XML is converted to a `ViewGroup` object containing a tree of `View`s. In the example above, the button gets translated to a `Button` object, and the text view gets translated to a `TextView` object. `Button` and `TextView` are both subclasses of `View`.

This is the reason why you can manipulate the views in your layout using Java code. Behind the scenes, all of the views are rendered to Java `View` objects.

Playing with views

Let's look at the most common GUI components. You've already seen some of these, but we'll review them anyway. We won't show you the whole API for each of these—just selected highlights to get you started.

Text view

This is a text view

A text view is used for displaying text.

Defining it in XML

You define a text view in your layout using the `<TextView>` element. You use `android:text` to say what text you want it to display, usually by using a String resource:

```
<TextView
    android:id="@+id/text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text" />
```

The `TextView` API includes many attributes to control the text view's appearance, such as the text size. To change the text size, you use the `android:textSize` attribute like this:

```
android:textSize="16sp"
```

You specify the text size using scale-independent pixels (sp). Scale-independent pixels take into account whether users want to use large fonts on their devices. A text size of 16sp will be physically larger on a device configured to use large fonts than on a device configured to use small fonts.

Using it in your activity code

You can change the text displayed in your text view using code like this:

```
TextView textView = (TextView) findViewById(R.id.text_view);
textView.setText("Some other String");
```

Editable text view

This is like a text view, but editable.

Defining it in XML

You define an editable text view in XML using the `<EditText>` element. You use the `android:hint` attribute to give a hint to the user as to how to fill in the field.

```
<EditText
    android:id="@+id/edit_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_text" />
```

You can use the `android:inputType` attribute to define what type of data you're expecting the user to enter so that Android can help them. As an example, if you're expecting the user to enter numbers, you can use:

```
android:inputType="number"
```

to provide them with a number keypad. Here are some more of our favorites:

You can find the entire list in the online Android developer documentation at https://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType.

Value	What it does
phone	Provides a phone number keypad.
textPassword	Displays a text entry keypad, and your input is concealed.
textCapSentences	Capitalizes the first word of a sentence.
textAutoCorrect	Automatically corrects the text being input.

You can specify multiple input types using the `|` character. As an example, to capitalize the first word of a sentence and automatically correct any misspellings, you'd use:

```
android:inputType="textCapSentences|textAutoCorrect"
```

Using it in your activity code

You can retrieve the text entered in an editable text view like this:

```
EditText editText = (EditText) findViewById(R.id.edit_text);
String text = editText.getText().toString();
```

Button

Buttons are usually used to make your app do something when they're clicked.

Defining it in XML

You define a button in XML using the `<Button>` element. You use the `android:text` attribute to say what text you want the button to display:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text" />
```

Using it in your activity code

You get the button to respond to the user clicking it by using the `android:onClick` attribute in the layout XML, and setting it to the name of the method you want to call in your activity code:

```
android:onClick="onButtonClicked"
```

You then define the method in your activity like this:

```
/** Called when the button is clicked */
public void onButtonClicked(View view) {
    // Do something in response to button click
}
```

Toggle button

A toggle button allows you to choose between two states by clicking a button.

This is what the toggle button looks like when it's off.

When you click the toggle button, it changes to being on.

Defining it in XML

You define a toggle button in XML using the `<ToggleButton>` element. You use the `android:textOn` and `android:textOff` attributes to say what text you want the button to display depending on the state of the button:

```
<ToggleButton
    android:id="@+id/toggle_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```

A compound button is a button with two states, checked and unchecked. A toggle button is an implementation of a compound button.

Using it in your activity code

You get the toggle button to respond to the user clicking it by using the `android:onClick` attribute in the layout XML. You give it the name of the method you want to call in your activity code:

```
android:onClick="onToggleButtonClicked"
```

This is exactly the same as calling a method when a normal button gets clicked.

You then define the method in your activity like this:

```
/** Called when the toggle button is clicked */
public void onToggleButtonClicked(View view) {
    // Get the state of the toggle button.
    boolean on = ((ToggleButton) view).isChecked();
    if (on) {
        // On
    } else {
        // Off
    }
}
```

This returns true if the toggle button is on, and false if the toggle button is off.

Switch

A switch is a slider control that acts in the same way as a toggle button.

Defining it in XML

You define a switch in XML using the `<Switch>` element. You use the `android:textOn` and `android:textOff` attributes to say what text you want the switch to display depending on the state of the switch:

```
<Switch
    android:id="@+id/switch_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```

Using it in your activity code

You get the switch to respond to the user clicking it by using the `android:onClick` attribute in the layout XML, and setting it to the name of the method you want to call in your activity code:

```
android:onClick="onSwitchClicked"
```

You then define the method in your activity like this:

```
/** Called when the switch is clicked */
public void onSwitchClicked(View view) {
    // Is the switch on?
    boolean on = ((Switch) view).isChecked();

    if (on) {
        // On
    } else {
        // Off
    }
}
```

This code is very similar to that used with the toggle button.

Checkboxes

Checkboxes let you display multiple options to users. They can then select whichever options they want. Each of the checkboxes can be checked or unchecked independently of any others.

Here we have two checkboxes. Users can choose milk, sugar, both milk and sugar, or neither.

Defining them in XML

You define each checkbox in XML using the `<CheckBox>` element. You use the `android:text` attribute to display text for each option:

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/milk" />

<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sugar" />
```

Using them in your activity code

You can find whether a particular checkbox is checked using the `isChecked()` method. It returns `true` if the checkbox is checked:

```
CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox_milk);
boolean checked = checkbox.isChecked();
if (checked) {
    //do something
}
```

Checkboxes (continued)

Just like buttons, you can respond to the user clicking a checkbox by using the `android:onClick` attribute in the layout XML, and setting it to the name of the method you want to call in your activity code:

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/milk"
    android:onClick="onCheckboxClicked"/>

<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sugar"
    android:onClick="onCheckboxClicked"/>
```

In this case, the onCheckboxClicked() method will get called no matter which checkbox gets clicked. We could have specified a different method for each checkbox if we'd wanted to.

You then define the method in your activity like this:

```
public void onCheckboxClicked(View view) {
    // Has the checkbox that was clicked been checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Retrieve which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_milk:
            if (checked)
                // Milky coffee
            else
                // Black as the midnight sky on a moonless night
            break;
        case R.id.checkbox_sugar:
            if (checked)
                // Sweet
            else
                // Keep it bitter
            break;
    }
}
```

Radio buttons

These let you display multiple options to the user. The user can select a single option.

Use radio buttons to restrict the user's choice to just one option.

Defining them in XML

You start by defining a radio group, a special type of view group, using the `<RadioGroup>` tag. Within this, you then define individual radio buttons using the `<RadioButton>` tag:

```
<RadioGroup android:id="@+id/radio_group"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton android:id="@+id/radio_cavemen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cavemen" />

    <RadioButton android:id="@+id/radio_astronauts"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/astronauts" />
</RadioGroup>
```

You can choose to display the radio buttons in a horizontal or vertical list.

Using them in your activity code

You can find which radio button is selected using the `getCheckedRadioButtonId()` method:

```
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
int id = radioGroup.getCheckedRadioButtonId();
if (id == -1){
    //no item selected
}
else{
    RadioButton radioButton = findViewById(id);
}
```

Radio buttons (continued)

You can respond to the user clicking a radio button by using the `android:onClick` attribute in the layout XML, and setting it to the name of the method you want to call in your activity code:

```
<RadioGroup android:id="@+id/radio_group"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton android:id="@+id/radio_cavemen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cavemen"
        android:onClick="onRadioButtonClicked" />

    <RadioButton android:id="@+id/radio_astronauts"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/astronauts"
        android:onClick="onRadioButtonClicked" />
</RadioGroup>
```

The radio group containing the radio buttons is a subclass of LinearLayout. You can use the same attributes with a radio group as you can with a linear layout.

You then define the method in your activity like this:

```
public void onRadioButtonClicked(View view) {
    RadioGroup radioGroup = (RadioGroup) findViewById(R.id.radioGroup);
    int id = radioGroup.getCheckedRadioButtonId();
    switch(id) {
        case R.id.radio_cavemen:
            // Cavemen win
            break;
        case R.id.radio_astronauts:
            // Astronauts win
            break;
    }
}
```

Spinner

As you've already seen, a spinner gives you a drop-down list of values from which only one can be selected.

We saw spinners back in Chapter 2.

An AdapterView is a view that can use an adapter. You'll find out about adapters later in the book.

This is an abstract base class for Spinner widgets.

Defining it in XML

You define a spinner in XML using the `<Spinner>` element. You add a static array of entries to the spinner by using the `android:entries` attribute and setting it to an array of Strings.

There are other ways of populating the spinner, which you'll see later in the book.

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/spinner_values" />
```

You can add an array of Strings to *strings.xml* like this:

```
<string-array name="spinner_values">
    <item>light</item>
    <item>amber</item>
    <item>brown</item>
    <item>dark</item>
</string-array>
```

Using it in your activity code

You can get the value of the currently selected item by using the `getSelectedItem()` method and converting it to a `String`:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
String string = String.valueOf(spinner.getSelectedItem());
```

Image view

You use an image view to display an image:

An image view contains an image.

The ImageView class is a direct subclass of View.

Adding an image to your project

You first need to create a *drawable* resource folder, the default folder for storing image resources in your app. To do this, select the *app/src/main/res* folder in your project, go to the File menu, choose the New... option, then click on the option to create a new Android resource directory. When prompted, choose a resource type of "drawable", name the folder "drawable", and click on OK. You then need to add your image to the *app/src/main/res/drawable* folder.

If you want, you can use different image files depending on the screen density of the device. This means you can display higher-resolution images on higher-density screens, and lower-resolution images on lower-density screens. To do this, you create different *drawable* folders in *app/src/main/res* for the different screen densities. The name of the folder relates to the screen density of the device:

`drawable-ldpi`	Low-density screens, around 120 dpi.
`drawable-mdpi`	Medium-density screens, around 160 dpi.
`drawable-hdpi`	High-density screens, around 240 dpi.
`drawable-xhdpi`	Extra-high-density screens, around 320 dpi.
`drawable-xxhdpi`	Extra-extra-high-density screens, around 480 dpi.
`drawable-xxxhdpi`	Extra-extra-extra high-density screens, around 640 dpi.

Depending on what version of Android Studio you're running, the IDE may create some of these folders for you automatically.

You then put different resolution images in each of the *drawable** folders, making sure that each of the image files has the same name. Android decides which image to use at runtime, depending on the screen density of the device it's running on. As an example, if the device has an extra-high-density screen, it will use the image located in the *drawable-xhdpi* folder.

If an image is added to just one of the folders, Android will use the same image file for all devices. If you want your app to use the same image regardless of screen density, you'd normally put it in the *drawable* folder.

Image view: the layout XML

You define an image view in XML using the `<ImageView>` element. You use the `android:src` attribute to specify what image you want to display, and the `android:contentDescription` attribute to add a String description of the image so that your app is more accessible:

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />
```

The `android:src` attribute takes a value of the form `"@drawable/image_name"`, where `image_name` is the name of the image (without its extension). Image resources are prefixed with `@drawable`, which tells Android that it's an image resource located in one or more of the *drawable** folders.

Using image views in your activity code

You can set the image source and description in your activity code using the `setImageResource()` and `setContentDescription()` methods:

```
ImageView photo = (ImageView)findViewById(R.id.photo);
int image = R.drawable.starbuzz_logo;
String description = "This is the logo";
photo.setImageResource(image);
photo.setContentDescription(description);
```

This code looks for the image resource called `starbuzz_logo` in the *drawable** folders, and sets it as the source of an image view with an ID of `photo`. When you need to refer to an image resource in your activity code, you use `R.drawable.image_name` where `image_name` is the name of the image (without its extension).

Adding images to buttons

In addition to displaying images in image views, you can also display images on buttons.

Displaying text and an image on a button

To display text on a button with an image to the right of it, use the `android:drawableRight` attribute and specify the image to be used:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableRight="@drawable/android"
    android:text="@string/click_me" />
```

Display the android image resource on the right side of the button.

If you want to display the image on the left, use the `android:drawableLeft` attribute:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableLeft="@drawable/android"
    android:text="@string/click_me" />
```

You can also use drawableStart and drawableEnd to support right-to-left languages.

Use the `android:drawableBottom` attribute to display the image underneath the text:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableBottom="@drawable/android"
    android:text="@string/click_me" />
```

The `android:drawableTop` attribute displays the image above the text:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableTop="@drawable/android"
    android:text="@string/click_me" />
```

Image button

An image button is just like a button, except it contains an image and no text.

Defining it in XML

You define an image button in XML using the `<ImageButton>` element. You use the `android:src` attribute to say what image you want the image button to display:

```
<ImageButton
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon" />
```

The ImageButton class extends the ImageView class, not the Button class.

Using it in your activity code

You get the image button to respond to the user clicking it by using the `android:onClick` attribute in the layout XML, and setting it to the name of the method you want to call in your activity code:

```
android:onClick="onButtonClicked"
```

You then define the method in your activity like this:

```
/** Called when the image button is clicked */
public void onButtonClicked(View view) {
    // Do something in response to button click
}
```

Scroll view

If you add lots of views to your layouts, you may have problems on devices with smaller screens—most layouts don't come with scrollbars to allow you to scroll down the page. As an example, when we added seven large buttons to a linear layout, we couldn't see all of them.

To add a vertical scrollbar to your layout, you surround your existing layout with a **<ScrollView>** element like this:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.hfad.views.MainActivity" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="16dp"
        android:paddingLeft="16dp"
        android:paddingRight="16dp"
        android:paddingTop="16dp"
        android:orientation="vertical" >
    ...
    </LinearLayout>
</ScrollView>
```

Move these attributes from the original layout to the <ScrollView>, as the <ScrollView> is now the root element.

To add a horizontal scrollbar to your layout, wrap your existing layout inside a **<HorizontalScrollView>** element instead.

Toasts

There's one final widget we want to show you in this chapter: a toast. A toast is a simple pop-up message you can display on the screen.

Toasts are purely informative, as the user can't interact with them. While a toast is displayed, the activity stays visible and interactive. The toast automatically disappears when it times out.

A toast isn't actually a type of view. It's a useful way of displaying a short message to the user, though, so we're sneaking it into this chapter.

Using it in your activity code

You create a toast using activity code only. You can't define one in your layout.

To create a toast, you call the `Toast.makeText()` method, and pass it three parameters: a `Context` (usually `this` for the current activity), a `CharSequence` that's the message you want to display, and an `int` duration. Once you've created the toast, you call its `show()` method to display it.

Here's the code you would use to create a toast that appears on screen for a short duration:

```
CharSequence text = "Hello, I'm a Toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(this, text, duration);
toast.show();
```

A toast is a message that pops up like toast in a toaster.

Exercise

It's time for you to try out some of the views we've introduced you to this chapter. Create a layout that will create this screen:

You probably won't want to write the code here, but why not experiment in the IDE?

Exercise Solution

Here's one of the many ways in which you can create the layout. Don't worry if your code looks different, as there are many different solutions.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.layoutexamples.MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="How do you like your tea served?" />

    <ToggleButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="Hot"
        android:textOff="Cold" />
```

We used a toggle button to display whether the drink should be served hot or cold.

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Milk" />

<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sugar" />

<CheckBox android:id="@+id/checkbox_lemon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Lemon" />

</LinearLayout>
```

We used a checkbox for each of the values (Milk, Sugar, and Lemon). We put each one on a separate row.

Remember, your code may look different from ours. This is just one way of building the layout.

Your Android Toolbox

You've got Chapter 5 under your belt and now you've added views and view groups to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- GUI components are all types of view. They are all subclasses of the `android.view.View` class.
- All layouts are subclasses of the `android.view.ViewGroup` class. A view group is a type of view that can contain multiple views.
- The layout XML file gets converted to a `ViewGroup` containing a hierarchical tree of views.
- A linear layout lists views either horizontally or vertically. You specify the direction using the `android:orientation` attribute.
- A frame layout stacks views.
- Use `android:padding*` attributes to specify how much padding you want around a view.
- In a linear layout, use `android:layout_weight` if you want a view to use up extra space in the layout.
- `android:layout_gravity` lets you say where you want views to appear in their available space.
- `android:gravity` lets you say where you want the contents to appear inside a view.
- `<ToggleButton>` defines a toggle button that allows you to choose between two states by clicking a button.
- `<Switch>` defines a switch control that behaves in the same way as a toggle button. It requires API level 14 or above.
- `<CheckBox>` defines a checkbox.
- To define a group of radio buttons, first use `<RadioGroup>` to define the radio group. Then put individual radio buttons in the radio group using `<RadioButton>`.
- Use `<ImageView>` to display an image.
- `<ImageButton>` defines a button with no text, just an image.
- Add scrollbars using `<ScrollView>` or `<HorizontalScrollView>`.
- A `Toast` is a pop-up message.

6 constraint layouts

Put Things in Their Place

Let's face it, you need to know how to create great layouts.

If you're building apps you want people to ***use***, you need to make sure they **look exactly the way you want**. So far you've seen how to use linear and frame layouts, but *what if your design is more complex?* To deal with this, we'll introduce you to Android's new **constraint layout**, a type of layout you ***build visually using a blueprint***. We'll show you how **constraints** let you position and size your views, *irrespective of screen size and orientation.* Finally, you'll find out how to save time by making Android Studio **infer and add constraints** on your behalf.

Nested layouts can be inefficient

You've seen how to build a simple user interface using linear layouts and frame layouts, but what if you need to create something more complex? While complex UIs are possible if you nest your linear and frame layouts deep enough, they can slow down your app and make your code hard to read and maintain.

As an example, suppose you wanted to create a layout containing two rows, each containing two items. One possibility would be to create this layout using three linear layouts: one linear layout at the root, and one nested linear layout for each row:

Linear layouts only allow you to display views in a single column or row, so you can't use a single linear layout to construct layouts like this. You can, however, nest linear layouts to produce the results you need.

For this layout, you could use one linear layout at the root, and one nested linear layout for each row.

When Android displays a layout on the device screen, it creates a hierarchy of views based on the layout components which helps it figure out where each view should be placed. If the layout contains nested layouts, the hierarchy is more complex, and Android may need to make more than one pass through the hierarchy:

This is the root linear layout.

This is for the first row.

This is for the second row.

Each view and layout needs to be initialized, measured, laid out, and drawn. This is a lot of work if you have deeply nested layouts, and it can slow down your app.

While the above hierarchy is still relatively simple, imagine if you needed more views, more nested layouts, and a deeper hierarchy. This could lead to bottlenecks in your app's performance, and leave you with a mass of code that's difficult to read and maintain.

If you have a more complex UI that requires you to nest multiple layouts, it's usually better to **use a different type of layout**.

Introducing the constraint layout

In this chapter we're going to focus on using a new type of layout called a **constraint layout**. This type of layout is more complex than a linear or frame layout, but it's a lot more flexible. It's also much more efficient for complex UIs as it gives you a flatter view hierarchy, which means that Android has less processing to do at runtime.

You design constraint layouts VISUALLY

Another advantage of using constraint layouts is that they're specifically designed to work with Android Studio's design editor. Unlike linear and frame layouts where you usually hack direct in XML, you build constraint layouts *visually*. You drag and drop GUI components onto the design editor's blueprint tool, and give it instructions for how each view should be displayed.

To see this in action, we're going to take you on a tour of using a constraint layout, then build the following UI:

To build constraint layouts using the visual tools, you need Android Studio 2.3 or above. If you're using an older version, check for updates.

Create a new project

We'll start by creating a new Android Studio project for an application named "My Constraint Layout" with a company domain of "hfad.com", making the package name `com.hfad.myconstraintlayout`. The minimum SDK should be API 19 so that it will work on most devices. You'll need an empty activity called "MainActivity" with a layout called "activity_main" so that your code matches ours. **Make sure you uncheck the Backwards Compatibility (AppCompat) option when you create the activity.**

Make sure your project includes the Constraint Layout Library

Unlike the other layouts you've seen so far, constraint layouts come in their own library, which you need to add to your project as a dependency before you can use it. Adding a library as a dependency means that the library gets included in your app, and downloaded to the user's device.

It's likely that Android Studio added the Constraint Layout Library to your project automatically, but let's check. In Android Studio, choose File→Project Structure. Then click on the app module and choose Dependencies. You'll be presented with the following screen:

If Android Studio has already added the Constraint Layout Library for you, you will see it listed as "com.android.support.constraint:constraint-layout," as shown above.

If the library hasn't been added for you, you will need to add it yourself. To do this, click on the "+" button at the bottom or right side of the Project Structure screen. Choose the Library Dependency option, and select the Constraint Layout Library option from the list. If you don't see it listed, type the following text into the search box:

```
com.android.support.constraint:constraint-layout:1.0.2
```

You only need to type this in if the Constraint Layout Library hasn't already been added to your project as a dependency.

When you click on the OK button, the Constraint Layout Library should be added to the list of dependencies. Click on OK again to save your changes and close the Project Structure window.

Now that we know that our project contains the Constraint Layout Library, let's add the String resources we'll need for our layout.

Add the String resources to strings.xml

Each of the views in our layout will display text values or hints, so we'll add these as String resources. Add the String values below to *strings.xml*:

```
...
    <string name="to_label">To:</string>
    <string name="email_hint">Enter email address</string>
    <string name="subject_hint">Subject</string>
    <string name="message_hint">Message</string>
    <string name="send_button">Send</string>
...
```

Now that we've added our String resources, we'll update the layout.

Change activity_main.xml to use a constraint layout

We're going to use a constraint layout. To do this (and to make sure that your layout matches ours), update your code in *activity_main.xml* to match the code below (our changes are in bold):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.hfad.myconstraintlayout.MainActivity">

</android.support.constraint.ConstraintLayout>
```

This is how you define a constraint layout.

If Android Studio added any extra views for you, delete them.

This defines a constraint layout to which we can add views. We'll do this using the design editor's blueprint tool.

Use the blueprint tool

To use the blueprint tool, first switch to the design view of your layout code by clicking on the Design tab. Then click on the Show Blueprint button in the design editor's toolbar to display the blueprint. Finally, drag a Button widget from the design editor's palette to the blueprint. This adds a button to your layout:

Position views using constraints

With a constraint layout, you don't specify where views should be positioned by dropping them on the blueprint in a particular place. Instead, you specify placement by defining **constraints**. A **constraint** is a connection or attachment that tells the layout where the view should be positioned. For example, you can use a constraint to attach a view to the start edge of the layout, or underneath another view.

We'll add a horizontal constraint to the button

To see how this works, let's add a constraint to attach our button to the left edge of the layout.

First, make sure the button's selected by clicking it. When you select a view, a bounding box is drawn around it, and handles are added to its corners and sides. The square handles in the corners let you resize the view, and the round handles on the sides let you add constraints:

To add a constraint, you click on one of the view's constraint handles and drag it to whatever you want to attach it to. In our case, we're going to attach the left edge of the button to the left edge of the layout, so click on the left constraint handle and drag it to the left edge of the blueprint:

This adds the constraint, and pulls the button over to the left:

That's how you add a horizontal constraint. We'll add a vertical constraint to the button next.

Add a vertical constraint

Let's add a second constraint to the button to attach it to the top of the layout. To do this, click on the button's top constraint handle, and drag it to the top of the blueprint's main area. This adds the second constraint, and the button slides to the top of the main area.

Each view in a constraint layout must have at least two constraints—a horizontal constraint and a vertical one—in order to specify where it should be positioned. If you omit the horizontal constraint, the view is displayed next to the start edge of the layout at runtime. If you omit the vertical constraint, the view is displayed at the top of the layout. **This is irrespective of where the view is positioned in the blueprint.**

Click on the round handle on the button's top edge, and drag it to the top of the blueprint.

The button slides to the top of the blueprint's main area.

Changing the view's margins

When you add a constraint to a view, the design editor automatically adds a margin on the same edge as the constraint. You can set the size of the default margin in the design editor's toolbar by changing the number in the Default Margin box:

Change the number here (in dps) to change the default margin.

Changing the size of the default margin specifies the size of any *new* margins that get added. The size of any existing margins remain unchanged, but you can change these using the property window.

The property window is displayed in a separate panel at the side of the design editor. When you select a view, it displays a diagram featuring the view's constraints and the size of its margins. To change the size of a margin, you change the number next to the relevant side of the view.

You can also change the size of a view's margins by clicking and dragging the view in the blueprint. This technique has the same effect as changing the size of the margin in the property window, but it's harder to be precise.

Try changing the size of both margins using each method before looking at the next page.

Changes to the blueprint are reflected in the XML

When you add views to the blueprint and specify constraints, these changes are reflected in the layout's underlying XML. To see this, switch to the text view of your code. Your code should look something like this (but don't worry if it's slightly different):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    ...>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="32dp"
        android:text="Button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

The design editor's added a button.

These are all attributes you've seen before.

These lines only apply to constraint layouts.

As you can see, our XML now includes a button. Most of the code for the button should look familiar to you, as it's material we covered in Chapter 5. The button's width, height, and margins are specified in exactly the same way as before. The only unfamiliar code is the two lines that specify the view's constraints on its left and top edges:

```
<Button>
    ...
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

These lines describe the constraints on the button's left and top edges.

Similar code is generated if you add constraints to the button's remaining edges.

Next, switch your code back to design view, and we'll look at other techniques for positioning your views in a constraint layout.

How to center views

So far you've seen how you can use constraints to attach a view to the edge of its layout. This works well if you want to position a view in the top-left corner, for example, but what if you want to position it in the center?

To position views in the center of its layout, you add constraints to opposite sides of the view. Let's see how this works by centering our button horizontally.

Make sure the button is selected, then click on the constraint handle on its right edge, and drag it to the right edge of the layout:

Click on the constraint handle on the button's right edge, and drag it to the right edge of the blueprint.

This adds a constraint to the view's right edge. As the button now has two horizontal constraints, one on each side, the button is pulled to the center, and the two opposing constraints are displayed in the blueprint as springs:

Constraints on opposite sides of a view are displayed as springs.

As the button is now attached to both sides of the layout, it's displayed in the center irrespective of screen size or orientation. You can experiment with this by running the app, or changing the size of the blueprint by dragging the blueprint's bottom-right corner:

You can resize the blueprint by clicking and dragging its bottom-right corner.

Adjust a view's position by updating its bias

Once you've added constraints to opposite sides of your view, you can control where it should be positioned relative to each side by changing its **bias**. This tells Android what the proportionate length of the constraint should be on either side of the view.

To see this in action, let's change our button's horizontal bias so that it's positioned off-center. First, make sure the button's selected, then look in the view's property window. Underneath the diagram of the view, you should see a slider with a number in it. This represents the view's horizontal bias as a percentage:

To change the value of the bias, simply move the slider. If you move the slider to the left, for example, it moves the button in the blueprint to the left too:

The view maintains this relative position irrespective of screen size and orientation.

When you add a bias to a view in the design editor, this is reflected in the underlying XML. If you change the horizontal bias of your view to 25%, for example, the following code gets added to the view's XML:

```
app:layout_constraintHorizontal_bias="0.25"
```

Now that you know how bias works, let's look at how you specify a view's size.

How to change a view's size

With a constraint layout, you have several different options for specifying a view's size:

- Make it a fixed size by specifying a specific width and height.
- Use `wrap_content` to make the view just big enough to display its contents.
- Tell it to match the size of its constraints (if you've added constraints to opposite sides of the view).
- Specify a ratio for the width and height so that, for example, the view's width is twice the size of its height.

We'll go through these options one-by-one.

1. Make the view a fixed size

There are a couple of ways of using the design editor to make the view a fixed size. One way is to simply resize the view in the blueprint by clicking and dragging the square resizing handles on its corners. The other way is to type values into the `layout_width` and `layout_height` fields in the properties window:

You can resize a view using the square resizing handles on its corners.

You can also hardcode the width and height in the view's property window.

In general, **making your view a fixed size is a bad idea**, as it means the view can't grow or shrink to fit the size of its contents or the size of the screen.

2. Make it just big enough

To make the view just large enough to display its contents, change the view's `layout_width` and `layout_height` properties to `wrap_content`. You do this in the view's property window as shown here:

Setting the width and height to "wrap_content" makes it just large enough to display its contents, just as it does in other layouts.

3. Match the view's constraints

If you've added constraints to opposite sides of your view, you can make the view as wide as its constraints. You do this by setting its width and/or height to 0dp: set its width to 0dp to get the view to match the size of its horizontal constraints, and set its height to 0dp to get it to match the size of its vertical constraints.

In our case, we've added constraints to the left and right sides of our button, so we can get the button to match the size of these constraints. To do this, go to the view's property window, and change the `layout_width` property to 0dp. In the blueprint, the button should expand to fill the available horizontal space (allowing for any margins):

4. Specify the width:height ratio

Finally, you can specify an aspect ratio for the view's width and height. To do this, change the view's `layout_width` *or* `layout_height` to 0dp as you did above, then click in the top-left corner of the view diagram that's displayed in the property window. This should display a ratio field, which you can then update:

Now that you've seen how to resize a view, try experimenting with the different techniques before having a go at the exercise on the next page.

BE the Constraint

Your job is to play like you're the constraint layout and draw the constraints that are needed to produce each layout. You also need to specify the layout_width, layout_height, and bias (when needed) for each view. We've completed the first one for you.

You need to add the views and constraints to each blueprint.

A

This is how we want the screen to look.

layout_width: wrap_content
layout_height: wrap_content

B

The button fills the available space.

C

D

BE the Constraint Solution

Your job is to play like you're the constraint layout and draw the constraints that are needed to produce each layout. You also need to specify the layout_width, layout_height, and bias (when needed) for each view. We've completed the first one for you.

A

This is how we want the screen to look.

layout_width: wrap_content
layout_height: wrap_content

B

layout_width: wrap_content
layout_height: wrap_content
bias: 50%

The button fills the available space.

C

The button needs to stretch in all directions, so it requires constraints on all edges, and its width and height need to be set to 0dp.

layout_width: 0dp
layout_height: 0dp

D

Button 1 is displayed in the top-left corner. Button 2 fills the remaining horizontal space.

Button 1:
layout_width: wrap_content
layout_height: wrap_content

Button 2:
layout_width: 0dp
layout_height: wrap_content

To make Button 2 fill the remaining horizontal space, we add constraints to each vertical edge and set its width to 0dp. Its left edge is attached to Button 1's right edge.

How to align views

So far you've seen how to position and size a single view. Next, let's examine how you align it with another view.

First, click on the Show Constraints button in the design edit toolbar to display all the constraints in the blueprint (not just the ones for the selected view). Then drag a second button from the palette to the blueprint, and place it underneath the first:

This is the Show Constraints button. Clicking on it shows (or hides) all the constraints in the layout.

Add a second button to the blueprint, underneath the first.

To display the second button underneath the first when the app runs, we need to add a constraint to the second button's top edge, and attach it to the first button's bottom edge. To do this, select the second button, and draw a constraint from its top edge to the bottom edge of the other button:

This adds a constraint attaching the top of one button to the bottom edge of the other.

To align the left edges of both buttons, select both buttons by holding down the Shift key when you select each one, then click on the Align Left Edges button in the design editor toolbar:

Clicking on this button gives you different options for aligning views.

Aligning the view's left edges adds another constraint.

This adds a constraint from the left edge of the second button to the left edge of the first, and this constraint aligns the view's edges.

Let's build a real layout

You now know enough about constraint layouts to start building a real one. Here's the layout we're going to create:

We'll build it from scratch in *activity_main.xml*, so before we get started, delete any views that are already in the layout so that the blueprint's empty, and make sure your *activity_main.xml* code looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.hfad.myconstraintlayout.MainActivity">

</android.support.constraint.ConstraintLayout>
```

First, add the top line of views

We're going to start by adding the views we want to appear at the top of the layout: a text view and an editable text field.

To do this, switch to the design editor if you haven't already done so, then drag a TextView from the palette to the top-left corner of the blueprint. Next, drag an E-mail component to the blueprint so it's positioned to the right of the text view. This is an editable text field that uses Android's email keyboard for data entry. Manually resize the E-mail component so that it lines up with the text view and fills any remaining horizontal space:

The top line of the layout features a TextView label and an EditText for the email address.

Notice that we haven't added any constraints to either view yet, and we've positioned them where we want them to appear when the layout's displayed on a device. There's a good reason for this: to save us some work, **we're going to get the design editor to figure out the constraints.**

Get the design editor to infer the constraints

As you already know, a constraint layout uses constraints to determine where its views should be positioned. The great news is that the design editor has an Infer Constraints button that's designed to work out what it thinks the constraints should be, and add them. To use this feature, simply click on the Infer Constraints button in the design editor's toolbar:

This is the Infer Constraints button. Click it now.

The Infer Constraints feature guesses which constraints to add

When you click on the Infer Constraints button, the design editor tries to figure out what the constraints should be and adds them for you. It's not completely foolproof, as it can't read your mind (as far as we know) to determine how you want the layout to behave on a real device. It simply guesses based on each view's position in the blueprint.

Here are the changes the design editor made for us when we clicked on the Infer Constraints button (yours may look different if you positioned your views differently):

Clicking on the Infer Constraints button added constraints to both views.

You can check the details of each constraint by selecting each view in turn and looking at its values in the property window.

If you don't like what the Infer Constraints feature has done, you can undo the changes it's made by choosing Undo Infer Constraints from the Edit menu, or adjust individual constraints.

We're going to tweak our views before adding more items to the blueprint. First, select the text view in the blueprint, then edit its properties in the properties panel to give it an ID of `to_label` and a text value of `"@string/to_label"`. This does the same thing as adding the following lines of code to the `<TextView>` element in the XML:

When you update the ID, don't worry if Android Studio displays a message telling you it's going to make changes to the code. We want it to do this because we're changing the view's ID.

```
android:id="@+id/to_label"
android:text="@string/to_label"
```

Android Studio adds these lines of code when you change the view's ID and text value.

Update this property to change the TextView's text value.

Next, select the E-mail component `EditText`, and change its ID to `email_address`, its `layout_height` to `"wrap_content"`, and its hint to `"@string/email_hint"`. This does the same thing as adding these lines to the `<EditText>` element in the XML:

```
android:id="@+id/email_address"
android:layout_height="wrap_content"
android:hint="@string/email_hint"
```

Android Studio adds these lines of code when you change the view's layout_height and hint value.

Now that we've added the first line of views to the blueprint, let's add some more.

Add the next line to the blueprint...

The next row of the layout contains an editable text field for the message subject. Drag a Plain Text component from the palette to the blueprint, and position it underneath the two items we've already added. This adds an `EditText` to the blueprint. Then change the component's size and position so that it lines up with the other views and fills the horizontal space:

The Plain Text component adds an EditText to the layout.

Then click on the Infer Constraints button again. The design editor adds more constraints, this time positioning the new component:

The design editor adds constraints to the new EditText when we click on the Infer Constraints button.

Select the new view in the blueprint, and then change its ID to `subject`, its `layout_height` to `"wrap_content"`, and its hint to `"@string/subject_hint"`, and delete any text in the `text` property that the design editor may have added.

...and then add the button

The button goes at the bottom of the layout, centered horizontally.

Next, we'll add a button to the bottom of the layout. Drag a Button component to the bottom of the blueprint and center it horizontally. When you click on the Infer Constraints button this time, the design editor adds these constraints to it:

Remember, when you click on the Infer Constraints button in your layout, it may give you different results than shown here.

Change the button's ID to `send_button` and its text to `"@string/send_button"`.

Finally, add a view for the message

We have one more view to add to our layout, an editable text field that we want to be able to grow to fill any remaining space. Drag a Plain Text component from the palette to the middle of the blueprint, change its size so that it fills the entire area, and then click on the Infer Constraints button:

Note that we could have added all these views in one go, and clicked the Infer Constraints button when we reached the end. We've found, however, that building it up step-by-step gives the best results. Why not experiment and try this out for yourself?

Select the new component in the blueprint, then change its ID to `message`, its hint to `"@string/message_hint"`, and its gravity to `top`, and delete any text in the `text` property that the design editor may have added.

You may need to click on the "View all properties" button to see the gravity property.

Let's take the app for a test drive and see what the layout looks like.

Test drive the app

When we run the app, `MainActivity`'s layout looks almost exactly how we want it to. When we rotate the device, the button stays centered, the email and subject editable text fields expand to fill the horizontal space, and the message view fills the remaining area:

Test your constraint layout on a variety of device sizes and orientations to make sure it behaves the way you want. If it doesn't, you may need to change the properties of some of your views and their constraints.

Remember that your layout may look and behave differently than ours depending on what constraints the design editor added when you clicked on the Infer Constraints button. The feature isn't perfect, but it usually takes you most of the way there, and you can undo or update any changes it makes.

Your Android Toolbox

You've got Chapter 6 under your belt and now you've added constraint layouts to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Constraint layouts are designed to work with Android Studio's design editor. They have their own library and can be used in apps where the minimum SDK is API level 9 or above.
- Position views by adding constraints. Each view needs at least one horizontal and one vertical constraint.
- Center views by adding constraints to opposite sides of the view. Change the view's bias to update its position between the constraints.
- You can change a view's size to match its constraints if the view has constraints on opposing sides.
- You can specify a width:height aspect ratio for the view's size.
- Clicking on the Infer Constraints button adds constraints to views based on their position in the blueprint.

there are no Dumb Questions

Q: Is a constraint layout my only option if I want to create complex layouts?

A: There are other types of layout as well, such as relative and grid layouts, but the constraint layout does everything that these do. Also it's designed to work with Android Studio's design editor, which makes building constraint layouts much easier.

If you're interested in finding out more about relative and grid layouts, they're covered in Appendix I at the back of the book.

Q: Why do constraint layouts have a separate library?

A: Constraint layouts are a fairly recent addition to Android compared to other types of layout. They're in a separate library so that they can be added to apps that support older versions of Android. You'll find out more about backward compatibility in later chapters.

Q: Can I still edit constraint layouts using XML?

A: Yes, but as they're designed to be edited visually, we've concentrated on building them using the design editor.

Q: I tried using the Infer Constraints feature but it didn't give me the results I wanted. Why not?

A: The Infer Constraints feature can only make guesses based on where you position views in the blueprint, so it may not always give you the results you want. You can, however, edit the changes the Infer Constraints feature makes to your app.

7 list views and adapters

Getting Organized

Want to know how best to structure your Android app?

You've learned about some of the basic building blocks that are used to create apps, and now **it's time to get organized**. In this chapter, we'll show you how you can take a bunch of ideas and **structure them into an awesome app**. You'll learn how **lists of data** can form the core part of your app design, and how **linking them together** can create a **powerful and easy-to-use app**. Along the way, you'll get your first glimpse of using **event listeners** and **adapters** to make your app more dynamic.

Every app starts with ideas

When you first come up with an idea for an app, you'll have lots of thoughts about what the app should contain.

As an example, the guys at Starbuzz Coffee want a new app to entice more customers to their stores. These are some of the ideas they came up with for what the app should include:

These are all ideas that users of the app will find useful. But how do you take all of these ideas and organize them into an intuitive, well-structured app?

Organize your ideas: top-level, category, and detail/edit activities

A useful way to bring order to these ideas is to organize them into three different types of activity: **top-level** activities, **category** activities, and **detail/edit** activities.

Top-level activities

A top-level activity contains the things that are most important to the user, and gives them an easy way of navigating to those things. In most apps, the first activity the user sees will be a top-level activity.

Display a start screen with a list of options.

Category activities

Category activities show the data that belongs to a particular category, often in a list. These type of activities are often used to help the user navigate to detail/edit activities. An example of a category activity is a list of all the drinks available at Starbuzz.

Display a menu showing all the food you can buy.

Show a list of all our stores.

Display a list of the drinks we sell.

Detail/edit activities

Detail/edit activities display details for a particular record, let the user edit the record, or allow the user to enter new records. An example of a detail/edit activity would be an activity that shows the user the details of a particular drink.

Show details of each drink.

Show details of an item of food.

Display the address and opening times of each store.

Once you've organized your activities, you can use them to construct a hierarchy showing how the user will navigate between activities.

Exercise

Think of an app you'd like to create. What activities should it include? Organize these activities into top-level activities, category activities, and detail/edit activities.

Navigating through the activities

When you organize the ideas you have into top-level, category, and detail/edit activities, you can use this organization scheme to figure out how to navigate through your app. In general, you want your users to navigate from top-level activities to detail/edit activities via category activities.

Top-level activities go at the top

These are the activities your user will encounter first, so they go at the top.

Category activities go between top-level and detail/edit activities

Your users will navigate from the top-level activity to the category activities. In complex apps, you might have several layers of categories and subcategories.

Detail/edit activities

These form the bottom layer of the activity hierarchy. Users will navigate to these from the category activities.

As an example, suppose a user wanted to look at details of one of the drinks that Starbuzz serves. To do this, she would launch the app, and be presented with the top-level activity start screen showing her a list of options. The user would click on the option to display a list of drinks. To see details of a particular drink, she would then click on her drink of choice in the list.

Use list views to navigate to data

When you structure your app in this way, you need a way of navigating between your activities. A common approach is to use **list views**. A list view allows you to display a list of data that you can then use to navigate through the app.

As an example, on the previous page, we said we'd have a category activity that displays a list of the drinks sold by Starbuzz. Here's what the activity might look like:

This is a list view containing a list of drinks.

The activity uses a list view to display all the drinks that are sold by Starbuzz. To navigate to a particular drink, the user clicks on one of the drinks, and the details of that drink are displayed.

If you click on the Latte option in the list view, you get shown the details for the latte.

We're going to spend the rest of this chapter showing you how to use list views to implement this approach, using the Starbuzz app as an example.

We're going to build part of the Starbuzz app

Rather than build all the category and detail/edit activities required for the entire Starbuzz app, **we'll focus on just the drinks.** We're going to build a top-level activity that the user will see when they launch the app, a category activity that will display a list of drinks, and a detail/edit activity that will display details of a single drink.

The top-level activity

When the user launches the app, she will be presented with the top-level activity, the main entry point of the app. This activity includes an image of the Starbuzz logo, and a navigational list containing entries for Drinks, Food, and Stores.

When the user clicks on an item in the list, the app uses her selection to navigate to a separate activity. As an example, if the user clicks on Drinks, the app starts a category activity relating to drinks.

The drink category activity

This activity is launched when the user chooses Drinks from the navigational list in the top-level activity. The activity displays a list of all the drinks that are available at Starbuzz. The user can click on one of these drinks to see more details of it.

The drink detail activity

The drink detail activity is launched when the user clicks on one of the drinks listed by the drink category activity.

This activity displays details of the drink the user has selected: its name, an image of it, and a description.

How the user navigates through the app

The user navigates from the top-level activity to the drink category activity by clicking on the Drinks item in the top-level activity. She then navigates to the drink detail activity by clicking on a drink.

The Starbuzz app structure

The app contains three activities. `TopLevelActivity` is the app's top-level activity and allows the user to navigate through the app. `DrinkCategoryActivity` is a category activity; it contains a list of all the drinks. The third activity, `DrinkActivity`, displays details of a given drink.

For now, we're going to hold the drink data in a Java class. In a later chapter, we're going to move it into a database, but for now we want to focus on building the rest of the app without teaching you about databases too.

Here's how the app will work:

1. **When the app gets launched, it starts activity TopLevelActivity.**
   This activity uses layout *activity_top_level.xml*. The activity displays a list of options for Drinks, Food, and Stores.

2. **The user clicks on Drinks in TopLevelActivity, which launches activity DrinkCategoryActivity.**
   This activity uses layout *activity_drink_category.xml* and displays a list of drinks. It gets information about the drinks from the *Drink.java* class file.

3. **The user clicks on a drink in DrinkCategoryActivity, which launches activity DrinkActivity.**
   The activity uses layout *activity_drink.xml*. This activity also gets details about the drinks from the *Drink.java* class file.

Here's what we're going to do

There are a number of steps we'll go through to build the app:

1. **Add the Drink class and image resources.** This class contains details of the available drinks, and we'll use images of the drinks and Starbuzz logo in the app.

2. **Create TopLevelActivity and its layout.** This activity is the entry point for the app. It needs to display the Starbuzz logo and include a navigational list of options. `TopLevelActivity` needs to launch `DrinkCategoryActivity` when the Drinks option is clicked.

3. **Create DrinkCategoryActivity and its layout.** This activity contains a list of all the drinks that are available. When a drink is clicked, it needs to launch `DrinkCategory`.

4. **Create DrinkActivity and its layout.** This activity displays details of the drink the user clicked on in `DrinkCategoryActivity`.

Create the project

You create the project for the app in exactly the same way you did in the previous chapters.

Create a new Android project for an application named "Starbuzz" with a company domain of "hfad.com", making the package name `com.hfad.starbuzz`. The minimum SDK should be API 19. You'll need an empty activity called "TopLevelActivity" and a layout called "activity_top_level". **Make sure you uncheck the Backwards Compatibility (AppCompat) checkbox.**

- [ ] **Add resources**
- [ ] **TopLevelActivity**
- [ ] **DrinkCategoryActivity**
- [ ] **DrinkActivity**

The Drink class

We'll start by adding the `Drink` class to the app. *Drink.java* is a pure Java class file that activities will get their drink data from. The class defines an array of three drinks, where each drink is composed of a name, description, and image resource ID. Switch to the Project view of Android Studio's explorer, select the *com.hfad.starbuzz* package in the *app/src/main/java* folder, then go to File→New...→Java Class. When prompted, name the class "Drink", and make sure the package name is `com.hfad.starbuzz`. Then replace the code in *Drink.java* with the following, and save your changes.

```
package com.hfad.starbuzz;

public class Drink {
    private String name;
    private String description;
    private int imageResourceId;

    //drinks is an array of Drinks
    public static final Drink[] drinks = {
        new Drink("Latte", "A couple of espresso shots with steamed milk",
                  R.drawable.latte),
        new Drink("Cappuccino", "Espresso, hot milk, and a steamed milk foam",
                  R.drawable.cappuccino),
        new Drink("Filter", "Highest quality beans roasted and brewed fresh",
                  R.drawable.filter)
    };

    //Each Drink has a name, description, and an image resource
    private Drink(String name, String description, int imageResourceId) {
        this.name = name;
        this.description = description;
        this.imageResourceId = imageResourceId;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

    public int getImageResourceId() {
        return imageResourceId;
    }

    public String toString() {
        return this.name;
    }
}
```

Each Drink has a name, description, and image resource ID. The image resource ID refers to drink images we'll add to the project on the next page.

drinks is an array of three Drinks.

These are images of the drinks. We'll add these next.

The Drink constructor

These are getters for the private variables.

The String representation of a Drink is its name.

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

The image files

The `Drink` code includes three image resources for its drinks with IDs of `R.drawable.latte`, `R.drawable.cappuccino`, and `R.drawable.filter`. These are so we can show the user images of the drinks. `R.drawable.latte` refers to an image file called *latte*, `R.drawable.cappuccino` refers to an image file called *cappuccino*, and `R.drawable.filter` refers to a file called *filter*.

We need to add these image files to the project, along with an image of the Starbuzz logo so that we can use it in our top-level activity. First, create the *app/src/main/res/drawable* folder in your Starbuzz project (if it doesn't already exist). To do this, make sure you've switched to the Project view of Android Studio's explorer, select the *app/src/main/res* folder, go to the File menu, choose the New... option, then click on the option to create a new Android resource directory. When prompted, choose a resource type of "drawable", name the folder "drawable", and click on OK.

Android Studio may have already added this folder for you. If so, you don't need to recreate it.

Once your project includes the *drawable* folder, download the files *starbuzz-logo.png*, *cappuccino.png*, *filter.png*, and *latte.png* from *https://git.io/v9oet*. Finally, add each file to the *app/src/main/res/drawable* folder.

When you add images to your apps, you need to decide whether to display different images for different density screens. In our case, we're going to use the same resolution image irrespective of screen density, so we've put a single copy of the images in one folder. If you decide to cater to different screen densities in your own apps, put images for the different screen densities in the appropriate *drawable** folders as described in Chapter 5.

Here are the four image files. You need to create the drawable folder, then add the image files to it.

When you save images to your project, Android assigns each of them an ID in the form `R.drawable.image_name` (where `image_name` is the name of the image). As an example, the file *latte.png* is given an ID of `R.drawable.latte`, which matches the value of the latte's image resource ID in the `Drink` class.

Drink

name: "Latte"

description: "A couple of expresso shots with steamed milk"

imageResourceId: R.drawable.latte

The image latte.png is given an ID of R.drawable.latte.

R.drawable.latte

Now that we've added the `Drink` class and image resources to the project, let's work on the activities. We'll start with the top-level activity.

The top-level layout contains an image and a list

- Add resources
- **TopLevelActivity**
- DrinkCategoryActivity
- DrinkActivity

When we created our project, we called our default activity *TopLevelActivity.java*, and its layout *activity_top_level.xml*. We need to change the layout so it displays an image and a list.

This is the Starbuzz logo. We added this image to the project on the previous page.

We'll add these items as a static list of options; then we'll make each list item clickable.

You saw how to display images in Chapter 5 using an image view. In this case, we need an image view that displays the Starbuzz logo, so we'll create one that uses *starbuzz_logo.png* as its source.

Here's the code to define the image view in the layout:

We're going to add this to activity_top_level.xml. We'll show you the full code soon.

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />
```

These are the dimensions we want the image to have.

The source of the image is the starbuzz_logo.png file we added to the app.

Adding a content description makes your app more accessible.

When you use an image view in your app, you use the `android:contentDescription` attribute to add a description of the image; this makes your app more accessible. In our case, we're using a String value of `"@string/starbuzz_logo"`, so add this to *strings.xml*:

```
<resources>
    ...
    <string name="starbuzz_logo">Starbuzz logo</string>
</resources>
```

That's everything we need to add the image to the layout, so let's move on to the list.

Use a list view to display the list of options

As we said earlier, a list view allows you to display a vertical list of data that people can use to navigate through the app. We're going to add a list view to the layout that displays the list of options, and later on we'll use it to navigate to a different activity.

How to define a list view in XML

You add a list view to your layout using the **`<ListView>`** element. You then add an array of entries to the list view by using the `android:entries` attribute and setting it to an array of Strings. The array of Strings then gets displayed in the list view as a list of text views.

Here's how you add a list view to your layout that gets its values from an array of Strings named `options`:

We're going to add this to activity_top_level.xml on the next page.

```
<ListView
    android:id="@+id/list_options"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/options" />
```

This defines the list view.

The values in the list view are defined by the options array.

You define the array in exactly the same way that you did earlier in the book, by adding it to *strings.xml* like this:

```
<resources>
    ...
    <string-array name="options">
        <item>Drinks</item>
        <item>Food</item>
        <item>Stores</item>
    </string-array>
</resources>
```

This populates the list view with three values: Drinks, Food, and Stores.

The entries attribute populates the list view with values from the options array. Each item in the list view is a text view.

The full top-level layout code

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

Here's the full layout code for our *activity_top_level.xml*; make sure you update your code to match ours:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.starbuzz.TopLevelActivity" >

    <ImageView
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:src="@drawable/starbuzz_logo"
        android:contentDescription="@string/starbuzz_logo" />

    <ListView
        android:id="@+id/list_options"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/options" />
</LinearLayout>
```

We're using a linear layout with a vertical orientation. This will display our list view directly underneath the Starbuzz logo.

Test drive

Make sure you've applied all the changes to *activity_top_level.xml*, and also updated *strings.xml*. When you run the app, you should see the Starbuzz logo displayed on the device screen with the list view underneath it. The list view displays the three values from the `options` array.

If you click on any of the options in the list, nothing happens, as we haven't told the list view to respond to clicks yet. The next thing we'll do is see how you get list views to respond to clicks and launch a second activity.

Get list views to respond to clicks with a listener

You make the items in a list view respond to clicks by implementing an **event listener.**

An event listener allows you to listen for events that take place in your app, such as when views get clicked, when they receive or lose the focus, or when the user presses a hardware key on their device. By implementing an event listener, you can tell when your user performs a particular action—such as clicking on an item in a list view—and respond to it.

The ListView needs to know the Activity cares what happens to it.

The ListView tells the Activity when an item gets clicked so the Activity can react.

OnItemClickListener listens for item clicks

When you want to get items in a list view to respond to clicks, you need to create an `OnItemClickListener` and implement its `onItemClick()` method. The `OnItemClickListener` listens for when items are clicked, and the `onItemClick()` method lets you say how your activity should respond to the click. The `onItemClick()` method includes several parameters that you can use to find out which item was clicked, such as a reference to the view item that was clicked, the item's position in the list view (starting at 0), and the row ID of the underlying data.

We want to start `DrinkCategoryActivity` when the first item in the list view is clicked—the item at position 0. If the item at position 0 is clicked, we need to create an intent to start `DrinkCategoryActivity`. Here's the code to create the listener; we'll add it to *TopLevelActivity.java* on the next page:

OnItemClickListener is a nested class within the AdapterView class. A ListView is a subclass of AdapterView.

```
AdapterView.OnItemClickListener itemClickListener = new AdapterView.OnItemClickListener(){
    public void onItemClick(AdapterView<?> listView,
                            View itemView,
                            int position,
                            long id) {
        if (position == 0) {
            Intent intent = new Intent(TopLevelActivity.this, DrinkCategoryActivity.class);
            startActivity(intent);
        }
    }
};
```

The view that was clicked (in this case, the list view).

These parameters give you info about which item was clicked in the list view, such as the item's view and its position.

Drinks is the first item in the list view, so it's at position 0.

The intent is coming from TopLevelActivity.

It needs to launch DrinkCategoryActivity.

Once you've created the listener, you need to add it to the list view.

Set the listener to the list view

Once you've created the `OnClickItemListener`, you need to attach it to the list view. You do this using the `ListView` `setOnItemClickListener()` method, which takes one argument, the listener itself:

```
AdapterView.OnItemClickListener itemClickListener = new AdapterView.OnItemClickListener(){
    public void onItemClick(AdapterView<?> listView,
        ...
    }
};
ListView listView = (ListView) findViewById(R.id.list_options);
listView.setOnItemClickListener(itemClickListener);
```

We'll add this to TopLevelActivity. The full code is listed on the next couple of pages so you can see it in context.

This is the listener we just created.

Adding the listener to the list view is crucial, as it's this step that notifies the listener when the user clicks on items in the list view. If you don't do this, the items in your list view won't be able to respond to clicks.

You've now seen everything you need in order to get the `TopLevelActivity` list view to respond to clicks.

The full TopLevelActivity.java code

Here's the complete code for *TopLevelActivity.java*. Replace the code the wizard created for you with the code below and on the next page, then save your changes:

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.widget.AdapterView;
import android.widget.ListView;
import android.view.View;

public class TopLevelActivity extends Activity {
```

We're using all these classes, so we need to import them.

Make sure your activity extends the Activity class.

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

TopLevelActivity.java (continued)

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_top_level);
        //Create an OnItemClickListener
        AdapterView.OnItemClickListener itemClickListener =
                                    new AdapterView.OnItemClickListener(){
            public void onItemClick(AdapterView<?> listView,
                                    View itemView,
                                    int position,
                                    long id) {
                if (position == 0) {
                    Intent intent = new Intent(TopLevelActivity.this,
                                               DrinkCategoryActivity.class);
                    startActivity(intent);
                }
            }
        };
        //Add the listener to the list view
        ListView listView = (ListView) findViewById(R.id.list_options);
        listView.setOnItemClickListener(itemClickListener);
    }
}
```

Create the listener.

Implement its onItemClick() method.

Launch DrinkCategoryActivity if the user clicks on the Drinks item. We'll create this activity next, so don't worry if Android Studio says it doesn't exist.

Add the listener to the list view.

What the TopLevelActivity.java code does

1. **The onCreate() method in TopLevelActivity creates an onItemClickListener and links it to the activity's ListView.**

2. **When the user clicks on an item in the list view, the onItemClickListener's onItemClick() method gets called.**
If the Drinks item is clicked, the `onItemClickListener` creates an intent to start `DrinkCategoryActivity`.

Where we've got to

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

So far we've added *Drink.java* to our app and created `TopLevelActivity` and its layout.

The next thing we need to do is create `DrinkCategoryActivity` and its layout so that it gets launched when the user clicks on the Drinks option in `TopLevelActivity`.

there are no Dumb Questions

Q: Why did we have to create an event listener to get items in the list view to respond to clicks? Couldn't we have just used its `android:onClick` attribute in the layout code?

A: You can only use the `android:onClick` attribute in activity layouts for buttons, or any views that are subclasses of `Button` such as `CheckBox`es and `RadioButton`s.

The `ListView` class isn't a subclass of `Button`, so using the `android:onClick` attribute won't work. That's why you have to implement your own listener.

Exercise

Here's some activity code from a separate project. When the user clicks on an item in a list view, the code is meant to display the text of that item in a text view (the text view has an ID of `text_view` and the list view has an ID of `list_view`). Does the code do what it's meant to? If not, why not?

```
package com.hfad.ch06ex;

import android.app.Activity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.TextView;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView textView = (TextView) findViewById(R.id.text_view);
        AdapterView.OnItemClickListener itemClickListener =
                new AdapterView.OnItemClickListener(){
                    public void onItemClick(AdapterView<?> listView,
                                            View v,
                                            int position,
                                            long id) {
                        TextView item = (TextView) v;
                        textView.setText(item.getText());
                    }
                };
        ListView listView = (ListView) findViewById(R.id.list_view);
    }
}
```

Here's some activity code from a separate project. When the user clicks on an item in a list view, the code is meant to display the text of that item in a text view (the text view has an ID of `text_view` and the list view has an ID of `list_view`). Does the code do what it's meant to? If not, why not?

```
package com.hfad.ch06ex;

import android.app.Activity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.TextView;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView textView = (TextView) findViewById(R.id.text_view);
        AdapterView.OnItemClickListener itemClickListener =
                new AdapterView.OnItemClickListener(){
                    public void onItemClick(AdapterView<?> listView,
                                            View v,
                                            int position,
                                            long id) {
                        TextView item = (TextView) v;
                        textView.setText(item.getText());
                    }
                };
        ListView listView = (ListView) findViewById(R.id.list_view);
    }
}
```

This is the item in the ListView that was clicked. It's a TextView, so we can get its text using getText().

The code doesn't work as intended because the line of code

listView.setOnItemClickListener(itemClickListener);

is missing from the end of the code. Apart from that, the code's fine.

- Add resources
- TopLevelActivity
- **DrinkCategoryActivity**
- DrinkActivity

A category activity displays the data for a single category

As we said earlier, `DrinkCategoryActivity` is an example of a category activity. A category activity is one that shows the data that belongs to a particular category, often in a list. You use the category activity to navigate to details of the data.

We're going to use `DrinkCategoryActivity` to display a list of drinks. When the user clicks on one of the drinks, we'll show them the details of that drink.

Create DrinkCategoryActivity

To work on the next step in our checklist, we'll create an activity with a single list view that displays a list of all the drinks. Select the *com.hfad.starbuzz* package in the *app/src/main/java* folder, then go to File→New...→Activity→Empty Activity. Name the activity "DrinkCategoryActivity", name the layout "activity_drink_category", make sure the package name is `com.hfad.starbuzz` and **uncheck the Backwards Compatibility (AppCompat) checkbox.**

We'll update the layout code on the next page.

Some versions of Android Studio may ask you what the source language of your activity should be. If prompted, select the option for Java.

Update activity_drink_category.xml

Here's the code for *activity_drink_category.xml*. As you can see, it's a simple linear layout with a list view. Update your version of *activity_drink_category.xml* to reflect ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.starbuzz.DrinkCategoryActivity">

    <ListView
        android:id="@+id/list_drinks"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

This layout only needs to contain a ListView.

There's one key difference between the list view we're creating here, and the one we created in *activity_top_activity.xml*: there's no `android:entries` attribute here. But why?

In *activity_top_activity.xml*, we used the `android:entries` attribute to bind data to the list view. This worked because the data was held as a static String array resource. The array was described in *strings.xml*, so we could easily refer to it using:

```
android:entries="@array/options"
```

where `options` is the name of the String array.

Using `android:entries` works fine if the data is a static array in *strings.xml*. But what if it isn't? What if the data is held in an array you've programmatically created in Java code, or held in a database? In that case, the `android:entries` attribute won't work.

If you need to bind your list view to data held in something other than a String array resource, you need to take a different approach; you need to write activity code to bind the data. In our case, we need to bind our list view to the `drinks` array in the `Drink` class.

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

For nonstatic data, use an adapter

If you need to display data in a list view that comes from a source other than *strings.xml* (such as a Java array or database), you need to use an **adapter**. An adapter acts as a bridge between the data source and the list view:

An adapter acts as a bridge between a view and a data source. An ArrayAdapter is a type of adapter that specializes in working with arrays.

There are several different types of adapter. For now, we're going to focus on **array adapters**.

An array adapter is a type of adapter that's used to bind arrays to views. You can use it with any subclass of the `AdapterView` class, which means you can use it with both list views and spinners.

In our case, we're going to use an array adapter to display data from the `Drink.drinks` array in the list view.

We'll see how this works on the next page.

Connect list views to arrays with an array adapter

- [x] Add resources
- [x] TopLevelActivity
- [ ] **DrinkCategoryActivity**
- [ ] DrinkActivity

You use an array adapter by initializing it and then attaching it to the list view.

To initialize the array adapter, you first specify what type of data is contained in the array you want to bind to the list view. You then pass the array adapter three parameters: a `Context` (usually the current activity), a layout resource that specifies how to display each item in the array, and the array itself.

Here's the code to create an array adapter that displays drink data from the `Drink.drinks` array (you'll add this code to *DrinkCategoryActivity.java* on the next page):

```
ArrayAdapter<Drink> listAdapter = new ArrayAdapter<>(
        this,
        android.R.layout.simple_list_item_1,
        Drink.drinks);
```

"this" refers to the current activity. The Activity class is a subclass of Context.

This is a built-in layout resource. It tells the array adapter to display each item in the array in a single text view.

The array

You then attach the array adapter to the list view using the `ListView setAdapter()` method:

```
ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
listDrinks.setAdapter(listAdapter);
```

Behind the scenes, the array adapter takes each item in the array, converts it to a `String` using its `toString()` method, and puts each result into a text view. It then displays each text view as a single row in the list view.

These are the drinks from the drinks array. Each row in the list view is a single text view, each one displaying a separate drink.

Add the array adapter to DrinkCategoryActivity

We'll change the *DrinkCategoryActivity.java* code so that the list view uses an array adapter to get drinks data from the `Drink` class. We'll put the code in the `onCreate()` method so that the list view gets populated when the activity gets created.

Here's the full code for the activity (update your copy of *DrinkCategoryActivity.java* to reflect ours, then save your changes):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class DrinkCategoryActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink_category);
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<>(
                this,
                android.R.layout.simple_list_item_1,
                Drink.drinks);
        ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
        listDrinks.setAdapter(listAdapter);
    }
}
```

We're using these classes so we need to import them.

Make sure your activity extends the Activity class.

This populates the list view with data from the drinks array.

These are all the changes needed to get your list view to display a list of the drinks from the `Drink` class. Let's go through what happens when the code runs.

What happens when you run the code

1. **When the user clicks on the Drinks option, DrinkCategoryActivity is launched.**
   Its layout has a `LinearLayout` that contains a `ListView`.

2. **DrinkCategoryActivity creates an ArrayAdapter<Drink>, an array adapter that deals with arrays of Drink objects.**

3. **The array adapter retrieves data from the drinks array in the Drink class.**
   It uses the `Drink.toString()` method to return the name of each drink.

4. **DrinkCategoryActivity makes the ListView use the array adapter via the setAdapter() method.**
   The list view uses the array adapter to display a list of the drink names.

- [x] Add resources
- [x] TopLevelActivity
- [ ] **DrinkCategoryActivity**
- [ ] DrinkActivity

Test drive the app

When you run the app, `TopLevelActivity` gets displayed as before. When you click on the Drinks item, `DrinkCategoryActivity` is launched. It displays the names of all the drinks from the `Drink` Java class.

Click on the Drinks item to see a list of drinks.

On the next page we'll review what we've done in the app so far, and what's left for us to do.

App review: where we are

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

So far we've added *Drink.java* to our app, and created activities `TopLevelActivity` and `DrinkCategoryActivity` along with their layouts.

Here's what our app currently does:

1. **When the app gets launched, it starts activity TopLevelActivity.**
   The activity displays a list of options for Drinks, Food, and Stores.

2. **The user clicks on Drinks in TopLevelActivity.**
   This launches activity `DrinkCategoryActivity`, which displays a list of drinks.

3. **DrinkCategoryActivity gets the values for its list of drinks from the *Drink.java* class file.**

The next thing we'll do is get `DrinkCategoryActivity` to launch `DrinkActivity`, passing it details of whichever drink was clicked.

Pool Puzzle

Your **goal** is to create an activity that binds a Java array of colors to a spinner. Take code snippets from the pool and place them into the blank lines in the activity. You may **not** use the same snippet more than once, and you won't need to use all the snippets.

Remember, we covered spinners in Chapter 5.

We're not using this activity in our app.

```
...
public class MainActivity extends Activity {

    String[] colors = new String[] {"Red", "Orange", "Yellow", "Green", "Blue"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner spinner = (............) findViewById(R.id.spinner);
        ArrayAdapter<............> adapter = new ArrayAdapter<>(
                .........,
                android.R.layout.simple_spinner_item,
                colors);
        spinner.................(adapter);
    }
}
```

This displays each value in the array as a single row in the spinner.

Note: each snippet from the pool can only be used once!

Answers on page 287.

How we handled clicks in TopLevelActivity

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

Earlier in this chapter, we needed to get `TopLevelActivity` to react to the user clicking on the first item in the list view, the Drinks option, by starting `DrinkCategoryActivity`. To do that, we had to create an `OnItemClickListener`, implement its `onItemClick()` method, and assign it to the list view. Here's a reminder of the code:

```
AdapterView.OnItemClickListener itemClickListener =
                                        new AdapterView.OnItemClickListener(){
    public void onItemClick(AdapterView<?> listView,
                            View itemView,
                            int position,
                            long id) {
        if (position == 0) {
            Intent intent = new Intent(TopLevelActivity.this,
                                       DrinkCategoryActivity.class);
            startActivity(intent);
        }
    }
};
ListView listView = (ListView) findViewById(R.id.list_options);
listView.setOnItemClickListener(itemClickListener);
```

Create the listener.

The list view

The item view that was clicked, its position in the list, and the row ID of the underlying data.

Add the listener to the list view.

We had to set up an event listener in this way because list views aren't hardwired to respond to clicks in the way that buttons are.

So how should we get `DrinkCategoryActivity` to handle user clicks?

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

Pass the ID of the item that was clicked by adding it to an intent

When you use a category activity to display items in a list view, you'll usually use the `onItemClick()` method to start another activity that displays details of the item the user clicked. To do this, you create an intent that starts the second activity. You then add the ID of the item that was clicked as extra information so that the second activity can use it when the activity starts.

In our case, we want to start `DrinkActivity` and pass it the ID of the drink that was selected. `DrinkActivity` will then be able to use this information to display details of the right drink. Here's the code for the intent:

DrinkCategoryActivity needs to start DrinkActivity.

```
Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int) id);
startActivity(intent);
```

Add the ID of the item that was clicked to the intent. This is the index of the drink in the drinks array.

We're using a constant for the name of the extra information in the intent so that we know DrinkCategoryActivity and DrinkActivity are using the same String. We'll add this constant to DrinkActivity when we create the activity.

It's common practice to pass the ID of the item that was clicked because it's the ID of the underlying data. If the underlying data is an array, the ID is the index of the item in the array. If the underlying data comes from a database, the ID is the ID of the record in the table. Passing the ID of the item in this way means that it's easier for the second activity to get details of the data, and then display it.

That's everything we need to make `DrinkCategoryActivity` start `DrinkActivity` and tell it which drink was selected. The full activity code is on the next page.

The full DrinkCategoryActivity code

Here's the full code for *DrinkCategoryActivity.java* (add the new method to your code, then save your changes):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.view.View;
import android.content.Intent;
import android.widget.AdapterView;

public class DrinkCategoryActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink_category);
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<>(
                this,
                android.R.layout.simple_list_item_1,
                Drink.drinks);
        ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
        listDrinks.setAdapter(listAdapter);

        //Create the listener
        AdapterView.OnItemClickListener itemClickListener =
                new AdapterView.OnItemClickListener(){
                    public void onItemClick(AdapterView<?> listDrinks,
                                            View itemView,
                                            int position,
                                            long id) {
                        //Pass the drink the user clicks on to DrinkActivity
                        Intent intent = new Intent(DrinkCategoryActivity.this,
                                                   DrinkActivity.class);
                        intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int) id);
                        startActivity(intent);
                    }
                };

        //Assign the listener to the list view
        listDrinks.setOnItemClickListener(itemClickListener);
    }
}
```

We're using these extra classes so we need to import them.

Create a listener to listen for clicks.

This gets called when an item in the list view is clicked.

When the user clicks on a drink, pass its ID to DrinkActivity and start it.

We'll add DrinkActivity next, so don't worry if Android Studio says it doesn't exist.

- Add resources
- TopLevelActivity
- DrinkCategoryActivity
- → DrinkActivity

A detail activity displays data for a single record

As we said earlier, `DrinkActivity` is an example of a detail activity. A detail activity displays details for a particular record, and you generally navigate to it from a category activity.

We're going to use `DrinkActivity` to display details of the drink the user selects. The `Drink` class includes the drink's name, description, and image resource ID, so we'll display this data in our layout. We'll include an image view for the drink image resource, and text views for the drink name and description.

To create the activity, select the *com.hfad.starbuzz* package in the *app/src/main/java* folder, then go to File→New...→Activity→Empty Activity. Name the activity "DrinkActivity", name the layout "activity_drink", make sure the package name is `com.hfad.starbuzz`, and **uncheck the Backwards Compatibility (AppCompat) checkbox.** Then replace the contents of *activity_drink.xml* with this:

Make sure you create the new activity.

If prompted for the activity's source language, select the option for Java.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.starbuzz.DrinkActivity" >

    <ImageView
        android:id="@+id/photo"
        android:layout_width="190dp"
        android:layout_height="190dp" />

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/description"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Now that you've created the layout of your detail activity, we can populate its views.

Add resources
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

Retrieve data from the intent

As you've seen, when you want a category activity to start a detail activity, you have to make items in the category activity list view respond to clicks. When an item is clicked, you create an intent to start the detail activity. You pass the ID of the item the user clicked as extra information in the intent.

When the detail activity is started, it can retrieve the extra information from the intent and use it to populate its views. In our case, we can use the information in the intent that started `DrinkActivity` to retrieve details of the drink the user clicked.

When we created `DrinkCategoryActivity`, we added the ID of the drink the user clicked as extra information in the intent. We gave it the label `DrinkActivity.EXTRA_DRINKID`, which we need to define as a constant in *DrinkActivity.java*:

```
public static final String EXTRA_DRINKID = "drinkId";
```

As you saw in Chapter 3, you can retrieve the intent that started an activity using the `getIntent()` method. If this intent has extra information, you can use the intent's `get*()` methods to retrieve it. Here's the code to retrieve the value of `EXTRA_DRINKID` from the intent that started `DrinkActivity`:

```
int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
```

Once you've retrieved the information from the intent, you can use it to get the data you need to display in your detail record.

In our case, we can use `drinkId` to get details of the drink the user selected. `drinkId` is the ID of the drink, the index of the drink in the `drinks` array. This means that you can get details about the drink the user clicked on using:

```
Drink drink = Drink.drinks[drinkId];
```

This gives us a `Drink` object containing all the information we need to update the views attributes in the activity:

drink

name="Latte"
description="A couple of espresso shots with steamed milk"
imageResourceId=R.drawable.latte

Update the views with the data

When you update the views in your detail activity, you need to make sure that the values they display reflect the data you've derived from the intent.

Our detail activity contains two text views and an image view. We need to make sure that each of these is updated to reflect the details of the drink.

Drink Magnets

See if you can use the magnets below to populate the `DrinkActivity` views with the correct data.

```
...
//Get the drink from the intent
int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
Drink drink = Drink.drinks[drinkId];


//Populate the drink name
TextView name = (TextView)findViewById(R.id.name);

name.                    (drink.getName());


//Populate the drink description
TextView description = (TextView)findViewById(R.id.description);

description.                    (drink.getDescription());


//Populate the drink image
ImageView photo = (ImageView)findViewById(R.id.photo);

photo.                                (drink.getImageResourceId());

photo.                                          (drink.getName());
...
```

setText

setContent

setContentDescription

setImageResourceId

setImageResource

setText

Drink Magnets Solution

See if you can use the magnets below to populate the `DrinkActivity` views with the correct data.

```
...
//Get the drink from the intent
int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
Drink drink = Drink.drinks[drinkId];


//Populate the drink name
TextView name = (TextView)findViewById(R.id.name);

name.setText(drink.getName());


//Populate the drink description
TextView description = (TextView)findViewById(R.id.description);

description.setText(drink.getDescription());


//Populate the drink image
ImageView photo = (ImageView)findViewById(R.id.photo);

photo.setImageResource(drink.getImageResourceId());

photo.setContentDescription(drink.getName());
...
```

Use setText() to set the text in a text view.

You set the source of the image using setImageResource().

This is needed to make the app more accessible.

You didn't need to use these.

`setContent`

`setImageResourceId`

- ☑ Add resources
- ☑ TopLevelActivity
- ☑ DrinkCategoryActivity
- → ☐ **DrinkActivity**

The DrinkActivity code

Here's the full code for *DrinkActivity.java* (replace the code the wizard gave you with the code below, then save your changes):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;

public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKID = "drinkId";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

        //Get the drink from the intent
        int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
        Drink drink = Drink.drinks[drinkId];

        //Populate the drink name
        TextView name = (TextView)findViewById(R.id.name);
        name.setText(drink.getName());

        //Populate the drink description
        TextView description = (TextView)findViewById(R.id.description);
        description.setText(drink.getDescription());

        //Populate the drink image
        ImageView photo = (ImageView)findViewById(R.id.photo);
        photo.setImageResource(drink.getImageResourceId());
        photo.setContentDescription(drink.getName());
    }
}
```

We're using these classes so we need to import them.

Make sure your activity extends the Activity class.

Add EXTRA_DRINKID as a constant.

Use the drinkId to get details of the drink the user chose.

Populate the views with the drink data.

What happens when you run the app

- ☑ Add resources
- ☑ TopLevelActivity
- ☑ DrinkCategoryActivity
- → ☐ **DrinkActivity**

1. **When the user starts the app, it launches TopLevelActivity.**

2. **The onCreate() method in TopLevelActivity creates an onItemClickListener and links it to the activity's ListView.**

3. **When the user clicks on an item in the ListView, the onItemClickListener's onItemClick() method gets called.**
If the Drinks item was clicked, the `onItemClickListener` creates an intent to start `DrinkCategoryActivity`.

4. **DrinkCategoryActivity displays a single ListView.**
The `DrinkCategoryActivity` list view uses an `ArrayAdapter<Drink>` to display a list of drink names.

The story continues

5. **When the user chooses a drink from DrinkCategoryActivity's ListView, onItemClickListener's onItemClick() method gets called.**

6. **The onItemClick() method creates an intent to start DrinkActivity, passing along the drink ID as extra information.**

7. **DrinkActivity launches.**
It retrieves the drink ID from the intent, and gets details for the correct drink from the `Drink` class. It uses this information to update its views.

Test drive the app

When you run the app, `TopLevelActivity` gets displayed.

We've implemented the Drinks part of the app. The other items won't do anything if you click on them.

When you click on the Drinks item, `DrinkCategoryActivity` is launched. It displays all the drinks from the `Drink` java class.

When you click on one of the drinks, `DrinkActivity` is launched and details of the selected drink are displayed.

Using these three activities, you can see how to structure your app into top-level activities, category activities, and detail/edit activities. In Chapter 15, we'll revisit the Starbuzz app to explain how you can retrieve the drinks from a database.

Pool Puzzle Solution

Your **goal** is to create an activity that binds a Java array of colors to a spinner. Take code snippets from the pool and place them into the blank lines in the activity. You may **not** use the same snippet more than once, and you won't need to use all the snippets.

```
...
public class MainActivity extends Activity {

    String[] colors = new String[] {"Red", "Orange", "Yellow", "Green", "Blue"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner spinner = ( Spinner ) findViewById(R.id.spinner);
        ArrayAdapter< String > adapter = new ArrayAdapter<>(
                this ,
                android.R.layout.simple_spinner_item,
                colors);
        spinner. setAdapter (adapter);
    }
}
```

We're using an array of type String.

Use setAdapter() to get the spinner to use the array adapter.

You didn't need to use these code snippets.

Your Android Toolbox

You've got Chapter 7 under your belt and now you've added list views and app design to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Sort your ideas for activities into top-level activities, category activities, and detail/edit activities. Use the category activities to navigate from the top-level activities to the detail/edit activities.
- A list view displays items in a list. Add it to your layout using the `<ListView>` element.
- Use `android:entries` in your layout to populate the items in your list views from an array defined in *strings.xml*.
- An adapter acts as a bridge between an `AdapterView` and a data source. `ListViews` and `Spinners` are both types of `AdapterView`.
- An `ArrayAdapter` is an adapter that works with arrays.
- Handle click events on `Buttons` using `android:onClick` in the layout code. Handle click events elsewhere by creating a listener and implementing its click event.

8 support libraries and app bars

Taking Shortcuts

Everybody likes a shortcut.

And in this chapter you'll see how to add shortcuts to your apps using **app bars**. We'll show you how to start activities by *adding actions* to your app bar, how to share content with other apps using the *share action provider*, and how to navigate up your app's hierarchy by implementing *the app bar's Up button*. Along the way we'll introduce you to the powerful **Android Support Libraries**, which are key to making your apps look fresh on older versions of Android.

Great apps have a clear structure

In the previous chapter, we looked at ways of structuring an app to create the best user experience. Remember that one way of creating an app is to organize the screens into three types:

Top-level screens

This is usually the first activity in your app that your user sees.

Category screens

Category screens show the data that belongs to a particular category, often in a list. They allow the user to navigate to detail/edit screens.

Detail/edit screens

These display details for a particular record, let the user edit the record, or allow the user to enter new records.

This is a rough sketch of a Pizza app. It contains details of pizzas, pasta dishes, and stores. It also allows the user to order a meal.

They also have great shortcuts

If a user's going to use your app a lot, they'll want quick ways to get around. We're going to look at navigational views that will give your user shortcuts around your app, providing more space in your app for actual content. Let's begin by taking a closer look at the top-level screen in the above Pizza app.

Different types of navigation

In the top-level screen of the Pizza app, there's a list of options for places in the app the user can go to.

The top three options link to category activities; the first presents the user with a list of pizzas, the second a list of pasta, and the third a list of stores. They allow the user to navigate around the app.

These are like the navigation options we looked at in Chapter 7.

The fourth option links to a detail/edit activity that allows the user to create an order. This option enables the user to perform an **action**.

In Android apps, you can add actions to the **app bar**. The app bar is the bar you often see at the top of activities; it's sometimes known as the **action bar**. You generally put your app's most important actions in the app bar so that they're prominent at the top of the screen.

In the Pizza app, we can make it easy for the user to place an order wherever they are in the app by making sure there's an app bar at the top of every activity that includes a Create Order button. This way the user will have access to it wherever they are.

Let's look at how you create app bars.

Here's what we're going to do

There are a few things we're going to cover in this chapter.

1. **Add a basic app bar.**
   We'll create an activity called `MainActivity` and add a basic app bar to it by applying a theme.

2. **Replace the basic app bar with a toolbar.**
   To use the latest app bar features, you need to replace the basic app bar with a toolbar. This looks the same as the basic app bar, but you can use it to do more things.

3. **Add a Create Order action.**
   We'll create a new activity called `OrderActivity`, and add an action to `MainActivity`'s app bar that opens it.

4. **Implement the Up button.**
   We'll implement the Up button on `OrderActivity`'s app bar so that users have an easy way of navigating back to `MainActivity`.

5. **Add a share action provider.**
   We'll add a share action provider to `MainActivity`'s app bar so that users can share text with other apps and invite their friends to join them for pizza.

   You'll find out what action providers are later in the chapter.

Let's start by looking at how you add a basic app bar.

- Basic app bar
- Toolbar
- Action
- Up button
- Share action

Add an app bar by applying a theme

An app bar has a number of uses:

- Displaying the app or activity name so that the user knows where in the app they are. As an example, an email app might use the app bar to indicate whether the user is in their inbox or junk folder.
- Making key actions prominent in a way that's predictable—for example, sharing content or performing searches.
- Navigating to other activities to perform an action.

To add a basic app bar, you need to use a **theme** that includes an app bar. A theme is a style that's applied to an activity or application so that your app has a consistent look and feel. It controls such things as the color of the activity background and app bar, and the style of the text.

Android comes with a number of built-in themes that you can use in your apps. Some of these, such as the Holo themes, were introduced in early releases of Android, and others, such as the Material themes, were introduced much later to give apps a more modern appearance.

The Holo themes have been in Android since API level 11.

The Material themes were introduced in API level 21.

These themes look a bit different from the one on the previous page, as they haven't had any extra styling applied to them. You'll find out how to add styling later in the chapter.

But there's a problem. You want your apps to look as modern and up-to-date as possible, but you can only use themes from the version of Android they were released in. As an example, you can't use the native Material themes on devices that are running a version of Android older than Lollipop, as the Material themes were introduced with API level 21.

The problem isn't just limited to themes. Every new release of Android introduces new features that people want to see in their apps, such as new GUI components. But not everyone upgrades to the latest version of Android as soon as it comes out. In fact, most people are at least one version of Android behind.

So how can you use the latest Android features and themes in your apps if most people aren't using the latest version? How can you give your users a consistent user experience irrespective of what version of Android they're using without making your app look old-fashioned?

Support libraries allow you to use new features in older versions of Android

The Android team solved this problem by coming up with the idea of **Support Libraries**.

The Android Support Libraries provide backward compatibility with older versions of Android. They sit outside the main release of Android, and contain new Android features that developers can use in the apps they're building. The Support Libraries mean that you can give users on older devices the same experience as users on newer devices *even if they're using different versions of Android*.

Here are some of the Support Libraries that are available for you to use:

v4 Support Library
Includes the largest set of features, such as support for application components and user interface features.

v7 AppCompat Library
Includes support for app bars.

v7 Cardview Library
Adds support for the `CardView` widget, allowing you to show information inside cards.

These are just some of the Support Libraries.

Constraint Layout Library
Allows you to create constraint layouts. You used features from this library in Chapter 6.

v7 RecyclerView Library
Adds support for the `RecyclerView` widget.

Design Support Library
Adds support for extra components such as tabs and navigation drawers.

Each library includes a specific set of features.

The v7 AppCompat Library contains a set of up-to-date themes that can be used with older versions of Android: in practice, they can be used with nearly all devices, as most people are using API level 19 or above. We're going to use the v7 AppCompat Library by applying one of the themes it contains to our app. This will add an app bar that will look up-to-date and work the same on all versions of Android that we're targeting. Whenever you want to use one of the Support Libraries, you first need to add it to your app. We'll look at how you do this after we've created the project.

- **Basic app bar**
- Toolbar
- Action
- Up button
- Share action

Create the Pizza app

We'll start by creating a prototype of the Pizza app. Create a new Android project for an application named "Bits and Pizzas" with a company domain of "hfad.com", making the package name `com.hfad.bitsandpizzas`. The minimum SDK should be API level 19 so that it works with most devices. You'll need an empty activity called "MainActivity" and a layout called "activity_main". Make sure you ***check* the Backwards Compatibility (AppCompat) checkbox** (you'll see why a few pages ahead).

Next, we'll look at how you add a Support Library to the project.

Add the v7 AppCompat Support Library

We're going to use one of the themes from the v7 AppCompat Library, so we need to add the library to our project as a dependency. Doing so means that the library gets included in your app, and downloaded to the user's device.

To manage the Support Library files that are included in your project, choose File→Project Structure. Then click on the app module and choose Dependencies. You'll be presented with the following screen:

Android Studio may have already added the AppCompat Support Library for you automatically. If so, you will see it listed as appcompat-v7, as shown above.

If the AppCompat Library hasn't been added for you, you will need to add it yourself. Click on the "+" button at the bottom or right side of the Project Structure screen. Choose the Library Dependency option, select the appcompat-v7 library, then click on the OK button. Click on OK again to save your changes and close the Project Structure window.

Once the AppCompat Support Library has been added to your project, you can use its resources in your app. In our case, we want to apply one of its themes in order to give `MainActivity` an app bar. Before we do that, however, we need to look at the type of activity we're using for `MainActivity`.

AppCompatActivity lets you use AppCompat themes

So far, all of the activities we've created have extended the `Activity` class. This is the base class for all activities, and it's what makes your activity an activity. If you want to use the AppCompat themes, however, you need to use a special kind of activity, called an **`AppCompatActivity`**, instead.

The `AppCompatActivity` class is a subclass of `Activity`. It lives in the AppCompat Support Library, and it's designed to work with the AppCompat themes. **Your activity needs to extend the `AppCompatActivity` class instead of the `Activity` class whenever you want an app bar that provides backward compatibility with older versions of Android**.

As `AppCompatActivity` is a subclass of the `Activity` class, everything you've learned about activities so far still applies. `AppCompatActivity` works with layouts in just the same way, and inherits all the lifecycle methods from the `Activity` class. The main difference is that, compared to `Activity`, `AppCompatActivity` contains extra smarts that allow it to work with the themes from the AppCompat Support Library.

Here's a diagram showing the `AppCompatActivity` class hierarchy:

Activity class
(android.app.Activity)
The `Activity` class implements default versions of the lifecycle methods.

FragmentActivity class
(android.support.v4.app.FragmentActivity)
The base class for activities that need to use support fragments. You'll find out about fragments in the next chapter.

AppCompatActivity class
(android.support.v7.app.AppCompatActivity)
The base class for activities that use the Support Library app bar.

YourActivity class
(com.hfad.foo)

We'll make sure `MainActivity` extends `AppCompatActivity` on the next page.

MainActivity needs to be an AppCompatActivity

We want to use one of the AppCompat themes, so we need to make sure our activities extend the `AppCompatActivity` class instead of the `Activity` class. Happily, this should already be the case if you checked the Backwards Compatibility (AppCompat) checkbox when you first created the activity. Open the file *MainActivity.java*, then make sure your code matches ours below:

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

The AppCompatActivity class lives in the v7 AppCompat Support Library.

Make sure your activity extends AppCompatActivity.

Now that we've confirmed that our activity extends `AppCompatActivity`, we can add an app bar by applying a theme from the AppCompat Support Library. You apply a theme in the app's *AndroidManifest.xml* file, so we'll look at this file next.

there are no Dumb Questions

Q: What versions of Android can the Support Libraries be used with?

A: It depends on the version of the Support Library. Prior to version 24.2.0, Libraries prefixed with v4 could be used with API level 4 and above, and those prefixed with v7 could be used with API level 7 and above. When version 24.2.0 of the Support Libraries was released, the minimum API for all Support Libraries became API level 9. The minimum API level is likely to increase in the future.

Q: In earlier chapters, Android Studio gave me activities that already extended `AppCompatActivity`. Why's that?

A: When you create an activity in Android Studio, the wizard includes a checkbox asking if you want to create a Backwards Compatible (AppCompat) activity. If you left this checked in earlier chapters, Android Studio would have generated activities that extend `AppCompatActivity`.

Q: I've seen code that extends `ActionBarActivity`. What's that?

A: In older versions of the AppCompat Support Library, you used the `ActionBarActivity` class to add app bars. This was deprecated in version 22.1 in favor of `AppCompatActivity`.

AndroidManifest.xml can change your app bar's appearance

As you've seen earlier in the book, an app's *AndroidManifest.xml* file provides essential information about the app, such as what activities it contains. It also includes a number of attributes that have a direct impact on your app bars.

Here's the *AndroidManifest.xml* code Android Studio created for us (we've highlighted the key areas):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.bitsandpizzas">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            ...
        </activity>
    </application>
</manifest>
```

App icons. Android Studio provides icons by default.

The user-friendly name of the app

The theme

The **`android:icon`** attribute assigns an icon to the app. The icon is used as the launcher icon for the app, and if the theme you're using displays an icon in the app bar, it will use this icon. **`android:roundIcon`** may be used instead on devices running Android 7.1 or above.

The icon is a **mipmap** resource. A mipmap is an image that can be used for application icons, and they're held in *mipmap** folders in *app/src/main/res*. Just as with drawables, you can add different images for different screen densities by adding them to an appropriately named *mipmap* folder. As an example, an icon in the *mipmap-hdpi* folder will be used by devices with high-density screens. You refer to mipmap resources in your layout using `@mipmap`.

Android Studio automatically added icons to our mipmap* folders when we created the project.

The **`android:label`** attribute describes a user-friendly label that gets displayed in the app bar. In the code above, it's used in the `<application>` tag to apply a label to the entire app. You can also add it to the `<activity>` tag to assign a label to a single activity.

The **`android:theme`** attribute specifies the theme. Using this attribute in the `<application>` element applies the theme to the entire app. Using it in the `<activity>` element applies the theme to a single activity.

We'll look at how you apply the theme on the next page.

How to apply a theme

When you want to apply a theme to your app, you have two main options:

- Hardcode the theme in *AndroidManifest.xml*.
- Apply the theme using a style.

Let's look at these two approaches.

1. Hardcoding the theme

To hardcode the theme in *AndroidManifest.xml*, you update the `android:theme` attribute in the file to specify the name of the theme you want to use. As an example, to apply a theme with a light background and a dark app bar, you'd use:

```
<application
    ...
    android:theme="Theme.AppCompat.Light.DarkActionBar">
```

This is a simple way of applying a basic theme, but it means you can't, for example, change its colors.

This approach works well if you want to apply a basic theme without making any changes to it.

2. Using a style to apply the theme

Most of the time, you'll want to apply the theme using a style, as this approach enables you to tweak the theme's appearance. You may want to override the theme's main colors to reflect your app's brand, for example.

To apply a theme using a style, you update the `android:theme` attribute in *AndroidManifest.xml* to the name of a style resource (which you then need to create). In our case, we're going to use a style resource named `AppTheme`, so update the `android:theme` attribute in your version of *AndroidManifest.xml* to the following:

```
<application
    ...
    android:theme="@style/AppTheme">
```

Android Studio may have already added this to your version of AndroidManifest.xml.

The `@style` prefix tells Android that the theme the app's using is a style that's defined in a **style resource file**. We'll look at this next.

- Basic app bar
- Toolbar
- Action
- Up button
- Share action

Define styles in a style resource file

The style resource file holds details of any themes and styles you want to use in your app. When you create a project in Android Studio, the IDE will usually create a default style resource file for you called *styles.xml* located in the *app/src/main/res/values* folder.

If Android Studio hasn't created the file, you'll need to add it yourself. Switch to the Project view of Android Studio's explorer, highlight the *app/src/main/res/values* folder, go to the File menu, and choose New. Then choose the option to create a new Values resource file, and when prompted, name the file "styles". When you click on OK, Android Studio will create the file for you.

A basic style resource file looks like this:

This is the theme used in the app.

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">

    </style>
</resources>
```

There may be extra code here to customize the theme. We'll look at this a couple of pages ahead.

A style resource file can contain one or more styles. Each style is defined through the `<style>` element.

Each style must have a name, which you define with the `name` attribute; for example:

```
name="AppTheme"
```

In the code above, the style has a name of `"AppTheme"`, and *AndroidManifest.xml* can refer to it using `"@style/AppTheme"`.

The `parent` attribute specifies where the style should inherit its properties from; for example:

```
parent="Theme.AppCompat.Light.DarkActionBar"
```

This gives the app a theme of **`"Theme.AppCompat.Light.DarkActionBar"`**, which gives activities a light background, with a dark app bar. We'll look at some more of Android's available themes on the next page.

The app bar has a dark background with white text.

The main activity background is light.

Theme gallery

Basic app bar
Toolbar
Action
Up button
Share action

Android comes with a whole bunch of built-in themes that you can use in your apps. Here are just a few of them:

Theme.AppCompat.Light

Theme.AppCompat

Theme.AppCompat.Light.NoActionBar

Theme.AppCompat.NoActionBar

Theme.AppCompat.Light.DarkActionBar

There's also a DayNight theme, which uses one set of colors in the day, and another set at night.

The theme determines the basic appearance of the app, such as the color of the app bar and any views. But what if you want to modify the app's appearance?

Customize the look of your app

You can use customize the look of your app by overriding the properties of an existing theme in the style resource file. For example, you can change the color of the app bar, the status bar, and any UI controls. You override the theme by adding `<item>` elements to the `<style>` to describe each modification you want to make.

We're going to override three of the colors used by our theme. To do this, make sure that your version of *styles.xml* matches ours below:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

These three lines of code modify the theme by changing three of the colors.

The above code includes three modifications, each one described by a separate `<item>`. Each `<item>` has a `name` attribute that indicates what part of the theme you want to change, and a value that specifies what you want to change it to, like this:

```
<item name="colorPrimary">@color/colorPrimary</item>
```

This will change the colorPrimary part of the theme so it has a value of @color/colorPrimary.

`name="colorPrimary"` refers to the main color you want to use for your app. This color gets used for your app bar, and to "brand" your app with a particular color.

`name="colorPrimaryDark"` is a darker variant of your main color. It gets used as the color of the status bar.

`name="colorAccent"` refers to the color of any UI controls such as editable text views or checkboxes.

You set a new color for each of these areas by giving each `<item>` a value. The value can either be a hardcoded hexadecimal color value, or a reference to a color resource. We'll look at color resources on the next page.

There are a whole host of other theme properties you can change, but we're not going to cover them here. To find out more, visit https://developer.android.com/guide/topics/ui/themes.html.

Define colors in a color resource file

A color resource file is similar to a String resource file except that it contains colors instead of Strings. Using a color resource file makes it easy to make changes to the color scheme of your app, as all the colors you want to use are held in one place.

The color resource file is usually called *colors.xml*, and it's located in the *app/src/main/res/values* folder. When you create a project in Android Studio, the IDE will usually create this file for you.

If Android Studio hasn't created the file, you'll need to add it yourself. Switch to the Project view of Android Studio's explorer, highlight the *app/src/main/res/values* folder, go to the File menu, and choose New. Then choose the option to create a new Values resource file, and when prompted, name the file "colors". When you click on OK, Android Studio will create the file for you.

Next, open *colors.xml* and make sure that your version of the file matches ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Each of these is a color resource.

The code above defines three color resources. Each one has a name and a value. The value is a hexadecimal color value:

This says it's a color resource.

```
<color name="colorPrimary">#3F51B5</color>
```

The color resource has a name of "colorPrimary", and a value of #3F51B5 (blue).

The style resource file looks up colors from the color resource file using `@color/colorName`. For example:

```
<item name="colorPrimary">@color/colorPrimary</item>
```

overrides the primary color used in the theme with the value of `colorPrimary` in the color resource file.

Now that we've seen how to add an app bar by applying a theme, let's update `MainActivity`'s layout and take the app for a test drive.

- Basic app bar
- Toolbar
- Action
- Up button
- Share action

The code for activity_main.xml

For `MainActivity`'s layout, we're going to display some default text in a linear layout. Here's the code to do that; update your version of *activity_main.xml* to match ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context="com.hfad.bitsandpizzas.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

We're just displaying some basic placeholder text in MainActivity's layout because right now we want you to focus on app bars.

Test drive the app

When you run the app, `MainActivity` gets displayed. At the top of the activity there's an app bar.

This is the app bar. The default color's been overridden so that it's blue.

The status bar's default color has been overridden so it's a darker shade of blue than the app bar.

That's everything you need to apply a basic app bar in your activities. Why not experiment with changing the theme and colors? Then when you're ready, turn the page and we'll move on to the next step.

ActionBar vs. Toolbar

So far, you've seen how to add a basic app bar to the activities in your app by applying a theme that includes an app bar. Adding an app bar in this way is easy, but it has one disadvantage: *it doesn't necessarily include all the latest app bar features*.

Behind the scenes, any activity that acquires an app bar via a theme uses the `ActionBar` class for its app bar. The most recent app bar features, however, have been added to the `Toolbar` class in the AppCompat Support Library instead. This means that if you want to use the most recent app bar features in your app, you need to use the `Toolbar` class from the Support Library.

A toolbar looks just like the app bar you had previously, but it gives you more flexibility and includes the most recent app bar features.

Using the `Toolbar` class also gives you more flexibility. A toolbar is a type of view that you add to your layout just as you would any other type of view, and this makes it much easier to position and control than a basic app bar.

How to add a toolbar

We're going to change our activity so that it uses a toolbar from the Support Library for its app bar. Whenever you want to use the `Toolbar` class from the Support Library, there are a number of steps you need to perform:

1. **Add the v7 AppCompat Support Library as a dependency.**
   This is necessary because the `Toolbar` class lives in this library.

2. **Make sure your activity extends the AppCompatActivity class.**
   Your activity must extend `AppCompatActivity` (or one of its subclasses) in order to use the Support Library toolbar.

3. **Remove the existing app bar.**
   You do this by changing the theme to one that doesn't include an app bar.

4. **Add a toolbar to the layout.**
   The toolbar is a type of view, so you can position it where you want and control its appearance.

5. **Update the activity to set the toolbar as the activity's app bar.**
   This allows the activity to respond to the toolbar.

We'll go through these steps now.

1. Add the AppCompat Support Library

Before you can use the `Toolbar` class from the Support Library in your activities, you need to make sure that the v7 AppCompat Support Library has been added to your project as a dependency. In our particular case, the library has already been added to our project, as we needed it for the AppCompat themes.

To double-check that the Support Library is there, in Android Studio choose File→Project Structure, click on the app module, and choose Dependencies. You should see the v7 AppCompat Library listed as shown below:

2. Extend the AppCompatActivity class

When you want to use a theme from the AppCompat Library, you have to make sure that your activities extend the `AppCompatActivity` class. This is also the case if you want to use a toolbar from the Support Library as your app bar.

We've already completed this step because, earlier in this chapter, we changed *MainActivity.java* to use `AppCompatActivity`:

```
...
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    ...
}
```

Our MainActivity already extends AppCompatActivity.

The next thing we need to do is remove the existing app bar.

3. Remove the app bar

You remove the existing app bar in exactly the same way that you add one—by applying a **theme**.

When we wanted to add an app bar to our app, we applied a theme that displayed one. To do this, we used the `theme` attribute in *AndroidManifest.xml* to apply a style called `AppTheme`:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.bitsandpizzas">
    <application
        ...
        android:theme="@style/AppTheme">
        ...
    </application>
</manifest>
```

This looks up the theme from styles.xml.

The theme was then defined in *styles.xml* like this:

This is the theme we're using. It displays a dark app bar.

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        ...
    </style>
</resources>
```

The theme `Theme.AppCompat.Light.DarkActionBar` gives your activity a light background with a dark app bar. To remove the app bar, we're going to change the theme to `Theme.AppCompat.Light.NoActionBar` instead. Your activity will look the same as it did before except that no app bar will be displayed.

To change the theme, update *styles.xml* like this:

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkNoActionBar">
        ...
    </style>
</resources>
```

We customized the theme by overriding some of the colors. You can leave this code in place.

Change the theme from DarkActionBar to NoActionBar. This removes the app bar.

Now that we've removed the current app bar, we can add the toolbar.

Basic app bar
Toolbar
Action
Up button
Share action

4. Add a toolbar to the layout

As we said earlier, a toolbar is a view that you add to your layout. Toolbar code looks like this:

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />
```

This defines the toolbar.

Give the toolbar an ID so you can refer to it in your activity code.

Set the toolbar's size.

These control the app bar's appearance.

You start by defining the toolbar using:

```
<android.support.v7.widget.Toolbar
    ... />
```

This is the full path of the Toolbar class in the Support Library.

where `android.support.v7.widget.Toolbar` is the fully qualified path of the `Toolbar` class in the Support Library.

Once the toolbar has been defined, you then use other view attributes to give it an ID, and specify its appearance. As an example, to make the toolbar as wide as its parent and as tall as the default app bar size from the underlying theme, you'd use:

```
android:layout_width="match_parent"
android:layout_height="?attr/actionBarSize"
```

The toolbar is as wide as its parent, and as tall as the default app bar.

The `?attr` prefix means that you want to use an attribute from the current theme. In this particular case, `?attr/actionBarSize` is the height of an app bar that's specified in our theme.

You can also change your toolbar's appearance so that it has a similar appearance to the app bar that we had before. To do this, you can change the background color, and apply a **theme overlay** like this:

```
android:background="?attr/colorPrimary"
android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
```

Make the toolbar's background the same color as the app bar we had previously.

This gives the toolbar the same appearance as the app bar we had before. We have to use a theme overlay, as the NoActionBar theme doesn't style app bars in the same way as the DarkActionBar theme did.

A theme overlay is a special type of theme that alters the current theme by overwriting some of its attributes. We want our toolbar to look like our app bar did when we used a theme of `Theme.AppCompat.Light.DarkActionBar`, so we're using a theme overlay of `ThemeOverlay.AppCompat.Dark.ActionBar`.

On the next page we'll add the toolbar to the layout.

Add the toolbar to the layout...

If your app contains a single activity, you can add the toolbar to your layout just as you would any other view. Here is an example of the sort of code you would use in this situation (we're using a different approach, so don't update your layout with the code below):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

The code here doesn't include any padding. This is so that the toolbar fills the screen horizontally.

This code displays the toolbar at the top of the activity.

Later in the chapter we'll add a second activity to our app, so we're not using this approach. So you don't need to change your layout code to match this example.

We're using a linear layout, so the text view will be positioned below the toolbar.

This code displays the toolbar at the top of the activity. We've positioned the text view Android Studio gave us so that it's displayed underneath the toolbar. Remember that a toolbar is a view like any other view, so you need to take this into account when you're positioning your other views.

Adding the toolbar code to your layout works well if your app contains a single activity, as it means that all the code relating to your activity's appearance is in a single file. It works less well, however, if your app contains multiple activities. If you wanted to display a toolbar in multiple activities, you would need to define the toolbar in the layout of each activity. This means that if you wanted to change the style of the toolbar in some way, you'd need to edit *every single layout file*.

So what's the alternative?

Basic app bar
Toolbar
Action
Up button
Share action

...or define the toolbar as a separate layout

An alternative approach is to define the toolbar in a separate layout, and then include the toolbar layout in each activity. This means that you only need to define the toolbar once, and if you want to change the style of your toolbar, you only need to edit one file.

MainActivity has a layout, activity_main.

activity_main doesn't explicitly contain the toolbar code. Instead, it contains a reference to the toolbar's layout.

The toolbar's layout is contained in a separate file. If your activity contains multiple activities, each one can reference the toolbar's layout.

We're going to use this approach in our app. Start by creating a new layout file. Switch to the Project view of Android Studio's explorer, highlight the *app/src/res/main/layout* folder in Android Studio, then go to the File menu and choose New → Layout resource file. When prompted, give the layout file a name of "toolbar_main" and then click on OK. This creates a new layout file called *toolbar_main.xml*.

Next, open *toolbar_main.xml*, and replace any code Android Studio has created for you with the following:

```
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />
```

This toolbar code goes in a separate layout file so multiple activities can reference it.

This code is almost identical to the toolbar code you've already seen. The main difference is that we've left out the toolbar's `id` attribute, as we'll define this in the activity's main layout file *activity_main.xml* instead.

On the next page we'll look at how you include the toolbar layout in *activity_main.xml*.

Include the toolbar in the activity's layout

You can display one layout inside another using the `<include>` tag. This tag must contain a `layout` attribute that specifies the name of the layout you want to include. As an example, here's how you would use the `<include>` tag to include the layout *toolbar_main.xml*:

```
<include
    layout="@layout/toolbar_main" />
```

The @layout tells Android to look for a layout called toolbar_main.

We want to include the `toolbar_main` layout in *activity_main.xml*. Here's our code; update your version of *activity_main.xml* to match ours:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context="com.hfad.bitsandpizzas.MainActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/toolbar" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

Remove the padding so that the toolbar fills the screen horizontally.

Include the toolbar_main layout.

We're giving the toolbar an ID so we can refer to it in our activity code.

Now that we've added the toolbar to the layout, there's one more change we need to make.

5. Set the toolbar as the activity's app bar

The final thing we need to do is tell `MainActivity` to use the toolbar as its app bar.

So far we've only added the toolbar to the layout. While this means that the toolbar gets displayed at the top of the screen, the toolbar doesn't yet have any app bar functionality. As an example, if you were to run the app at this point, you'd find that the title of the app isn't displayed in the toolbar as it was in the app bar we had previously.

If you don't update your activity code after adding a toolbar to your layout, your toolbar will just appear as a plain strip with nothing in it.

To get the toolbar to behave like an app bar, we need to call the `AppCompatActivity`'s `setSupportActionBar()` method in the activity's `onCreate()` method, which takes one parameter: the toolbar you want to set as the activity's app bar.

Here's the code for *MainActivity.java*; update your code to match ours:

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

We're using the Toolbar class, so we need to import it.

Get a reference to the toolbar, and set it as the activity's app bar.

We need to use setSupportActionBar(), as we're using the toolbar from the Support Library.

That's all the code that you need to replace the activity's basic app bar with a toolbar. Let's see how it looks.

Test drive the app

When you run the app, a new toolbar is displayed in place of the basic app bar we had before. It looks similar to the app bar, but as it's based on the Support Library `Toolbar` class, it includes all the latest Android app bar functionality.

Here's our new toolbar. It looks like the app bar we had before, but it gives you more flexibility.

You've seen how to add an app bar, and how to replace the basic app bar with a toolbar. Over the next few pages we'll look at how to add extra functionaility to the app bar.

there are no Dumb Questions

Q: You've mentioned app bars, action bars, and toolbars. Is there a difference?

A: An app bar is the bar that usually appears at the top of your activities. It's sometimes called an action bar because in earlier versions of Android, the only way of implementing an app bar was via the `ActionBar` class.

The `ActionBar` class is used behind the scenes when you add an app bar by applying a theme. If your app doesn't rely on any new app bar features, this may be sufficient for your app.

An alternative way of adding an app bar is to implement a toolbar using the `Toolbar` class. The result looks similar to the default theme-based app bar, but it includes newer features of Android.

Q: I've added a toolbar to my activity, but when I run the app, it just looks like a band across the top of the screen. It doesn't even include the app name. Why's that?

A: First, check *AndroidManifest.xml* and make sure that your app has been given a label. This is where the app bar gets the app's name from.

Also, check that your activity calls the `setSupportActionBar()` method in its `onCreate()` method, as this sets the toolbar as the activity's app bar. Without it, the name of the app or activity won't get displayed in the toolbar.

Q: I've seen the <include> tag in some of the code that Android Studio has created for me. What does it do?

A: The `<include>` tag is used to include one layout inside another. Depending on what version of Android Studio you're using and what type of project you create, Android Studio may split your layout code into one or more separate layouts.

Add actions to the app bar

In most of the apps you create, you'll probably want to add actions to the app bar. These are buttons or text in the app bar that you click on to make something happen. We're going to add a "Create Order" button to the app bar. When you click on it, it will start a new activity we'll create called `OrderActivity`:

We'll create a new Create Order action that will start OrderActivity.

Create OrderActivity

We'll start by creating `OrderActivity`. Select the *com.hfad.bitsandpizzas* package in the *app/src/main/java* folder, then go to File→New...→Activity→Empty Activity. Name the activity "OrderActivity", name the layout "activity_order", make sure the package name is `com.hfad.bitsandpizzas`, and ***check* the Backwards Compatibility (AppCompat) checkbox.**

If prompted for the activity's source language, select the option for Java.

Sharpen your pencil

We want `OrderActivity` to display the same toolbar as `MainActivity`. See if you can complete the code for *activity_order.xml* below to display the toolbar.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.OrderActivity">
    ..............................................................
    ..............................................................
    ..............................................................

</LinearLayout>
```

The code for adding the toolbar needs to go here.

Sharpen your pencil Solution

We want `OrderActivity` to display the same toolbar as `MainActivity`. See if you can complete the code for *activity_order.xml* below to display the toolbar.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.OrderActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/toolbar" />

</LinearLayout>
```

This is the same code that we had in MainActivity. It includes the toolbar_main layout in activity_order.

Update activity_order.xml

We'll start by updating *activity_order.xml* so that it displays a toolbar. The toolbar will use the same layout we created earlier.

Here's our code; update yours so that it matches ours:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.OrderActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/toolbar" />

</LinearLayout>
```

Add the toolbar layout we created earlier.

Update OrderActivity.java

Next we'll update `OrderActivity` so that it uses the toolbar we set up in the layout as its app bar. To do this, we need to call the `setSupportActionBar()` method, passing in the toolbar as a parameter, just as we did before.

Here's the full code for *OrderActivity.java*; update your version of the code so that it matches ours:

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;

public class OrderActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_order);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

Make sure the activity extends AppCompatActivity.

Set the toolbar as the activity's app bar.

Add a String resource for the activity's title

Before we move on to creating an action to start `OrderActivity`, there's one more change we're going to make. We want to make it obvious to users when `OrderActivity` gets started, so we're going to change the text that's displayed in `OrderActivity`'s app bar to make it say "Create Order" rather than the name of the app.

To do this, we'll start by adding a String resource for the activity's title. Open the file *strings.xml* in the *app/src/main/res/values* folder, then add the following resource:

```
<string name="create_order">Create Order</string>
```

We'll use this to display "Create Order" in OrderActivity's app bar.

We'll update the text that gets displayed in the app bar on the next page.

Change the app bar text by adding a label

As you saw earlier in the chapter, you tell Android what text to display in the app bar by using the `label` attribute in file *AndroidManifest.xml*.

Here's our current code for *AndroidManifest.xml*. As you can see, the code includes a `label` attribute of `@string/app_name` inside the `<application>` element. This means that the name of the app gets displayed in the app bar for the entire app.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.bitsandpizzas">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            ...
        </activity>

        <activity android:name=".OrderActivity">
        </activity>

    </application>
</manifest>
```

The label attribute tells Android what text to display in the app bar.

This is the entry for MainActivity that we had before.

This is the entry for OrderActivity. Android added this for us when we created the new activity.

We want to override the label for `OrderActivity` so that the text "Create Order" gets displayed in the app bar whenever `OrderActivity` has the focus. To do this, we'll add a new `label` attribute to `OrderActivity`'s `<activity>` element to display the new text:

```
<activity
    android:name=".OrderActivity"
    android:label="@string/create_order">
</activity>
```

Adding a label to an activity means that for this activity, the activity's label gets displayed in its app bar instead of the app's label.

We'll show you this code in context on the next page.

The code for AndroidManifest.xml

Here's our code for *AndroidManifest.xml*. Update your code to reflect our changes.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.bitsandpizzas">
    <application
        ...
        android:label="@string/app_name"
        ...>

        <activity android:name=".MainActivity">
            ...
        </activity>

        <activity android:name=".OrderActivity"
            android:label="@string/create_order">
        </activity>

    </application>
</manifest>
```

The app's label is the default label for the entire app.

We don't need to change the code for MainActivity. MainActivity has no label of its own, so it will use the label in the <application> element.

Adding a label to OrderActivity overrides the app's label for this activity. It means that different text gets displayed in the app bar.

That's everything we need for `OrderActivity`. Next we'll look at how you add an action to the app bar so that we can start it.

How to add an action to an app bar

To add an action to the app bar, you need to do four things:

1. **Add resources for the action's icon and text.**
2. **Define the action in a menu resource file.**
   This tells Android what actions you want on the app bar.
3. **Get the activity to add the menu resource to the app bar.**
   You do this by implementing the `onCreateOptionsMenu()` method.
4. **Add code to say what the action should do when clicked.**
   You do this by implementing the `onOptionsItemSelected()` method.

We'll start by adding the action's icon and text resources.

1. Add the action's resources

When you add an action to an app bar, you generally assign it an icon and a short text title. The icon usually gets displayed if the action appears in the main area of the app bar. If the action doesn't fit in the main area, it's automatically moved to the app bar overflow, and the title appears instead.

We'll start with the icon.

This is the app bar's overflow. Android moves actions into the overflow that don't fit on the main area of the app bar.

Add the icon

If you want to display your action as an icon, you can either create your own icon from scratch or use one of the icons provided by Google. You can find the Google icons here: *https://material.io/icons/*.

We're going to use the "add" icon `ic_add_white_24dp`, and we'll add a version of it to our project's *drawable** folders, one for each screen density. Android will decide at runtime which version of the icon to use depending on the screen density of the device.

The icon for the new action

First, switch to the Project view of Android Studio's explorer if you haven't done so already, highlight the *app/src/main/res* folder, and then create folders called *drawable-hdpi*, *drawable-mdpi*, *drawable-xhdpi*, *drawable-xxhdpi*, and *drawable-xxxhdpi* if they're not already there. Then go to *https://git.io/v9oet*, and download the *ic_add_white_24dp.png* Bits and Pizzas images. Add the image in the *drawable-hdpi* folder to the *drawable-hdpi* folder in your project, then repeat this process for the other folders.

Add the action's title as a String resource

In addition to adding an icon for the action, we'll also add a title. This will get used if Android displays the action in the overflow area of the app bar, for example if there's no space for the action in the main area of the app bar.

We'll create the title as a String resource. Open the file *strings.xml* in the *app/src/main/res/values* folder, then add the following String resource:

```
<string name="create_order_title">Create Order</string>
```

We'll use this as the action item's title.

Now that we've added resources for the action's icon and title, we can create the menu resource file.

2. Create the menu resource file

A menu resource file tells Android what actions you want to appear on the app bar. Your app can contain multiple menu resource files. For example, you can create a separate menu resource file for each set of actions; this is useful if you want different activities to display different actions on their app bars.

We're going to create a new menu resource file called *menu_main.xml* in the folder *app/src/main/res/menu*. All menu resource files go in this folder.

Android Studio may have already created this file for you. If it has, simply replace its contents with the code below.

To create the menu resource file, select the *app/src/main/res* folder, go to the File menu, and choose New. Then choose the option to create a new Android resource file. You'll be prompted for the name of the resource file and the type of resource. Give it a name of "menu_main" and a resource type of "Menu", and make sure that the directory name is *menu*. When you click on OK, Android Studio will create the file for you , and add it to the *app/src/main/res/menu* folder.

Here's the code to add the new action. Replace the contents of *menu_main.xml* with the code below:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_create_order"
        android:title="@string/create_order_title"
        android:icon="@drawable/ic_add_white_24dp"
        android:orderInCategory="1"
        app:showAsAction="ifRoom" />
</menu>
```

The <menu> element identifies the file as a menu resource file.

The <item> element defines the action.

The menu resource file has a `<menu>` element at its root. Inside the `<menu>` element, you get a number of `<item>` elements, each one describing a separate action. In this particular case, we have a single action.

You use attributes of `<item>` to describe each action. The code creates an action with an `id` of `action_create_order`. This is so that we can refer to the action in our activity code, and respond to the user clicking on it.

The action includes a number of other attributes that determine how the action appears on the app bar, such as its icon and text. We'll look at these on the next page.

Control the action's appearance

Whenever you create an action to be displayed on the app bar, it's likely you'll want to display it as an icon. The icon can be any drawable resource. You set the icon using the `icon` attribute:

```
android:icon="@drawable/ic_add_white_24dp"
```

This is the name of the drawable resource we want to use as the icon.

Sometimes Android can't display the action's icon. This may be because the action has no icon, or because the action is displayed in the app bar overflow instead of in the main area. For this reason, it's a good idea to set an action's title so that the action can display a short piece of text instead of an icon. You set the action's title using the `title` attribute:

```
android:title="@string/create_order_title"
```

The title doesn't always get displayed, but it's a good idea to include it in case the action appears in the overflow.

If your app bar contains multiple actions, you might want to specify the order in which they appear. To do this, you use the `orderInCategory` attribute, which takes an integer value that reflects the action's order. Actions with a lower number will appear before actions with a higher number.

```
android:orderInCategory="1"
```

An action with an orderInCategory of 1 will appear before an action with an orderInCategory of 10.

Finally, the `showAsAction` attribute is used to say how you want the item to appear in the app bar. As an example, you can use it to get an item to appear in the overflow area rather than the main part of the app bar, or to place an item on the main app bar only if there's room. The `showAsAction` attribute can take the following values:

"ifRoom"	Place the item in the app bar if there's space. If there's not space, put it in the overflow.
"withText"	Include the item's title text.
"never"	Put the item in the overflow area, and never in the main app bar.
"always"	Always place the item in the main area of the app bar. This value should be used sparingly; if you apply this to many items, they may overlap each other.

There are other attributes for controlling an action's appearance, but these are the most common ones.

In our example, we want the action to appear on the main area of the app bar if there's room, so we're using:

```
app:showAsAction="ifRoom"
```

Our menu resource file is now complete. The next thing we need to do is implement the `onCreateOptionsMenu()` method in our activity.

3. Add the menu to the app bar with the onCreateOptionsMenu() method

Once you've created the menu resource file, you add the actions it contains to an activity's app bar by implementing the activity's `onCreateOptionsMenu()` method. This method runs when the app bar's menu gets created. It takes one parameter, a `Menu` object that's a Java representation of the menu resource file.

Here's our `onCreateOptionsMenu()` method for *MainActivity.java* (update your code to reflect our changes):

```
package com.hfad.bitsandpizzas;

import android.view.Menu;
...

public class MainActivity extends AppCompatActivity {

    ...

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the app bar.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return super.onCreateOptionsMenu(menu);
    }
}
```

The onCreateOptionsMenu() method uses the Menu class.

Implementing this method adds any items in the menu resource file to the app bar.

All onCreateOptionsMenu() methods generally look like this.

The line:

```
getMenuInflater().inflate(R.menu.menu_main, menu);
```

This is a Menu object that's a Java representation of the menu resource file.

This is the menu resource file.

inflates your menu resource file. This means that it creates a `Menu` object that's a Java representation of your menu resource file, and any actions the menu resource file contains are translated to `MenuItem`s. These are then added to the app bar.

There's one more thing we need to do: get our action to start `OrderActivity` when it's clicked. We'll do that on the next page.

4. React to action item clicks with the onOptionsItemSelected() method

To make your activity react when an action in the app bar is clicked, you implement the `onOptionsItemSelected()` method in your activity:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

The MenuItem object is the action on the app bar that was clicked.

Get the action's ID.

The `onOptionsItemSelected()` method runs whenever an action gets clicked. It takes one parameter, a `MenuItem` object that represents the action on the app bar that was clicked. You can use the `MenuItem`'s `getItemId()` method to get the ID of the action so that you can perform an appropriate action, such as starting a new activity.

We want to start `OrderActivity` when our action is clicked. Here's the code for the `onOptionsItemSelected()` method that will do this:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_create_order:
            //Code to run when the Create Order item is clicked
            Intent intent = new Intent(this, OrderActivity.class);
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

This intent is used to start OrderActivity when the Create Order action is clicked.

Returning true tells Android you've dealt with the item being clicked.

The full code for *MainActivity.java* is on the next page.

The full MainActivity.java code

Here's the full code for *MainActivity.java*. Update your code so that it matches ours. We've highlighted our changes.

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.content.Intent;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_create_order:
                Intent intent = new Intent(this, OrderActivity.class);
                startActivity(intent);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

These classes are used by the onOptionsItemSelected() method so we need to import them.

This method gets called when an action on the app bar is clicked.

Let's see what happens when we run the app.

Test drive the app

- [x] Basic app bar
- [x] Toolbar
- [x] Action
- [ ] Up button
- [ ] Share action

When you run the app, a new Create Order action is displayed in the `MainActivity` app bar. When you click on the action item, it starts `OrderActivity`.

But how do we get back to MainActivity?

To return to `MainActivity` from `OrderActivity`, we currently need to click on the Back button on our device. But what if we want to get back to it from the app bar?

One option would be to add an action to `OrderActivity`'s app bar that starts `MainActivity`, but there's a better way. We can get `OrderActivity` to return to `MainActivity` by enabling the Up button on `OrderActivity`'s app bar.

Enable Up navigation

If you have an app that contains a hierarchy of activities, you can enable the Up button on the app bar to let users navigate through the app using hierarchical relationships. As an example, `MainActivity` in our app includes an action on its app bar that starts a second activity, `OrderActivity`. If we enable the Up button on `OrderActivity`'s app bar, the user will be able to return to `MainActivity` by clicking on this button.

Up navigation may sound the same as using the Back button on the device, but it's different. The Back button allows users to work their way back through the history of activities they've been to. The Up button, on the other hand, is purely based on the app's hierarchical structure. If your app contains a lot of activities, implementing the Up button gives your users a quick and easy way to return to an activity's parent without having to keep pressing the Back button.

Use the Back button to navigate back to the previous activity.

Use the Up button to navigate up the app's hierarchy.

We're going to enable the Up button on `OrderActivity`'s app bar. When you click on it, it will display `MainActivity`.

Set an activity's parent

- Basic app bar
- Toolbar
- Action
- **Up button**
- Share action

The Up button enables the user to navigate up a hierarchy of activities in the app. You declare this hierarchy in *AndroidManifest.xml* by specifying the parent of each activity. As an example, we want the user to be able to navigate from `OrderActivity` to `MainActivity` when they press the Up button, so this means that `MainActivity` is the parent of `OrderActivity`.

For API level 16 and above, you specify the parent activity using the `android:parentActivityName` attribute. For older versions of Android, you need to include a `<meta-data>` element that includes the name of the parent activity. Here are both approaches in our *AndroidManifest.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.bitsandpizzas">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".OrderActivity"
            android:label="@string/create_order"
            android:parentActivityName=".MainActivity">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".MainActivity" />
        </activity>
    </application>

</manifest>
```

Apps at API level 16 or above use this line. It says that OrderActivity's parent is MainActivity.

You only need to add the <meta-data> element if you're supporting apps below API level 16. We've only included it so you can see what it looks like, but including it in your code won't do any harm.

Finally, we need to enable the Up button in `OrderActivity`.

Adding the Up button

You enable the Up button from within your activity code. You first get a reference to the app bar using the activity's `getSupportActionBar()` method. This returns an object of type `ActionBar`. You then call the `ActionBar` `setDisplayHomeAsUpEnabled()` method, passing it a value of `true`.

```
ActionBar actionBar = getSupportActionBar();
actionBar.setDisplayHomeAsUpEnabled(true);
```

If you enable the Up button for an activity, you MUST specify its parent.

If you don't, you'll get a null pointer exception when you call the setDisplayHomeAsUpEnabled() method.

We want to enable the Up button in `OrderActivity`, so we'll add the above code to the `onCreate()` method in *OrderActivity.java*. Here's our full activity code:

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.support.v7.app.ActionBar;

public class OrderActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCrate(savedInstanceState);
        setContentView(R.layout.activity_order);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

We're using the ActionBar class, so we need to import it. It comes from the AppCompat Support Library.

You need to use getSupportActionBar(), as we're using the toolbar from the Support Library.

This enables the Up button. Even though we're using a toolbar for our app bar, we need to use the ActionBar class for this method.

That's all the changes we need to make to enable the Up button. Let's see what happens when we run the app.

Test drive the app

- [x] Basic app bar
- [x] Toolbar
- [x] Action
- [x] **Up button**
- [ ] Share action

When you run your app and click on the Create Order action item, `OrderActivity` is displayed as before.

`OrderActivity` displays an Up button in its app bar. When you click on the Up button, it displays its hierarchical parent `MainActivity`.

So far, we've looked at how you add an app bar to your app and add basic actions to it. Next we'll look at how you add more sophisticated actions using **action providers**.

Sharing content on the app bar

The next thing we'll look at is how to add an action provider to your app bar. An action provider is an action that defines its own appearance and behavior.

We're going to concentrate on using the share action provider, which allows users to share content in your app with other apps such as Gmail. As an example, you could use it to let users send details of a particular pizza to one of their contacts.

The share action provider defines its own icon, so you don't have to add it yourself. When you click on it, it provides you with a list of apps you can use to share content. It adds a separate icon for the most commonly used app you choose to share content with.

This is what the share action looks like on the app bar. When you click on it, it gives you a list of apps that you can use to share content.

The share action also displays an icon for the app you most commonly share content with, in this case the Messenger app. This may not be visible at first.

You share the content with an intent

To get the share action provider to share content, you pass it an intent that defines the content you want to share, and its type. As an example, if you define an intent that passes text with an `ACTION_SEND` action, the share action will offer you a list of apps on your device that are capable of sharing text.

Here's how the share action works (you'll see this in action over the next two pages):

1. **Your activity creates an intent and passes it to the share action provider.**
   The intent describes the content that needs to be shared, its type, and an action.

2. **When the user clicks on the share action, the share action uses the intent to present the user with a list of apps that can deal with the intent.**
   The user chooses an app, and the share action provider passes the intent to the app's activity that can handle it.

Add a share action provider to menu_main.xml

You add a share action to the app bar by including it in the menu resource file.

To start, add a new `action_share` String to *strings.xml*. We'll use this String to add a title to the share action in case it appears in the overflow:

```
<string name="action_share">Share</string>
```

You add the share action to the menu resource file using the `<item>` element as before. This time, however, you need to specify that you're using a share action provider. You do this by adding an attribute of `app:actionProviderClass` and setting it to `android.support.v7.widget.ShareActionProvider`.

Here's the code to add the share action; update your copy of *menu_main.xml* to match ours:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_create_order"
        android:title="@string/create_order_title"
        android:icon="@drawable/ic_add_white_24dp"
        android:orderInCategory="1"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/action_share"
        android:title="@string/action_share"
        android:orderInCategory="2"
        app:showAsAction="ifRoom"
        app:actionProviderClass="android.support.v7.widget.ShareActionProvider" />

</menu>
```

Display the share action provider in the app bar if there's room.

This is the share action provider class. It comes from the AppCompat Support Library.

As we mentioned earlier, when you add a share action to your menu resource file, there's no need to include an icon. The share action provider already defines one.

Now that we've added the share action to the app bar, let's specify what content to share.

Specify the content with an intent

To get the share action to share content when it's clicked, you need to tell it what to share in your activity code. You do this by passing the share action provider an intent using its `setShareIntent()` method. Here's how you'd get the share action to share some default text when it's clicked:

We're using these extra classes, so we need to import them.

```
package com.hfad.bitsandpizzas;

...
import android.support.v7.widget.ShareActionProvider;
import android.support.v4.view.MenuItemCompat;

public class MainActivity extends AppCompatActivity {

    private ShareActionProvider shareActionProvider;

    ...
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        MenuItem menuItem = menu.findItem(R.id.action_share);
        shareActionProvider =
                    (ShareActionProvider) MenuItemCompat.getActionProvider(menuItem);
        setShareActionIntent("Want to join me for pizza?");
        return super.onCreateOptionsMenu(menu);
    }

    private void setShareActionIntent(String text) {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, text);
        shareActionProvider.setShareIntent(intent);
    }
}
```

Add a ShareActionProvider private variable.

Get a reference to the share action provider and assign it to the private variable. Then call the setShareActionIntent() method.

We created the setShareActionIntent() method. It creates an intent, and passes it to the share action provider using its setShareIntent() method.

You need to call the share action provider's `setShareIntent()` method whenever the content you wish to share has changed. As an example, if you're flicking through images in a photo app, you need to make sure you share the current photo.

We'll show you our full activity code on the next page, and then we'll see what happens when the app runs.

The full MainActivity.java code

Here's the full activity code for *MainActivity.java*. Update your code to reflect ours.

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.content.Intent;
import android.support.v7.widget.ShareActionProvider;
import android.support.v4.view.MenuItemCompat;

public class MainActivity extends AppCompatActivity {

    private ShareActionProvider shareActionProvider;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        MenuItem menuItem = menu.findItem(R.id.action_share);
        shareActionProvider =
                (ShareActionProvider) MenuItemCompat.getActionProvider(menuItem);
        setShareActionIntent("Want to join me for pizza?");
        return super.onCreateOptionsMenu(menu);
    }
```

We're using these extra classes, so we need to import them.

This is the default text that the share action should share.

The code continues on the next page.

The MainActivity.java code (continued)

```
    private void setShareActionIntent(String text) {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, text);
        shareActionProvider.setShareIntent(intent);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_create_order:
                //Code to run when the Create Order item is clicked
                Intent intent = new Intent(this, OrderActivity.class);
                startActivity(intent);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

This sets the default text in the share action provider.

On the next page we'll check what happens when the code runs by taking the app for a test drive.

Test drive the app

- ☑ Basic app bar
- ☑ Toolbar
- ☑ Action
- ☑ Up button
- ☑ **Share action**

When you run the app, the share action is displayed in the app bar:

The share action provider also added the Messenger icon to our app bar. We usually share this app's content with the Messenger app, so the share action gave us a shortcut.

When you click on the share action, it gives you a list of apps to choose from that can accept the intent you want to share:

The intent we passed to the share action provider says we want to share text using ACTION_SEND. It displays a list of apps that can do this.

When you choose an app to share content with, the app gets launched and the default text is shared with it:

Your Android Toolbox

You've got Chapter 8 under your belt and now you've added Android Support Libraries and app bars to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- You add a basic app bar by applying a theme that contains one.
- The Android Support Libraries provide backward compatibility with older versions of Android.
- The `AppCompatActivity` class is a type of activity that resides in the v7 AppCompat Support Library. In general, your activity needs to extend the `AppCompatActivity` class whenever you want an app bar that provides backward compatibility with older versions of Android.
- The `android:theme` attribute in *AndroidManifest.xml* specifies which theme to apply.
- You define styles in a style resource file using the `<style>` element. The `name` attribute gives the style a name. The `parent` attribute specifies where the style should inherit its properties from.
- The latest app bar features are in the `Toolbar` class in the v7 AppCompat Support Library. You can use a toolbar as your app bar.
- Add actions to your app bar by adding them to a menu resource file.
- Add the items in the menu resource file to the app bar by implementing the activity's `onCreateOptionsMenu()` method.
- You determine what items should do when clicked by implementing the activity's `onOptionsItemSelected()` method.
- Add an Up button to your app bar to navigate up the app's hierarchy. Specify the hierarchy in *AndroidManifest.xml*. Use the `ActionBar setDisplayHomeAsUpEnabled()` method to enable the Up button.
- You can share content by adding the share action provider to your app bar. Add it by including it in your menu resource file. Call its `setShareIntent()` method to pass it an intent describing the content you wish to share.

9 fragments

Make It Modular

You've seen how to create apps that work in the same way no matter what device they're running on.

But what if you want your app to look and behave differently depending on whether it's running on a *phone* or a *tablet*? In this case you need **fragments**, modular code components that can be **reused by different activities**. We'll show you how to create **basic fragments** and **list fragments**, how to **add them to your activities**, and how to get your fragments and activities to **communicate** with one another.

Your app needs to look great on ALL devices

One of the great things about Android development is that you can put the exact same app on devices with completely different screen sizes and processors, and have them run in exactly the same way. But that doesn't mean that they always have to *look* exactly the same.

On a phone:

Take a look at this image of an app on a phone. It displays a list of workouts, and when you click on one, you are shown the details of that workout.

Click on an item in a list, and it launches a second activity.

On a tablet:

On a larger device, like a tablet, you have a lot more screen space available, so it would be good if all the information appeared on the same screen. On the tablet, the list of workouts only goes partway across the screen, and when you click on an item, the details appear on the right.

There's a lot more space on a tablet, so we can use the space in a different way.

To make the phone and tablet user interfaces look different from each other, you can use separate layouts for large devices and small devices.

Your app may need to behave differently too

It's not enough to simply have different layouts for different devices. You also need *different Java code* to run alongside the layouts so that the app can behave differently depending on the device. In our Workout app, for instance, we need to provide **one activity for tablets**, and **two activities for phones**.

On a phone:

On a tablet:

But that means you might duplicate code

The second activity that runs only on phones will need to insert the details of a workout into the layout. But that code will also need to be available in the main activity for when the app is running on a tablet. *The same code needs to be run by multiple activities.*

Rather than duplicate the code in the two activities, we can use **fragments**. So what's a fragment?

Fragments allow you to reuse code

Fragments are like reusable components or subactivities. A fragment is used to control part of a screen, and can be reused between screens. This means we can create a fragment for the list of workouts, and a fragment to display the details of a single workout. These fragments can then be shared between layouts.

A fragment has a layout

Just like an activity, a fragment has an associated layout. If you design it carefully, the fragment's Java code can be used to control everything within the interface. If the fragment code contains all that you need to control its layout, it greatly increases the chances that you'll be able to reuse it elsewhere in the app.

We're going to show you how to create and use fragments by building the Workout app.

The phone version of the app

We're going to build the phone version of the app in this chapter, and reuse the fragments we create to build the tablet version of the app in Chapter 10. Here's how the phone version of the app will work:

Don't worry if the app structure looks complex.

We'll work through it step by step over the course of this chapter.

1. **When the app gets launched, it starts MainActivity.**
   `MainActivity` uses *activity_main.xml* for its layout, and contains a fragment called `WorkoutListFragment`.

2. **WorkoutListFragment displays a list of workouts.**

3. **When the user clicks on one of the workouts, DetailActivity starts.**
   `DetailActivity` uses *activity_detail.xml* for its layout, and contains a fragment called `WorkoutDetailFragment`.

4. **WorkoutDetailFragment uses fragment_workout_detail.xml for its layout.**
   It displays the details of the workout the user has selected.

5. **WorkoutListFragment and WorkoutDetailFragment get their workout data from *Workout.java*.**
   Workout.java contains an array of `Workout`s.

We'll go through the steps for creating this app on the next page.

Here's what we're going to do

There are three main steps we'll go through to build the app:

1. **Create WorkoutDetailFragment.** `WorkoutDetailFragment` displays the details of a specific workout. We'll start by creating two activities, `MainActivity` and `DetailActivity`, and then we'll add `WorkoutDetailFragment` to `DetailActivity`. We'll also get `MainActivity` to launch `DetailActivity` when a button is pressed. We'll also add a plain old Java class, *Workout.java*, that will provide the data for `WorkoutDetailFragment`.

2. **Create WorkoutListFragment.** `WorkoutListFragment` displays a list of workouts. We'll add this fragment to `MainActivity`.

3. **Coordinate the fragments to display the correct workout.** When the user clicks on an item in `WorkoutListFragment`, we'll start `DetailActivity` and get `WorkoutDetailFragment` to display details of the workout the user selected.

Let's get started.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Create the project and activities

We're going to start by creating a project that contains two activities, `MainActivity` and `DetailActivity`. `MainActivity` will be used for the fragment that displays a list of workouts, and `DetailActivity` will be used for the fragment that displays details of one particular workout.

To do this, first create a new Android project with an empty activity for an application named "Workout" with a company domain of "hfad.com", making the package name `com.hfad.workout`. The minimum SDK should be API 19 so that it works on most devices. Name the activity "MainActivity" and name the layout "activity_main". **Make sure you check the Backwards Compatibility (AppCompat) checkbox.**

Next, create a second empty activity by highlighting the *com.hfad.workout* package in the *app/src/main/java* folder, and going to File→New...→Activity→Empty Activity. Name the activity "DetailActivity", name the layout "activity_detail", make sure the package name is `com.hfad.workout`, and **check the Backwards Compatibility (AppCompat) checkbox.**

If prompted for the activity's source language, select the option for Java.

Add the AppCompat Support Library

We're going to be using activities and fragments from the v7 AppCompat Library, which means you need to make sure the library has been added to your project as a dependency. To do this, go to the File menu and choose Project Structure. Then click on the app module and choose Dependencies.

This is the v7 AppCompat Support Library.

If Android Studio has already added the v7 AppCompat Support Library to your project, you'll see it listed in the list of dependencies. If it's not there, you'll need to add it yourself. To do this, click on the "+" button at the bottom or right side of the screen. When prompted, choose the Library Dependency option, then select the appcompat-v7 library from the list of options. Finally, use the OK buttons to save your changes.

Once you've made sure the v7 AppCompat Support Library has been added, you can close the Project Structure window. On the next page we'll update `MainActivity`.

Add a button to MainActivity's layout

We're going to add a button to `MainActivity` that will start `DetailActivity`. This is because we're going to work on the fragment for `DetailActivity` first, and adding a button to `MainActivity` will give us an easy way of navigating from `MainActivity` to `DetailActivity`.

We'll start by adding the button to the layout. Open file *activity_main.xml*, then update your code so that it matches ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.workout.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onShowDetails"
        android:text="@string/details_button" />
</LinearLayout>
```

This is the button we're adding.

The button calls the onShowDetails() method in MainActivity when it's clicked. We need to write this method.

The button uses a String resource for its text, so we need to add it to the String resource file. Open file *strings.xml*, then add the following String resource:

```
<resources>
    ...
    <string name="details_button">Show details</string>
</resources>
```

This text will be displayed on the button.

When the button is clicked, we've specified that `MainActivity`'s `onShowDetails()` method should be called. We'll write the code for this method next.

Make the button respond to clicks

We need to get `MainActivity`'s button to start `DetailActivity` when it's clicked. To do this, we'll add a method called `onShowDetails()` to `MainActivity`. The method will start `DetailActivity` using an intent, just as we've done in previous chapters.

Here's the full code for *MainActivity.java*. Update your code so that it matches ours.

```
package com.hfad.workout;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.content.Intent;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onShowDetails(View view) {
        Intent intent = new Intent(this, DetailActivity.class);
        startActivity(intent);
    }
}
```

The activity extends AppCompatActivity.

This method is called when the button is clicked. It starts DetailActivity.

That's everything we need to get `MainActivity` to start `DetailActivity`. On the next page we'll add a new fragment to our project called `WorkoutDetailFragment` that we'll then add to `DetailActivity`.

How to add a fragment to your project

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

We're going to add a new fragment called `WorkoutDetailFragment` to the project to display details of a single workout. You add a new fragment in a similar way to how you add a new activity: by switching to the Project view of Android Studio's explorer, highlighting the *com.hfad.workout* package in the *app/src/main/java* folder, going to the File menu, and choosing New...→Fragment→Fragment (Blank).

You will be asked to choose options for your new fragment. Name the fragment "WorkoutDetailFragment", check the option to create layout XML for it, and give the fragment layout a name of "fragment_workout_detail". Uncheck the options to include fragment factory methods and interface callbacks; these options generate extra code that you don't need to use. When you're done, click on the Finish button.

We suggest looking at the extra code Android generates for you **after** you've finished this book. You might find some of it useful depending on what you want to do.

When you click on the Finish button, Android Studio creates your new fragment and adds it to the project.

What fragment code looks like

When you create a fragment, Android Studio creates two files for you: Java code for the fragment itself, and XML code for the fragment's layout. The Java code describes the fragment's behavior, and the layout describes the fragment's appearance.

We'll look at the Java code first. Go to the *com.hfad.workout* package in the *app/src/main/java* folder and open the file *WorkoutDetailFragment.java* that Android Studio just created for us. Then replace the code that Android Studio generated with the code below:

```
package com.hfad.workout;

import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }
}
```

We're using the Fragment class from the Android Support Library.

WorkoutDetailFragment extends the Fragment class.

This is the onCreateView() method. It's called when Android needs the fragment's layout.

This tells Android which layout the fragment uses (in this case, it's fragment_workout_detail).

The above code creates a basic fragment. As you can see, the code for a fragment looks very similar to activity code.

To create a fragment, you first need to extend the `Fragment` class or one of its subclasses. We're using fragments from the Support Library, so our fragment needs to extend the `android.support.v4.app.Fragment` class. This is because the Support Library fragments are backward compatible with earlier versions of Android, and contain the latest fragment features.

The fragment implements the `onCreateView()` method, which gets called each time Android needs the fragment's layout, and it's where you say which layout the fragment uses. The `onCreateView()` method is optional, but as you need to implement it whenever you're creating a fragment with a layout, you'll need to implement it nearly every time you create a fragment. We'll look at this method in more detail on the next page.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

The fragment's onCreateView() method

The `onCreateView()` method returns a `View` object that represents the fragment's user interface. It gets called when Android is ready to instantiate the user interface, and it takes three parameters:

The onCreateView() method gets called when Android needs the fragment's layout.

```
public View onCreateView(LayoutInflater inflater,
                         ViewGroup container,
                         Bundle savedInstanceState) {
}
```

The first parameter is a `LayoutInflator` that you can use to inflate the fragment's layout. Inflating the layout turns your XML views into Java objects.

The second parameter is a `ViewGroup`. This is the `ViewGroup` in the activity's layout that will contain the fragment.

The final parameter is a `Bundle`. This is used if you've previously saved the fragment's state, and want to reinstate it.

You specify the fragment's layout using the `LayoutInflator`'s `inflate()` method:

```
public View onCreateView(LayoutInflater inflater,
                         ViewGroup container,
                         Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_workout_detail,
                            container,
                            false);
}
```

This inflates the fragment's layout from XML to Java objects.

This is the fragment equivalent of calling an activity's `setContentView()` method. You use it to say what layout the fragment should use, in this case `R.layout.fragment_workout_detail`.

The `inflate()` method's container argument specifies the `ViewGroup` in the activity that the fragment's layout needs to be inserted into. It gets passed to the fragment as the second parameter in the fragment's `onCreateView()` method.

Now that you've seen the fragment's code, let's have a look at its layout.

All fragments must have a public no-argument constructor.

Android uses it to reinstantiate the fragment when needed, and if it's not there, you'll get a runtime exception.

In practice, you only need to add a public no-argument constructor to your fragment code if you include ***another*** *constructor with one or more arguments. This is because if a Java class contains no constructors, the Java compiler automatically adds a public no-argument constructor for you.*

Fragment layout code looks just like activity layout code

As we said earlier, fragments use layout files to describe their appearance. Fragment layout code looks just like activity layout code, so when you write your own fragment layout code, you can use any of the views and layouts you've already been using to write activity layout code.

We're going to update our layout code so that our fragment contains two text views, one for the workout title and one for the workout description.

Open the file *fragment_workout_detail.xml* in the *app/src/res/layout* folder, and replace its contents with the code below:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/workout_title"
        android:id="@+id/textTitle" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/workout_description"
        android:id="@+id/textDescription" />
</LinearLayout>
```

We're using a LinearLayout for our fragment, but we could have used any of the other layout types we've looked at instead.

We'll display the workout title and description in two separate TextViews.

This makes the text large.

Static String resources.

For now we're using static String resources for the title and description of the workout so that we can see our fragment working. Open *strings.xml*, and add the following String resources:

```
<resources>
    ...
    <string name="workout_title">Title</string>
    <string name="workout_description">Description</string>
</resources>
```

We'll use these to display default text in our fragment.

That's everything we need for our fragment. On the next page we'll look at how you add the fragment to an activity.

Add a fragment to an activity's layout

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

We're going to add our `WorkoutDetailFragment` to `DetailActivity` so that the fragment gets displayed in the activity's layout. To do this, we're going to add a `<fragment>` element to `DetailActivity`'s layout.

The `<fragment>` element is a view that specifies the name of the fragment you want to display. It looks like this:

```
<fragment
    android:name="com.hfad.workout.WorkoutDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

This is the full name of the fragment class.

You specify the fragment using the `android:name` attribute and giving it a value of the fully qualified name of the fragment. In our case, we want to display a fragment called `WorkoutDetailFragment` that's in the `com.hfad.workout` package, so we use:

```
android:name="com.hfad.workout.WorkoutDetailFragment"
```

When Android creates the activity's layout, it replaces the `<fragment>` element with the `View` object returned by the fragment's `onCreateView()` method. This view is the fragment's user interface, so the `<fragment>` element is really a placeholder for where the fragment's layout should be inserted.

You add the `<fragment>` element to your layout in the same way that you add any other element. As an example, here's how you'd add the fragment to a linear layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:name="com.hfad.workout.WorkoutDetailFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

This adds the fragment to the activity's layout.

If your layout only contains a single fragment with no other views, however, you can simplify your layout code.

Simplify the layout

If the layout code for your activity comprises a single fragment contained within a layout element with no other views, you can simplify your layout code by removing the root layout element.

Each layout file you create must have a single root element that's either a view or view group. This means that if your layout contains multiple items, these items must be contained within a view group such as a linear layout.

If your layout contains a single fragment, the `<fragment>` element can be the layout file's root. This is because the `<fragment>` element is a type of view, and Android replaces it with the layout of the fragment at runtime.

In our code example on the previous page, we showed you a fragment contained within a linear layout. As there are no other views in the layout, we can remove the linear layout so that our code looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

You can add your fragment to your layout like this if your layout ONLY contains a single fragment.

A layout file requires a single view or view group as its root element. If your activity only contains a fragment, the fragment itself can be the root element.

This code does exactly the same thing as the code on the previous page, but it's much shorter.

That's the only code we need for `DetailActivity`'s layout, so replace the code you have in your version of *activity_detail.xml*, with the code above and save your changes.

On the next page we'll look at the code for the activity itself.

Support Library fragments need activities that extend FragmentActivity

When you add a fragment to an activity, you usually need to write code that controls any interactions between the fragment and the activity. You'll see examples of this later in the chapter.

Currently, `WorkoutDetailFragment` only contains static data. `DetailActivity` only has to display the fragment, and doesn't need to interact with it, so this means that we don't need to write any extra activity code to control the interaction.

There's one important point to be aware of, however. When you're using fragments from the Support Library, as we are here, you must make sure that **any activity you want to use them with extends the `FragmentActivity` class, or one of its subclasses**. The `FragmentActivity` class is designed to work with Support Library fragments, and if your activity doesn't extend this class, your code will break.

In practice, this isn't a problem. This is because the `AppCompatActivity` class is a subclass of `FragmentActivity`, so as long as your activity extends the `AppCompatActivity` class, your Support Library fragments will work.

As long as your activity extends FragmentActivity, or one of its subclasses such as AppCompatActivity, you can use fragments from the Support Library.

Here's the code for *DetailActivity.java*. Update your code so that it matches ours below:

```
package com.hfad.workout;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class DetailActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
    }
}
```

DetailActivity extends AppCompatActivity.

That's everything we need to display the `WorkoutDetailFragment` in our activity. Let's see what happens when we run the app.

What the code does

Before we take the app for a test drive, let's go through what happens when the code runs.

1. **When the app is launched, activity MainActivity gets created.**
The user clicks on the button in `MainActivity` to start `DetailActivity`.

2. **DetailActivity's onCreate() method runs.**
The `onCreate()` method specifies that *activity_detail.xml* should be used for `DetailActivity`'s layout.

3. **activity_detail.xml sees that it includes a <fragment> element that refers to WorkoutDetailFragment.**

4. **WorkoutDetailFragment's onCreateView() method is called.**
The `onCreateView()` method specifies that *fragment_workout_detail.xml* should be used for `WorkoutDetailFragment`'s layout. It inflates the layout to a `View` object.

The story continues

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

5 **activity_detail.xml's Views are inflated to View Java objects.**
`DetailActivity` layout uses `WorkoutDetailFragment`'s `View` object in place of the `<fragment>` element in its layout's XML.

activity_detail.xml only contains the <fragment> element. This means that when it's inflated, it only contains WorkoutDetailFragment's View object.

6 **Finally, DetailActivity is displayed on the device.**
Its layout contains the fragment `WorkoutDetailFragment`.

Our WorkoutDetailFragment is displayed in DetailActivity. You'll see this better in the test drive below.

Test drive the app

When we run our app, `MainActivity` gets launched.

When we click on `MainActivity`'s button, it starts `DetailActivity`. `DetailActivity` contains `WorkoutDetailFragment`, and we see this on the device.

Click on the button...

...and placeholder text for the workout title and description is displayed.

BE the Layout

Your job is to play like you're the layout and say whether each of these layouts is valid or invalid and why. Assume that any fragments or String resources referred to in the layout already exist.

A

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

B

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/details_button" />
```

C

```
<?xml version="1.0" encoding="utf-8"?>
<Button
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/details_button" />
```

BE the Layout Solution

Your job is to play like you're the layout and say whether each of these layouts is valid or invalid and why. Assume that any fragments or String resources referred to in the layout already exist.

A

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

This layout is valid, as it consists of a single fragment.

B

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/details_button" />
```

This layout is invalid. A layout must have a single View or ViewGroup as its root element. To make this layout valid, you would need to put the fragment and Button in a ViewGroup.

C

```
<?xml version="1.0" encoding="utf-8"?>
<Button
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/details_button" />
```

This layout is valid as it has a single View, in this case a Button, as its root element.

Get the fragment and activity to interact

So far we've looked at how you add a basic fragment to an activity. The next thing we'll look at is how you get the fragment and activity to interact.

To do this, we'll start by changing `WorkoutDetailFragment` so that it displays details of a workout instead of the placeholder text we have currently.

The first thing we'll do is update the fragment's layout to remove the static text that's currently displayed. Open file *fragment_workout_detail.xml*, then update the code to match our changes below:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/workout_title"
        android:id="@+id/textTitle" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/workout_description"
        android:id="@+id/textDescription" />
</LinearLayout>
```

Delete both these lines.

On the next page we'll add a new class to our project to hold the workout data.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

The Workout class

We're going to hold our workout data in a file called *Workout.java*, which is a pure Java class file that the app will get workout data from. The class defines an array of four workouts, where each workout is composed of a name and description. Select the *com.hfad.workout* package in the *app/src/main/java* folder in your project, then go to File→New...→Java Class. When prompted, name the class "Workout", and make sure the package name is `com.hfad.workout`. Then replace the code in *Workout.java* with the following, and save your changes.

```
package com.hfad.workout;

public class Workout {
    private String name;
    private String description;

    public static final Workout[] workouts = {
            new Workout("The Limb Loosener",
                    "5 Handstand push-ups\n10 1-legged squats\n15 Pull-ups"),
            new Workout("Core Agony",
                    "100 Pull-ups\n100 Push-ups\n100 Sit-ups\n100 Squats"),
            new Workout("The Wimp Special",
                    "5 Pull-ups\n10 Push-ups\n15 Squats"),
            new Workout("Strength and Length",
                    "500 meter run\n21 x 1.5 pood kettleball swing\n21 x pull-ups")
    };

    //Each Workout has a name and description
    private Workout(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return this.name;
    }
}
```

Each Workout has a name and description.

workouts is an array of four Workouts.

These are getters for the private variables.

The String representation of a Workout is its name.

The data will be used by the fragment to display details of a particular workout. We'll look at this next.

Pass the workout ID to the fragment

When you have an activity that uses a fragment, the activity will usually need to talk to the fragment in some way. As an example, if you have a fragment that displays detail records, you need the activity to tell the fragment which record it needs to display details of.

In our case, we need `WorkoutDetailFragment` to display details of a particular workout. To do this, we'll add a simple setter method to the fragment that sets the value of the workout ID. The activity will then be able to use this method to set the workout ID. Later on, we'll use the workout ID to update the fragment's views.

Here's the revised code for `WorkoutDetailFragment` (update your code with our changes):

```
package com.hfad.workout;

import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

This is the ID of the workout the user chooses. Later, we'll use it to set the values of the fragment's views with the workout details.

This is a setter method for the workout ID. The activity will use this method to set the value of the workout ID.

We need to get the `DetailActivity` to call the fragment's `setWorkout()` method and pass it the ID of a particular workout. In order to do this, the activity must get a reference to the fragment. But how?

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Use the fragment manager to manage fragments

Before an activity can talk to its fragment, the activity first needs to get a reference to the fragment. You get a reference to an activity's fragments using the activity's **fragment manager**. The fragment manager is used to keep track of and deal with any fragments used by the activity.

There are two methods for getting a reference to the fragment manager, `getFragmentManager()` and `getSupportFragmentManager()`. The `getSupportFragmentManager()` method gets a reference to the fragment manager that deals with fragments from the Support Library like ours, and the `getFragmentManager()` method gets a reference to the fragment manager that deals with fragments that use the native Android fragment class instead. You then use the fragment manager's `findFragmentById()` method to get a reference to the fragment.

We're using fragments from the Support Library, so we're going to use the `getSupportFragmentManager()` method like this:

This is the ID of the fragment in the activity's layout.

```
getSupportFragmentManager().findFragmentById(R.id.fragment_id)
```

findFragmentById() is a bit like findViewById() except you use it to get a reference to a fragment.

We're going to use `DetailActivity`'s fragment manager to get a reference to its `WorkoutDetailFragment`. In order to do this, we first need to assign an ID to the fragment.

You assign an ID to an activity's fragment in the activity's layout. Open the file *activity_detail.xml*, then add an ID to the activity's fragment by adding the line of code highlighted below:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutDetailFragment"
    android:id="@+id/detail_frag"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Add an ID to the fragment.

The above code gives the fragment an ID of `detail_frag`. On the next page we'll use the ID to get a reference to the fragment.

Get the activity to set the workout ID

To get a reference to the fragment, we need to add the following code:

```
WorkoutDetailFragment frag = (WorkoutDetailFragment)
        getSupportFragmentManager().findFragmentById(R.id.detail_frag);
```

We can then call the fragment's `setWorkout()` method to tell the fragment which workout we want it to display details for. For now, we'll hardcode which workout we want it to display so that we can see it working. Later on, we'll change the code so that the user can select which workout she wants to see.

Here's our revised code for *DetailActivity.java*. Update your code to reflect our changes:

```
package com.hfad.workout;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class DetailActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        WorkoutDetailFragment frag = (WorkoutDetailFragment)
                getSupportFragmentManager().findFragmentById(R.id.detail_frag);
        frag.setWorkout(1);
    }
}
```

We're going to get WorkoutDetailFragment to display details of a workout here to check that it's working.

This gets us a reference to WorkoutDetailFragment. Its id in the activity's layout is detail_frag.

As you can see, we've got a reference to the fragment after calling `setContentView()`. Getting the reference here is really important, because before this point, the fragment won't have been created.

The next thing we need to do is get the fragment to update its views when the fragment is displayed to the user. Before we can do this, we need to understand the fragment's lifecycle so that we add our code to the correct method in the fragment.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Activity states revisited

Just like an activity, a fragment has a number of key lifecycle methods that get called at particular times. It's important to know what these methods do and when they get called so your fragment works in just the way you want.

Fragments are contained within and controlled by activities, so the fragment lifecycle is closely linked to the activity lifecycle. Here's a reminder of the different states an activity goes through, and on the next page we'll show you how these relate to fragments.

The activity is created when its onCreate() method runs.
At this point, the activity is initialized, but isn't visible.

The activity is started when its onStart() method runs.
The activity is visible, but doesn't have the focus.

The activity is resumed when its onResume() method runs.
The activity is visible, and has the focus.

The activity is paused when its onPause() method runs.
The activity is still visible, but no longer has the focus.

The activity is stopped when its onStop() method runs.
The activity is no longer visible, but still exists.

The activity is destroyed when its onDestroy() method runs.
The activity no longer exists.

The fragment lifecycle

A fragment's lifecycle is very similar to an activity's, but it has a few extra steps. This is because it needs to interact with the lifecycle of the activity that contains it. Here are the fragment lifecycle methods, along with where they fit in with the different activity states.

Activity states	Fragment callbacks
Activity created	onAttach()
	onCreate()
	onCreateView()
	onActivityCreated()
Activity started	onStart()
Activity resumed	onResume()
Activity paused	onPause()
Activity stopped	onStop()
Activity destroyed	onDestroyView()
	onDestroy()
	onDetach()

onAttach(Context)
This happens when the fragment is associated with a context, in this case an activity.

onCreate(Bundle)
This is very similar to the activity's `onCreate()` method. It can be used to do the initial setup of the fragment.

onCreateView(LayoutInflater, ViewGroup, Bundle)
Fragments use a layout inflater to create their view at this stage.

onActivityCreated(Bundle)
Called when the `onCreate()` method of the activity has completed.

onStart()
Called when the fragment is about to become visible.

onResume()
Called when the fragment is visible and actively running.

onPause()
Called when the fragment is no longer interacting with the user.

onStop()
Called when the fragment is no longer visible to the user.

onDestroyView()
Gives the fragment the chance to clear away any resources that were associated with its view.

onDestroy()
In this method, the fragment can clear away any other resources it created.

onDetach()
Called when the fragment finally loses contact with the activity.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Fragments inherit lifecycle methods

As you saw earlier, our fragment extends the Android `fragment` class. This class gives our fragment access to the fragment lifecycle methods. Here's a diagram showing the class hierarchy.

Object class
(java.lang.Object)

Fragment class
(android.support.v4.app.Fragment)
The `Fragment` class implements default versions of the lifecycle methods. It also defines other methods that fragments need, such as `getView()`.

OurFragment class
(com.hfad.foo)
Most of the behavior of our fragment is handled by superclass methods our fragment inherits. All you do is override the methods you need.

Even though fragments have a lot in common with activities, the `Fragment` class doesn't extend the `Activity` class. This means that some methods that are available to activities aren't available to fragments.

Note that the `Fragment` class doesn't implement the `Context` class. Unlike an activity, a fragment isn't a type of context and therefore doesn't have direct access to global information about the application environment. Instead, fragments must access this information using the context of other objects such as its parent activity.

Now that you understand the fragment's lifecycle better, let's get back to getting `WorkoutDetailFragment` to update its views.

Set the view's values in the fragment's onStart() method

We need to get `WorkoutDetailFragment` to update its views with details of the workout. We need to do this when the activity becomes visible, so we'll use the fragment's `onStart()` method. Update your code to match ours:

```
package com.hfad.workout;

import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

We're using this class in the onStart() method.

The getView() method gets the fragment's root View. We can then use this to get references to the workout title and description text views.

As we said on the previous page, fragments are distinct from activities, and therefore don't have all the methods that an activity does. Fragments don't include a `findViewById()` method, for instance. To get a reference to a fragment's views, we first have to get a reference to the fragment's root view using the `getView()` method, and use that to find its child views.

Now that we've got the fragment to update its views, let's take the app for a test drive.

You should always call up to the superclass when you implement any fragment lifecycle methods.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

What happens when the code runs

Before we run the app, let's go through what happens when the code runs.

1. **When the app is launched, MainActivity gets created.**
The user clicks on the button in `MainActivity` to start `DetailActivity`.

2. **DetailActivity's onCreate() method runs.**
The `onCreate()` method specifies that *activity_detail.xml* should be used for `DetailActivity`'s layout. *activity_detail.xml* includes a `<fragment>` element with an ID of `detail_frag` that refers to the fragment `WorkoutDetailFragment`.

3. **WorkoutDetailFragment's onCreateView() method runs.**
The `onCreateView()` method specifies that *fragment_workout_detail.xml* should be used for `WorkoutDetailFragment`'s layout. It inflates the layout to a `View` object.

The story continues

4 **activity_detail.xml's Views are inflated to View Java objects.**
`DetailActivity` uses `WorkoutDetailFragment`'s `View` object in place of the `<fragment>` element in its layout's XML, and gives it an ID of `detail_frag`.

5 **DetailActivity's onCreate() method continues to run.**
`DetailActivity` gets a reference to `WorkoutDetailFragment` by asking the fragment manager for the fragment with an ID of `detail_frag`.

5 **DetailActivity calls WorkoutDetailFragment's setWorkout() method.**
`DetailActivity` passes `WorkoutDetailFragment` a workout ID of `1`. The fragment sets its `workoutId` variable to `1`.

6 **The fragment uses the value of the workout ID in its onStart() method to set the values of its views.**

Let's take the app for a test drive.

Test drive the app

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

When we run the app, `MainActivity` is launched.

When we click on `MainActivity`'s button, it starts `DetailActivity`. `DetailActivity` contains `WorkoutDetailFragment`, and the fragment displays details of the Core Agony workout.

there are no Dumb Questions

Q: Why can't an activity get a fragment by calling the `findViewById()` method?

A: Because `findViewById()` always returns a `View` object and, surprisingly, fragments aren't views.

Q: Why isn't `findFragmentById()` an activity method like `findViewById()` is?

A: That's a good question. Fragments weren't available before API 11, so it uses the fragment manager as a way to add a whole bunch of useful code for managing fragments, without having to pack lots of extra code into the activity base class.

Q: Why don't fragments have a `findViewById()` method?

A: Because fragments aren't views or activities. Instead, you need to use the fragment's `getView()` method to get a reference to the fragment's root view, and then call the view's `findViewById()` method to get its child views.

Q: Activities need to be registered in *AndroidManifest.xml* so that the app can use them. Do fragments?

A: No. Activities need to be registered in *AndroidManifest.xml*, but fragments don't.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Where we've got to

Here's a reminder of the structure of the app, and what we want it to do:

1. **When the app gets launched, it starts MainActivity.**
   `MainActivity` uses *activity_main.xml* for its layout, and contains a fragment called `WorkoutListFragment`.

2. **WorkoutListFragment displays a list of workouts.**

3. **When the user clicks on one of the workouts, DetailActivity starts.**
   `DetailActivity` uses *activity_detail.xml* for its layout, and contains a fragment called `WorkoutDetailFragment`.

4. **WorkoutDetailFragment uses fragment_workout_detail.xml for its layout.**
   It displays the details of the workout the user has selected.

5. **WorkoutListFragment and WorkoutDetailFragment get their workout data from *Workout.java*.**
   Workout.java contains an array of `Workout`s.

So far we've created both activities and their layouts, *WorkoutDetailFragment.java* and its layout, and also *Workout.java*. The next thing we need to look at is `WorkoutListFragment`.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

We need to create a fragment with a list

We need to create a second fragment, `WorkoutListFragment`, that contains a list of the different workouts that the user can choose from. Using a fragment for this means that later on, we'll be able to use it to create different user interfaces for phones and tablets.

You've already seen how to add a list view to an activity, so we could do something similar for the fragment. But rather than create a new fragment with a layout that contains a single list view, we're going to use a different approach that involves a new type of fragment called a **list fragment**.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

A list fragment is a fragment that contains only a list

A list fragment is a type of fragment that specializes in working with a list. It's automatically bound to a list view, so you don't need to create one yourself. Here's what one looks like:

A list fragment comes complete with its own list view so you don't need to add it yourself. You just need to provide the list fragment with data.

ListFragment is a subclass of Fragment.

There are a couple of major advantages in using a list fragment to display categories of data:

- **You don't need to create your own layout.** List fragments define their own layout programmatically, so there's no XML layout for you to create or maintain. The layout the list fragment generates includes a single list view. You access this list view in your fragment code using the list fragment's `getListView()` method. You need this in order to specify what data should be displayed in the list view.

- **You don't have to implement your own event listener.** The `ListFragment` class automatically implements an event listener that listens for when items in the list view are clicked. Instead of creating your own event listener and binding it to the list view, you just need to implement the list fragment's `onListItemClick()` method. This makes it easier to get your fragment to respond when the user clicks on items in the list view. You'll see this in action later on.

So what does the list fragment code look like?

A list fragment is a type of fragment that specializes in working with a list view. It has a default layout that contains the list view.

How to create a list fragment

You add a list fragment to your project in the same way you add a normal fragment. Highlight the *com.hfad.workout* package in the *app/src/main/java* folder, then go to File→New...→Fragment→Fragment (Blank). Name the fragment "WorkoutListFragment", and then uncheck the options to create layout XML, and also the options to include fragment factory methods and interface callbacks (list fragments define their own layouts programmatically, so you don't need Android Studio to create one for you). When you click on the Finish button, Android Studio creates a new list fragment in a file called *WorkoutListFragment.java* in the *app/src/main/java* folder.

Uncheck these options, as we don't need them. If prompted for the fragment's source language, select the option for Java.

Here's what the basic code looks like to create a list fragment. As you can see, it's very similar to that of a normal fragment. Replace the code in *WorkoutListFragment.java* with the code below:

```
package com.hfad.workout;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutListFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

The activity needs to extend ListFragment, not Fragment.

Calling the superclass onCreateView() method gives you the default layout for the ListFragment.

The above code creates a basic list fragment called `WorkoutListFragment`. As it's a list fragment, it needs to extend the `ListFragment` class rather than `Fragment`.

The `onCreateView()` method is optional. It gets called when the fragment's view gets created. We're including it in our code as we want to populate the fragment's list view with data as soon as it gets created. If you don't need your code to do anything at this point, you don't need to include the `onCreateView()` method.

The next thing we need to do is add data to the list view in the `onCreateView()` method.

there are no Dumb Questions

Q: When we create a list fragment, why do we choose the option for Fragment (Blank) instead of Fragment (List)?

A: The Fragment (List) option produces code that's more complex, most of which we don't need to use. The code generated by the Fragment (Blank) is simpler.

Adapters revisited

As we said in Chapter 7, you can connect data to a list view using an adapter. The adapter acts as a bridge between the data and the list view. This is still the case when your list view is in a fragment, or a list fragment:

An adapter acts as a bridge between a view and a data source. An array adapter is a type of adapter that specializes in working with arrays.

We want to supply the list view in `WorkoutListFragment` with an array of workout names, so we'll use an array adapter to bind the array to the list view as before. As you may recall, an array adapter is a type of adapter that's used to bind arrays to views. You can use it with any subclass of the `AdapterView` class, which means you can use it with both list views and spinners.

In our case, we're going to use an array adapter to display an array of data from the `Workout` class in the list view.

We'll see how this works on the next page.

Our previous array adapter

As we said in Chapter 7, you use an array adapter by initializing it and attaching it to the list view.

To initialize the array adapter, you first specify what type of data is contained in the array you want to bind to the list view. You then pass it three parameters: a `Context` (usually the current activity), a layout resource that specifies how to display each item in the array, and the array itself.

Here's the code we used in Chapter 7 to create an array adapter to displays `Drink` data from the `Drink.drinks` array:

```
ArrayAdapter<Drink> listAdapter = new ArrayAdapter<>(
    this,
    android.R.layout.simple_list_item_1,
    Drink.drinks);
```

The current context. In our Chapter 7 scenario, it was the current activity.

This is a built-in layout resource. It tells the array adapter to display each item in the array in a single text view.

The array

There's a big difference between the situation we had back in Chapter 7 and the situation we have now. Back in Chapter 7, we used the array adapter to display data in an activity. But this time, we want to display data in a fragment. What difference does this make?

A fragment isn't a subclass of Context

As you saw earlier in the book, the `Activity` class is a subclass of the `Context` class. This means that all of the activities you create have access to global information about the app's environment.

But the `Fragment` class *isn't* a subclass of the `Context` class. It has no access to global information, and you can't use `this` to pass the current context to the array adapter. Instead, you need to get the current context in some other way.

One way is to use another object's `getContext()` method to get a reference to the current context. If you create the adapter in the fragment's `onCreateView()` method, you can use the `getContext()` method of the `onCreateView()` `LayoutInflator` parameter to get the context instead.

Once you've created the adapter, you bind it to the `ListView` using the fragment's `setListAdapter()` method:

```
setListAdapter(listAdapter);
```

We'll show you the full code on the next page.

The updated WorkoutListFragment code

We've updated our *WorkoutListFragment.java* code so that it populates the list view with the names of the workouts. Apply these changes to your code, then save your changes:

```
package com.hfad.workout;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class WorkoutListFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        String[] names = new String[Workout.workouts.length];
        for (int i = 0; i < names.length; i++) {
            names[i] = Workout.workouts[i].getName();
        }

        ArrayAdapter<String> adapter = new ArrayAdapter<>(
            inflater.getContext(), android.R.layout.simple_list_item_1,
            names);
        setListAdapter(adapter);

        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

We're using this class in the onCreateView() method.

Create a String array of the workout names.

Create an array adapter.

Get the context from the layout inflater.

Bind the array adapter to the list view.

Now that the `WorkoutListFragment` contains a list of workouts, let's add it to `MainActivity`.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Display WorkoutListFragment in the MainActivity layout

We're going to add our new `WorkoutListFragment` to `MainActivity`'s *layout activity_main.xml*. The layout currently displays a button that we're using to navigate from `MainActivity` to `DetailActivity`:

MainActivity's layout currently displays a button.

We want to remove the button, and display `WorkoutListFragment` in its place. Here's what the new version of the layout will look like:

We're going to change the layout so that it displays WorkoutListFragment instead of the button.

What will the code be like? Have a go at the exercise on the next page.

Layout Magnets

Somebody put a new version of *activity_main.xml* on our fridge door. Unfortunately some of the magnets fell off when we shut the door too hard. Can you piece the layout back together again? (You won't need to use all of the magnets below.)

The layout needs to display `WorkoutListFragment`.

```
<?xml version="1.0" encoding="utf-8"?>

<                                  xmlns:android="http://schemas.android.com/apk/res/android"

                                   =""

    android:layout_width="match_parent"

    android:layout_height="match_parent"/>
```

`fragment`

`android:fragment`

`Fragment`

`com.hfad.workout.`

`LinearLayout`

`WorkoutListFragment`

`android:name`

Layout Magnets Solution

Somebody put a new version of *activity_main.xml* on our fridge door. Unfortunately some of the magnets fell off when we shut the door too hard. Can you piece the layout back together again? (You won't need to use all of the magnets below.)

The layout needs to display `WorkoutListFragment`.

```
<?xml version="1.0" encoding="utf-8"?>
<  fragment  xmlns:android="http://schemas.android.com/apk/res/android"
    android:name  ="  com.hfad.workout.  WorkoutListFragment  "
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

You declare a fragment with the <fragment> element.

You need to give the full name of the fragment.

`android:fragment` `Fragment` `LinearLayout`

You didn't need to use these magnets.

The code for activity_main.xml

As we want `MainActivity`'s layout to only contain a single fragment, we can replace nearly all of the code we currently have.

Here's the updated code for *activity_main.xml*. As you can see, it's much shorter than the original version. Update your version of the code to reflect our changes.

Our layout only contains a single fragment, so we can get rid of the LinearLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.workout.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onShowDetails"
        android:text="@string/details_button" />
</LinearLayout>
```

We no longer need this button.

Here's the fragment.

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutListFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

We'll go through what happens when this code runs over the next couple of pages.

What happens when the code runs

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Here's a runthrough of what happens when we run the app.

1. **When the app is launched, MainActivity gets created.**
`MainActivity`'s `onCreate()` method runs. This specifies that *activity_main.xml* should be used for `MainActivity`'s layout. *activity_main.xml* includes a `<fragment>` element that refers to `WorkoutListFragment`.

2. **WorkoutListFragment is a ListFragment, so it uses a ListView as its layout.**

3. **WorkoutListFragment creates an ArrayAdapter<String>, an array adapter that deals with arrays of String objects.**

4. **The ArrayAdapter<String> retrieves data from the names array.**

The story continues

5 **WorkoutListFragment attaches the array adapter to the ListView using the setListAdapter() method.**
The list view uses the array adapter to display a list of the workout names.

Test drive the app

When we run the app, `MainActivity` gets launched.

`MainActivity`'s layout contains the fragment `WorkoutListFragment`. The fragment contains a list of the workout names, and this is displayed in the activity.

Here's a list of all the workout titles from the Workout class.

That looks great, but when we click on one of the workouts, nothing happens. We need to update the code so that when we click on one of the workouts, details of that workout are displayed.

Connect the list to the detail

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

There are a few ways that we can start `DetailActivity` and display the details of the workout that was clicked. We'll use this technique:

1. Add code to `WorkoutListFragment` that waits for a workout to be clicked.
2. When that code runs, call some code in *MainActivity.java* that will start `DetailActivity`, passing it the ID of the workout.
3. Get `DetailActivity` to pass the ID to `WorkoutDetailFragment` so that the fragment can display details of the correct workout.

We don't want to write code in `WorkoutListFragment` that talks *directly* to `MainActivity`. Can you think why?

The answer is *reuse*. We want our fragments to know as little as possible about the environment that contains them so that we can reuse them elsewhere. The more a fragment needs to know about the activity using it, the less reusable it is.

We need to use an *interface* to decouple the fragment from the activity.

We have two objects that need to talk to each other—the fragment and the activity—and we want them to talk without one side knowing too much about the other. The way we do that in Java is with an *interface*. When we define an interface, we're saying *what the minimum requirements are for one object to talk usefully to another*. That means that we'll be able to get the fragment to talk to any activity, so long as that activity implements the interface.

We need to decouple the fragment by using an interface

We're going to create an interface called **`Listener`**. If `MainActivity` implements the interface, `WorkoutListFragment` will be able to tell `MainActivity` when one of its items has been clicked. To do this, we'll need to make changes to `WorkoutListFragment` and `MainActivity`.

What WorkoutListFragment needs to do

We'll start with the code for `WorkoutListFragment`. There are a few changes we need to make, in this order.

1. **Define the interface.**
   We'll define the listener's interface in `WorkoutListFragment`. We're defining the interface here, as its purpose is to allow `WorkoutListFragment` to communicate with any activity.
2. **Register the listener (in this case MainActivity) when WorkoutListFragment gets attached to it.**
   This will give `WorkoutListFragment` a reference to `MainActivity`.
3. **Tell the listener when an item gets clicked.**
   `MainActivity` will then be able to respond to the click.

You need to go through similar steps to these whenever you have a fragment that needs to communicate with the activity it's attached to.

We'll go through each change individually, then show you the full code.

1. Define the listener interface

We want any activities that implement the listener interface to respond to item clicks, so we'll define a method for the interface, `itemClicked()`. The `itemClicked()` method has one parameter, the ID of the item that's clicked.

Here's the interface:

We'll call the interface Listener.

```
interface Listener {
    void itemClicked(long id);
};
```

Any activities that implement the Listener interface must include this method. We'll use it to get the activity to respond to items in the fragment being clicked.

Next we'll look at how you register the listener on the next page.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

2. Register the listener

We need to save a reference to the activity `WorkoutListFragment` gets attached to. This activity will implement the `Listener` interface, so we'll add the following private variable to `WorkoutListFragment`:

```
private Listener listener;
```

We need to set this variable when `WorkoutListFragment` gets attached to an activity. If you look back at the fragment lifecycle, when a fragment gets attached to an activity, the fragment's `onAttach()` method is called. We'll use this method to set the value of the listener:

```
public void onAttach(Context context) {
    super.onAttach(context);
    this.listener = (Listener)context;
}
```

This is the context (in this case, the activity) the fragment is attached to.

3. Respond to clicks

When an item in `WorkoutListFragment` gets clicked, we want to call the listener's `itemClicked()` method. This is the method we defined in the interface on the previous page. But how can we tell when an item's been clicked?

Whenever an item gets clicked in a list fragment, the list fragment's `onListItemClick()` method gets called. Here's what it looks like:

```
public void onListItemClick(ListView listView,
                            View itemView,
                            int position,
                            long id) {
    //Do something
}
```

The list view

The item in the list view that was clicked, its position, and its ID

The `onListItemClick()` method has four parameters: the list view, the item in the list that was clicked, its position, and the row ID of the underlying data. This means we can use the method to pass the listener the ID of the workout the user clicked on:

```
public void onListItemClick(ListView listView, View itemView, int position, long id) {
    if (listener != null) {
        listener.itemClicked(id);
    }
}
```

Call the itemClicked() method in the activity, passing it the ID of the workout the user selected.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

The code for WorkoutListFragment.java

Here's the full code for *WorkoutListFragment.java* code (apply these changes to your code, then save your work):

```
package com.hfad.workout;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.content.Context;
import android.widget.ListView;

public class WorkoutListFragment extends ListFragment {

    static interface Listener {
        void itemClicked(long id);
    };

    private Listener listener;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        String[] names = new String[Workout.workouts.length];
        for (int i = 0; i < names.length; i++) {
            names[i] = Workout.workouts[i].getName();
        }
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
                inflater.getContext(), android.R.layout.simple_list_item_1,
                names);
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        this.listener = (Listener)context;
    }

    @Override
    public void onListItemClick(ListView listView, View itemView, int position, long id) {
        if (listener != null) {
            listener.itemClicked(id);
        }
    }
}
```

Import these classes.

Add the listener to the fragment.

This is called when the fragment gets attached to the activity. Remember, the Activity class is a subclass of Context.

Tell the listener when an item in the ListView is clicked.

MainActivity needs to implement the interface

Next we need to make `MainActivity` implement the `Listener` interface we just created. The interface specifies an `itemClicked()` method, so we'll make the method start `DetailActivity`, passing it the ID of the workout the user selected.

Here's the full code for *MainActivity.java*. Update your code so that it matches ours.

```
package com.hfad.workout;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.content.Intent;

public class MainActivity extends AppCompatActivity
                          implements WorkoutListFragment.Listener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onShowDetails(View view) {
        Intent intent = new Intent(this, DetailActivity.class);
        startActivity(intent);
    }

    @Override
    public void itemClicked(long id) {
        Intent intent = new Intent(this, DetailActivity.class);
        intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int)id);
        startActivity(intent);
    }
}
```

Implement the listener interface defined in WorkoutListFragment.

This is the method called by MainActivity's button. We've removed the button, so we no longer need this method.

This method is defined by the interface, so we need to implement it.

Pass the ID of the workout to DetailActivity. EXTRA_WORKOUT_ID is the name of a constant we'll define in DetailActivity.

Those are all the changes we need to make to `MainActivity`. There's just one more code change we need to make to our app.

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

DetailActivity needs to pass the ID to WorkoutDetailFragment

So far, `WorkoutListFragment` passes the ID of the workout that was clicked to `MainActivity`, and `MainActivity` passes it to `DetailActivity`. We need to make one more change, which is to pass the ID from `DetailActivity` to `WorkoutDetailActivity`.

Here's the updated code for `DetailActivity` that does this. Update your version of *DetailActivity.java* to reflect our changes:

```
package com.hfad.workout;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class DetailActivity extends AppCompatActivity {

    public static final String EXTRA_WORKOUT_ID = "id";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        WorkoutDetailFragment frag = (WorkoutDetailFragment)
                getSupportFragmentManager().findFragmentById(R.id.detail_frag);
        frag.setWorkout(1);
        int workoutId = (int) getIntent().getExtras().get(EXTRA_WORKOUT_ID);
        frag.setWorkout(workoutId);
    }
}
```

We're using a constant to pass the ID from MainActivity to DetailActivity to avoid hardcoding this value.

We're no longer hardcoding an ID of 1, so remove this line.

Get the ID from the intent, and pass it to the fragment via its setWorkout() method.

Over the next couple of pages we'll examine what happens when the code runs.

What happens when the code runs

WorkoutDetailFragment
WorkoutListFragment
Coordinate fragments

Here's a runthrough of what happens when we run the app.

1. **When the app is launched, MainActivity gets created.**
`WorkoutListFragment` is attached to `MainActivity`, and `WorkoutListFragment`'s `onAttach()` method runs.

2. **WorkoutListFragment registers MainActivity as a Listener.**

3. **When an item is clicked in WorkoutListFragment, the fragment's onListItemClick() method is called.**
This calls `MainActivity`'s `itemClicked()` method, passing it the ID of the workout that was clicked, in this example `1`.

4. **MainActivity's itemClicked() method starts DetailActivity, passing it the value of the workout ID in an intent.**

The story continues...

- ☑ WorkoutDetailFragment
- ☑ WorkoutListFragment
- ☐ **Coordinate fragments**

5 **DetailActivity calls WorkoutDetailFragment's setWorkout() method, passing it the value of the workout ID.**
`WorkoutDetailFragment` uses the workout ID, in this case `1`, to display the workout title and description in its views.

Test drive the app

- ☑ WorkoutDetailFragment
- ☑ WorkoutListFragment
- ☑ **Coordinate fragments**

When we run the app, `MainActivity` gets launched. It displays a list of workouts in its fragment, `WorkoutListFragment`.

When you click on one of the workouts, `DetailActivity` is displayed. It shows details of the workout that we selected.

That's everything we need to do to use the fragments we've created in a user interface for a phone. In the next chapter, you'll see how to reuse the fragments, and create a different user interface that will work better for tablets.

Your Android Toolbox

You've got Chapter 9 under your belt and now you've added fragments to your toolbox.

BULLET POINTS

- A fragment is used to control part of a screen. It can be reused across multiple activities.
- A fragment has an associated layout.
- The `onCreateView()` method gets called each time Android needs the fragment's layout.
- Add a fragment to an activity's layout using the `<fragment>` element and adding a `name` attribute.
- The fragment lifecycle methods tie in with the states of the activity that contains the fragment.
- The `Fragment` class doesn't extend the `Activity` class or implement the `Context` class.
- Fragments don't have a `findViewById()` method. Instead, use the `getView()` method to get a reference to the root view, then call the view's `findViewById()` method.
- A list fragment is a fragment that comes complete with a `ListView`. You create one by subclassing `ListFragment`.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

Fragment lifecycle methods

onAttach()
↓
onCreate()
↓
onCreateView()
↓
onActivityCreated()
↓
onStart()
↓
onResume()
↓
onPause()
↓
onStop()
↓
onDestroyView()
↓
onDestroy()
↓
onDetach()

10 fragments for larger interfaces

Different Size, Different Interface

They're using a tablet? Maybe we should rethink the UI...

So far we've only run our apps on devices with a small screen.

But what if your users have tablets? In this chapter you'll see how to create **flexible user interfaces** by making your app **look and behave differently** depending on the device it's running on. We'll show you how to control the behavior of your app when you press the Back button by introducing you to the **back stack** and **fragment transactions**. Finally, you'll find out how to **save and restore the state** of your fragment.

The Workout app looks the same on a phone and a tablet

In the previous chapter, we created a version of the Workout app designed to work on a phone.

As a reminder, when the app launches, it displays `MainActivity`. This contains a fragment, `WorkoutListFragment`, that displays a list of workouts. When the user clicks on one of the workouts, `DetailActivity` starts, and displays details of the workout in its fragment, `WorkoutDetailFragment`.

Click on an item in a list, and it launches a second activity.

When we run the app on a tablet, the app works in exactly the same way. As the screen size is larger, however, there's lots of empty space in the user interface that we could make better use of.

Look at all this empty space. Maybe we could use it for something.

Designing for larger interfaces

One way in which we could better use the empty space is to display details of the workout to the right of the list of workouts. When the user clicks on one of the workouts, details of that workout could be displayed on the same screen without us having to start a second activity:

We don't want to change our app completely though. We still want our app to work as it does currently if it's running on a phone.

We're going to get our app to adapt to the type of device it's running on. If the app's running on a phone, we'll display details of the workout in a separate activity (this is the app's current behavior). If the app's running on a tablet, we'll display details of the workout next to the list of workouts.

Before we get started, let's remind ourselves how the app's currently structured.

The phone version of the app

The phone version of the app we built in Chapter 9 works in the following way:

1. **When the app gets launched, it starts MainActivity.**
   `MainActivity` uses *activity_main.xml* for its layout, and contains a fragment called `WorkoutListFragment`.

2. **WorkoutListFragment displays a list of workouts.**

3. **When the user clicks on one of the workouts, DetailActivity starts.**
   `DetailActivity` uses *activity_detail.xml* for its layout, and contains a fragment called `WorkoutDetailFragment`.

4. **WorkoutDetailFragment uses fragment_workout_detail.xml for its layout.**
   It displays the details of the workout the user has selected.

5. **WorkoutListFragment and WorkoutDetailFragment get their workout data from *Workout.java*.**
   Workout.java contains an array of `Workout`s.

So how does it need to work differently on a tablet?

The tablet version of the app

Here's how the app will work when it runs on a tablet:

1. **When the app gets launched, it starts MainActivity as before.**
   `MainActivity` uses *activity_main.xml* for its layout.

2. **MainActivity's layout displays two fragments, WorkoutListFragment and WorkoutDetailFragment.**

3. **WorkoutListFragment displays a list of workouts.**
   It's a list fragment, so it has no extra layout file.

4. **When the user clicks on one of the workouts, its details are displayed in WorkoutDetailFragment.**
   `WorkoutDetailFragment` uses *fragment_workout_detail.xml* for its layout.

5. **Both fragments get their workout data from *Workout.java* as before.**

There are two key differences.

The first is that `MainActivity`'s layout needs to display both fragments, not just `WorkoutListFragment`.

The second difference is that we no longer need to start `DetailActivity` when the user clicks on one of the workouts. Instead, we need to display `WorkoutDetailFragment` in `MainActivity`.

We'll go through the steps for how to change the app on the next page.

Here's what we're going to do

There are a number of steps we'll go through to change the app:

1. **Create a tablet AVD (Android Virtual Device).**
We're going to create a new UI for a tablet, so we'll create a new tablet AVD to run it on. This will allow us to check how the app looks and behaves on a device with a larger screen.

2. **Create a new tablet layout.**
We'll reuse the fragments we've already created in a new layout that's designed to work on devices with larger screens. We'll display details of the first workout in the first instance so that we can see the fragments side by side.

3. **Display details of the workout the user selects.**
We'll update the app so that when the user clicks on one of the workouts, we'll display the details of the workout the user selected.

Do this!

We're going to update the Workout app in this chapter, so open your original Workout project from Chapter 9 in Android Studio.

Create a tablet AVD

Before we get into changing the app, we're going to create a new Nexus 7 AVD running API level 25 so that you can see how the app looks and behaves when it's running on a tablet. The steps are nearly the same as when you created a Nexus 5X AVD back in Chapter 1.

Open the Android Virtual Device Manager

You create AVDs using the AVD Manager. Open the AVD Manager by selecting Android on the Tools menu and choosing AVD Manager.

You'll be presented with a screen showing you a list of the AVDs you've already set up. Click on the Create Virtual Device button at the bottom of the screen.

Click on the Create Virtual Device button to create an AVD.

Select the hardware

On the next screen, you'll be prompted to choose a device definition, the type of device your AVD will emulate.

We're going to see what our app looks like running on a Nexus 7 tablet. Choose Tablet from the Category menu and Nexus 7 from the list. Then click the Next button.

Choose the Tablet and Nexus 7 options.

Creating a tablet AVD (continued)

Create AVD
Create layout
Show workout

Select a system image

Next, you need to select a system image. The system image gives you an installed version of the Android operating system. You can choose the version of Android you want to be on your AVD.

You need to choose a system image for an API level that's compatible with the app you're building. As an example, if you want your app to work on a minimum of API level 19, choose a system image for *at least* API level 19. As in Chapter 1, we want our AVD to run API level 25, so choose the system image with a release name of Nougat and a target of Android 7.1.1, the version number of API level 25. Then click on the Next button.

We'll choose the same system image as we did in Chapter 1.

Creating a tablet AVD (continued)

Verify the AVD configuration

On the next screen, you'll be asked to verify the AVD configuration. This screen summarizes the options you chose over the last few screens, and gives you the option of changing them. Change the screen startup orientation to Landscape, then click on the Finish button.

The AVD Manager will create the Nexus 7 AVD for you, and when it's done, display it in its list of devices. You may now close the AVD Manager.

Now that we've created our tablet AVD, we can get to work on updating the Workout app. We want to change the app so that `MainActivity` uses one layout when it's running on a phone, and another layout when it's running on a tablet. But how can we do this?

Put screen-specific resources in screen-specific folders

Earlier in the book, you saw how you could get different devices to use image resources appropriate to their screen size by putting different-sized images in the different *drawable** folders. As an example, you put images intended for devices with high-density screens in the *drawable-hdpi* folder.

You can do something similar with other resources such as layouts, menus, and values. If you want to create multiple versions of the same resource for different screen specs, you need to create multiple resource folders with an appropriate name, then add the resource to that folder. The device will then load the resource at runtime from the folder that's the closest match to its screen spec.

If you want to have one layout for large screen devices such as tablets, and another layout for smaller devices such as phones, you put the layout for the tablet in the *app/src/main/res/layout-large* folder, and the layout for the phone in the *app/src/main/res/layout* folder. When the app runs on a phone, it will use the layout in the *layout* folder. If it's run on a tablet, it will use the layout in the *layout-large* folder instead.

Android uses the names of your resource folders to decide which resources it should use at runtime.

Layouts in the layout folder can be used by any device, but layouts in the layout-large folder will only be used by devices with a large screen.

On the next page, we'll show you all the different options you can use for your resource folder names.

The different folder options

You can put all kinds of resources (drawables or images, layouts, menus, and values) in different folders to specify which types of device they should be used with. The screen-specific folder name can include screen size, density, orientation and aspect ratio, with each part separated by hyphens. As an example, if you want to create a layout that will only be used by very large tablets in landscape mode, you would create a folder called *layout-xlarge-land* and put the layout file in that folder. Here are the different options you can use for the folder names:

You must specify a resource type.

Screen density is based on dots per inch.

Resource type	Screen size	Screen density	Orientation	Aspect ratio
drawable	-small	-ldpi	-land	-long
layout	-normal	-mdpi	-port	-notlong
menu	-large	-hdpi		
mipmap	-xlarge	-xhdpi		
values		-xxhdpi		
		-xxxhdpi		
		-nodpi		
		-tvdpi		

A mipmap resource is used for application icons. Older versions of Android Studio use drawables instead.

This is for density-independent resources. Use -nodpi for any image resources you don't want to scale (e.g., a folder called drawable-nodpi).

long is for screens that have a very high value for height.

Android decides at runtime which resources to use by checking the spec of the device and looking for the best match. If there's no exact match, it will use resources designed for a smaller screen than the current one. If resources are only available for screens *larger* than the current one, Android won't use them and the app will crash.

If you only want your app to work on devices with particular screen sizes, you can specify this in *AndroidManifest.xml* using the `<supports-screens>` attribute. As an example, if you don't want your app to run on devices with small screens, you'd use:

```
<supports-screens android:smallScreens="false"/>
```

Using the different folder names above, you can create layouts that are tailored for phones and tablets.

For more information on the settings on this page, see:
https://developer.android.com/guide/practices/screens_support.html.

BE the Folder Structure

Below you'll see the code for an activity. You want to display one layout when it runs on devices with large-sized screens, and another layout when it runs on devices with smaller-sized screens. Which of these folder structures will allow you to do that?

Here's the activity.

```
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    ...
    }
}
```

C
app/src/main
res
layout
activity_main.xml
layout-large
activity_main.xml

D
app/src/main
res
layout
activity_main.xml
activity_main_tablet.xml

E
app/src/main
res
layout-large
activity_main.xml
layout-normal
activity_main.xml

F
app/src/main
res
layout
activity_main.xml
layout-large-land
activity_main.xml
layout-large-port
activity_main.xml

BE the Folder Structure Solution

Below you'll see the code for an activity. You want to display one layout when it runs on devices with large-sized screens, and another layout when it runs on devices with smaller-sized screens. Which of these folder structures will allow you to do that?

```
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    ...
    }
}
```

A

Android doesn't recognize the folder name layout-tablet. activity_main.xml in the layout folder will be displayed on all devices.

B

A device with a large screen will use the layout in the layout-large-land folder when it's oriented landscape. But when the device is rotated to portrait, it will use the layout in the layout folder—and that's not what you want.

C

Devices with a large screen will use the layout in the layout-large folder. Devices with smaller screens will use the layout in the layout folder.

D

The activity uses a layout file called activity_main.xml. The layout activity_main_tablet.xml won't be used because it has the wrong filename.

E

Devices with a large screen will use the layout in the layout-large folder. Devices with normal screens will use the layout in the layout-normal folder. There's no layout for devices with a small screen, so they won't be able to run the app.

F

Devices with a large screen will use the layout in the layout-large-land folder when the device is turned landscape, and the layout in the layout-large-port folder when the device is turned portrait. Other devices will use the layout in the layout folder.

Tablets use layouts in the layout-large folder

Create AVD
Create layout
Show workout

To get the tablet version of our app up and running, we need to copy our existing activity layout file *activity_main.xml* into the *app/src/main/res/layout-large* folder and then update that version of the file. This layout will then only be used by devices with a large screen.

If the *app/src/main/res/layout-large* folder doesn't exist in your Android Studio project, you'll need to create it. To do this, switch to the Project view of Android Studio's explorer, highlight the *app/src/main/res* folder, and choose File→New...→Directory. When prompted, give the folder a name of "layout-large". When you click on the OK button, Android Studio will create the new *app/src/main/res/layout-large* folder.

To copy the *activity_main.xml* layout file, highlight the file in the explorer, and choose the Copy command from the Edit menu. Then highlight the new *layout-large* folder, and choose the Paste command from the Edit menu. Android Studio will copy the *activity_main.xml* file into the *app/src/main/res/layout-large* folder.

Here's the folder Android Studio created.

If you open the file you just pasted, it should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.WorkoutListFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

We've not changed the layout, just copied it to the layout-large folder.

This is exactly the same layout that we had before. It contains a single fragment, `WorkoutListFragment`, that displays a list of workouts. The next thing we need to do is update the layout so that it displays two fragments side by side, `WorkoutListFragment` and `WorkoutDetailFragment`.

Create AVD
Create layout
Show workout

The layout-large version of the layout needs to display two fragments

We're going to change the version of *activity_main.xml* in the *layout-large* folder so that it contains the two fragments. To do this, we'll add the fragments to a linear layout with the orientation set to `horizontal`. We'll adjust the width of the fragments so that `WorkoutListFragment` takes up two-fifths of the available space, and `WorkoutDetailFragment` takes up three-fifths.

Our version of *activity_main.xml* is below. Update your code to reflect our changes. Make sure that you only edit the tablet version of the layout that's in the *layout-large* folder.

We're putting the fragments in a LinearLayout with a horizontal orientation so the two fragments will be displayed alongside each other.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:name="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>

    <fragment
        android:name="com.hfad.workout.WorkoutDetailFragment"
        android:id="@+id/detail_frag"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent"/>
</LinearLayout>
```

Our layout already includes WorkoutListFragment.

The fragments need IDs so that Android doesn't lose track of where to put each fragment.

We're adding WorkoutDetailFragment to MainActivity's layout.

We'll run through what happens when the code runs on the next page.

What the updated code does

Create AVD
Create layout
Show workout

Before we take the app for a test drive, let's go through what happens when the code runs.

1. **When the app is launched, MainActivity gets created.**
`MainActivity`'s `onCreate()` method runs. This specifies that *activity_main.xml* should be used for `MainActivity`'s layout.

2a. **If the app's running on a tablet, it uses the version of activity_main.xml that's in the layout-large folder.**
The layout displays `WorkoutListFragment` and `WorkoutDetailFragment` side by side.

2b. **If the app's running on a device with a smaller screen, it uses the version of activity_main.xml that's in the layout folder.**
The layout displays `WorkoutListFragment` on its own.

Test drive the app

When you run the app on a phone, the app looks just as it did before. `MainActivity` displays a list of workout names, and when you click on one of the workouts, `DetailActivity` starts and displays its details.

When you run the app on a tablet, `MainActivity` displays a list of workout names on the left, and details of the first workout appear next to it.

When you click on one of the workouts, `DetailActivity` still gets displayed. We need to change our code so that if the app's running on a tablet, `DetailActivity` no longer starts. Instead, we need to display details of the workout the user selects in `MainActivity`, and not just the first workout.

We need to change the itemClicked() code

We need to change the code that decides what to do when items in `WorkoutListFragment` are clicked. This means that we need to change the `itemClicked()` method in `MainActivity`. Here's the current code:

```
...

 public class MainActivity extends AppCompatActivity
                          implements WorkoutListFragment.Listener {
...
     @Override
     public void itemClicked(long id) {
         Intent intent = new Intent(this, DetailActivity.class);
         intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int)id);
         startActivity(intent);
     }
 }
```

This is the itemClicked() method we wrote in the previous chapter. It starts DetailActivity, and passes it the ID of the workout that was clicked.

The current code starts `DetailActivity` whenever the user clicks on one of the workouts. We need to change the code so that this only happens if the app's running on a device with a small screen such as a phone. If the app's running on a device with a large screen, when the user picks a workout we need to display the details of the workout shown to the right of the list of workouts in `WorkoutDetailFragment`.

But how do we update the workout details?

The `WorkoutDetailFragment` updates its views when it is started. But once the fragment is displayed onscreen, how do we get the fragment to update the details?

You might be thinking that we could play with the fragment's lifecycle so that we get it to update. Instead, **we'll replace the detail fragment with a *brand-new* detail fragment, each time we want its text to change**.

There's a really good reason why...

You want fragments to work with the Back button

Suppose you have a user that runs the app on a phone. When they click on a workout, details of that workout are displayed in a separate activity. If the user clicks on the Back button, they're returned to the list of workouts:

Then suppose the user runs the app on a tablet, and clicks on one workout, followed by a second workout. If they click on the Back button, they're probably going to expect to be returned to the first workout they chose:

In every app we've built so far, when we've clicked on the Back button, we've been returned to the previous activity. This is standard Android behavior, and something that Android has handled for us automatically. If we're running this particular app on a tablet, however, we don't want the Back button to return us to the previous *activity*. We want it to return us to the previous *fragment state*.

Welcome to the back stack

When you go from activity to activity in your app, Android keeps track of each activity you've visited by adding it to the **back stack**. The back stack is a log of the places you've visited on the device, each place recorded as a separate transaction.

A back stack scenario

1. Suppose you start by visiting a fictitious activity in your app, `Activity1`. Android records your visit to `Activity1` on the back stack as a transaction.

2. You then go to `Activity2`. Your visit to `Activity2` is added to the top of the back stack as a separate transaction.

3. You then go to `Activity3`. `Activity3` is added to the top of the back stack.

4. When you click on the Back button, `Activity3` pops off the top of the back stack. Android displays `Activity2`, as this activity is now at the top of the back stack.

5. If you click on the Back button again, `Activity2` pops off the top of the back stack, and `Activity1` is displayed.

Back stack transactions don't have to be activities

We've shown you how the back stack works with activities, but the truth is, it doesn't just apply to activities. It applies to any sort of transaction, including changes to fragments.

These are two different fragment transactions for WorkoutDetailFragment. The top one displays details of the Core Agony workout, and the bottom one displays details of the Wimp Special.

This means that *fragment* changes can be reversed when you click on the Back button, just like *activity* changes can.

When you click on the Back button, the transaction that contains details of the Core Agony is popped off the top of the back stack. Details of the Wimp Special are displayed.

So how can we record changes to fragments as separate transactions on the back stack?

Don't update—instead, replace

We're going to replace the entire `WorkoutDetailFragment` with a new instance of it each time the user selects a different workout. Each new instance of `WorkoutDetailFragment` will be set up to display details of the workout the user selects. That way, we can add each fragment replacement to the back stack as a separate transaction. Each time the user clicks on the Back button, the most recent transaction will be popped off the top of the stack, and the user will see details of the previous workout they selected.

To do this, we first need to know how to replace one fragment with another. We'll look at this on the next page.

Android builds the back stack as you navigate from one activity to another. Each activity is recorded in a separate transaction.

Use a frame layout to replace fragments programmatically

Add a fragment using a <FrameLayout> whenever you need to replace fragments programmatically, such as when you need to add fragment changes to the back stack.

To replace one fragment with another in `MainActivity`'s tablet user interface, we need to begin by making a change to the *activity_main.xml* layout file in the *layout-large* folder. Instead of inserting `WorkoutDetailFragment` directly using the `<fragment>` element, we'll use a frame layout.

We covered frame layouts in Chapter 5.

We'll add the fragment to the frame layout programmatically. Whenever an item in the `WorkoutListFragment` list view gets clicked, we'll replace the contents of the frame layout with a new instance of `WorkoutDetailFragment` that displays details of the correct workout.

Here's our new version of the code for *activity_main.xml* in the *layout-large* folder. Update your code to include our changes.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:name="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>

    ~~<fragment~~
    <FrameLayout
        ~~android:name="com.hfad.workout.WorkoutDetailFragment"~~
        ~~android:id="@+id/detail_frag"~~
        android:id="@+id/fragment_container"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent"/>
</LinearLayout>
```

We're going to display the fragment inside a FrameLayout.

We'll add the fragment to the frame layout programmatically.

We'll give the FrameLayout an ID of fragment_container so we can refer to it in our activity code.

Create AVD
Create layout
Show workout

Use layout differences to tell which layout the device is using

We want `MainActivity` to perform different actions when the user clicks on a workout depending on whether the device is running on a phone or a tablet. We can tell which version of the layout's being used by checking whether or not the layout includes the frame layout we added on the previous page.

If the app is running on a tablet, the device will be using the version of *activity_main.xml* that's in the *layout-large* folder. This layout includes a frame layout with an ID of `fragment_container`. When the user clicks on a workout, we want to display a new instance of `WorkoutDetailFragment` in the frame layout.

If the app's running on a phone, the device will be using *activity_main.xml* in the *layout* folder. This layout doesn't include the frame layout. If the user clicks on a workout, we want `MainActivity` to start `DetailActivity` as it does currently.

If we can get our `MainActivity` code to check for the existence of a view with an ID of `fragment_container`, we can get `MainActivity` to behave differently depending on whether the app's running on a phone or a tablet.

The revised MainActivity code

We've updated `MainActivity` so that the `itemClicked()` method looks for a view with an ID of `fragment_container`. We can then perform different actions depending on whether or not the view is found.

Here's our full code for *MainActivity.java*; update your version of the code to match ours:

```
package com.hfad.workout;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.content.Intent;

public class MainActivity extends AppCompatActivity
                          implements WorkoutListFragment.Listener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void itemClicked(long id) {
        View fragmentContainer = findViewById(R.id.fragment_container);
        if (fragmentContainer != null) {
            //Add the fragment to the FrameLayout
        } else {
            Intent intent = new Intent(this, DetailActivity.class);
            intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int) id);
            startActivity(intent);
        }
    }
}
```

We've not changed this method.

Get a reference to the frame layout that will contain WorkoutDetailFragment. This will only exist if the app is being run on a device with a large screen.

We need to write code that will run if the frame layout exists.

If the frame layout doesn't exist, the app must be running on a device with a smaller screen. In that case, start DetailActivity and pass it the ID of the workout as before.

The next thing we need to do is see how we can add `WorkoutDetailFragment` to the frame layout programmatically.

Create AVD
Create layout
Show workout

Using fragment transactions

You can programmatically add a fragment to an activity's layout so long as the activity's running. All you need is a view group in which to place the fragment, such as a frame layout.

You add, replace, or remove fragments at runtime using a **fragment transaction**. A fragment transaction is a set of changes relating to the fragment that you want to apply, all at the same time.

When you create a fragment transaction, you need to do three things:

1. **Begin the transaction.**
   This tells Android that you're starting a series of changes that you want to record in the transaction.

2. **Specify the changes.**
   These are all the actions you want to group together in the transaction. This can include adding, replacing, or removing a fragment, updating its data, and adding it to the back stack.

3. **Commit the transaction.**
   This finishes the transaction and applies the changes.

1. Begin the transaction

You begin the transaction by first getting a reference to the activity's fragment manager. As you may remember from the previous chapter, the fragment manager is used to manage any fragments used by the activity. If you're using fragments from the Support Library as we are here, you get a reference to the fragment manager using the following method:

```
getSupportFragmentManager();
```

This returns the fragment manager that deals with fragments from the Support Library.

Once you have a reference to the fragment manager, you call its `beginTransaction()` method to begin the transaction:

```
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
```

The start of the fragment transaction

That's all you need to do to begin the transaction. On the next page we'll look at how you specify the changes you want to make.

2. Specify the changes

After beginning the transaction, you need to say what changes the transaction should include.

If you want to add a fragment to your activity's layout, you call the fragment transaction's `add()` method. This takes two parameters, the resource ID of the view group you want to add the fragment to, and the fragment you want to add. The code looks like this:

```
WorkoutDetailFragment fragment = new WorkoutDetailFragment();
transaction.add(R.id.fragment_container, fragment);
```

Create the fragment.

Add the fragment to the ViewGroup.

To replace the fragment, you use the `replace()` method:

```
transaction.replace(R.id.fragment_container, fragment);
```

Replace the fragment.

To remove the fragment completely, you use the `remove()` method:

```
transaction.remove(fragment);
```

Remove the fragment.

You can optionally use the `setTransition()` method to say what sort of transition animation you want for this transaction:

```
transaction.setTransition(transition);
```

You don't have to set a transition.

`transition` is the type of animation. Options for this are `TRANSIT_FRAGMENT_CLOSE` (a fragment is being removed from the stack), `TRANSIT_FRAGMENT_OPEN` (a fragment is being added), `TRANSIT_FRAGMENT_FADE` (the fragment should fade in and out), and `TRANSIT_NONE` (no animation). By default, there are no animations.

Once you've specified all the actions you want to take as part of the transaction, you can use the `addToBackStack()` method to add the transaction to the back stack. This method takes one parameter, a `String` name you can use to label the transaction. This parameter is needed if you need to programmatically retrieve the transaction. Most of the time you won't need to do this, so you can pass in a `null` value like this:

```
transaction.addToBackStack(null);
```

Most of the time you won't need to retrieve the transaction, so it can be set to null.

3. Commit the transaction

Finally, you need to commit the transaction. This finishes the transaction, and applies the changes you specified. You commit the transaction by calling the transaction's `commit()` method like this:

```
transaction.commit();
```

That's everything we need to know in order to create fragment transactions, so let's put it into practice by getting our `MainActivity` code to display an updated version of `WorkoutDetailFragment` every time the user clicks on a workout.

Activity Magnets

We want to write a new version of the `MainActivity`'s `itemClicked()` method. It needs to change the workout details that are displayed in `WorkoutDetailFragment` each time the user clicks on a new workout. See if you can finish the code below.

```
public void itemClicked(long id) {
    View fragmentContainer = findViewById(R.id.fragment_container);
    if (fragmentContainer != null) {
        WorkoutDetailFragment details = new WorkoutDetailFragment();

        FragmentTransaction ft = getSupportFragmentManager().________;
        details.setWorkout(id);

        ft.________(R.id.fragment_container, ________);

        ft.________(FragmentTransaction.TRANSIT_FRAGMENT_FADE);

        ft.________(null);

        ft.________;
    } else {
        Intent intent = new Intent(this, DetailActivity.class);
        intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int) id);
        startActivity(intent);
    }
}
```

You won't need to use all of the magnets.

`replace`

`commit()`

`beginTransaction()`

`setTransition`

`details`

`endTransaction()`

`addToBackStack`

`startTransaction()`

Activity Magnets Solution

We want to write a new version of the `MainActivity`'s `itemClicked()` method. It needs to change the workout details that are displayed in `WorkoutDetailFragment` each time the user clicks on a new workout. See if you can finish the code below.

```
public void itemClicked(long id) {
    View fragmentContainer = findViewById(R.id.fragment_container);
    if (fragmentContainer != null) {
        WorkoutDetailFragment details = new WorkoutDetailFragment();

        FragmentTransaction ft = getSupportFragmentManager().[beginTransaction()];
        details.setWorkout(id);

        ft.[replace](R.id.fragment_container, [details]);

        ft.[setTransition](FragmentTransaction.TRANSIT_FRAGMENT_FADE);

        ft.[addToBackStack](null);

        ft.[commit()];
    } else {
        Intent intent = new Intent(this, DetailActivity.class);
        intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int) id);
        startActivity(intent);
    }
}
```

This begins the transaction.

Each time the user clicks on a workout, we'll replace the fragment with a new instance of it.

This is a new instance of WorkoutDetailFragment. It displays details of the workout the user selected.

Set the fragment to fade in and out.

Add the transaction to the back stack.

Commit the transaction.

`endTransaction()` ← You didn't need to use these magnets. → `startTransaction()`

The updated MainActivity code

We're going to get a new instance of `WorkoutDetailFragment` (one that displays the correct workout), display the fragment in the activity, and then add the transaction to the back stack. Here's the full code. Update your version of *MainActivity.java* to reflect our changes:

```
package com.hfad.workout;

import android.support.v4.app.FragmentTransaction;
...

public class MainActivity extends AppCompatActivity
                          implements WorkoutListFragment.Listener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void itemClicked(long id) {
        View fragmentContainer = findViewById(R.id.fragment_container);
        if (fragmentContainer != null) {
            WorkoutDetailFragment details = new WorkoutDetailFragment();
            FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
            details.setWorkout(id);
            ft.replace(R.id.fragment_container, details);
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.addToBackStack(null);
            ft.commit();
        } else {
            Intent intent = new Intent(this, DetailActivity.class);
            intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int) id);
            startActivity(intent);
        }
    }
}
```

We're using a FragmentTransaction from the Support Library as we're using Support Library Fragments.

We haven't changed this method.

Start the fragment transaction.

Replace the fragment.

Add the transaction to the back stack.

Get the new and old fragments to fade in and out.

Commit the transaction.

On the next page we'll see what happens when the code runs.

What happens when the code runs

Create AVD
Create layout
Show workout

Here's a runthrough of what happens when we run the app.

1. **The app is launched on a tablet and MainActivity starts.**
`WorkoutListFragment` is attached to `MainActivity`, and `MainActivity` is registered as a listener on `WorkoutListFragment`.

2. **When an item is clicked in WorkoutListFragment, the fragment's onListItemClick() method is called.**
This calls `MainActivity`'s `itemClicked()` method, passing it the ID of the workout that was clicked; in this example, the ID is `1`.

3. **MainActivity's itemClicked() method sees that the app is running on a tablet.**
It creates a new instance of `WorkoutDetailFragment`, and begins a new fragment transaction.

The story continues...

4 **As part of the transaction, WorkoutDetailFragment's views are updated with details of the workout that was selected, in this case the one with ID 1.**
The fragment is added to the `FrameLayout` `fragment_container` in `MainActivity`'s layout, and the whole transaction is added to the back stack.

5 **MainActivity commits the transaction.**
All of the changes specified in the transaction take effect, and the `WorkoutDetailFragment` is displayed next to `WorkoutListFragment`.

Let's take the app for a test drive.

Test drive the app

- Create AVD
- Create layout
- **Show workout**

When we run the app, a list of the workouts appears on the left side of the screen. When we click on one of the workouts, details of that workout appear on the right. If we click on another workout and then click on the Back button, details of the workout we chose previously appear on the screen.

The app seems to be working fine as long as we don't rotate the screen. If we change the screen orientation, there's a problem. Let's see what happens.

Rotating the tablet breaks the app

When you run the app on a phone and rotate the device, the app works as you'd expect. Details of the workout the user selected continue to be displayed on the screen:

But when you run the app on a tablet, there's a problem. Regardless of which workout you've chosen, when you rotate the device, the app displays details of the first workout in the list:

Why does the app do this? Give this some thought before turning the page. Hint: you saw behavior similar to this back in Chapter 4 when we looked at the activity lifecycle.

Saving an activity's state (revisited)

When we first looked at the activity lifecycle back in Chapter 4, you saw how when you rotate the device, Android destroys and recreates the activity. When this happens, local variables used by the activity can get lost. To prevent this from happening, we saved the state of our local variables in the activity's `onSaveInstanceState()` method:

```
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
}
```

Earlier in the book, we used the onSaveInstanceState() method to save the state of these two variables.

We then restored the state of the variables in the activity's `onCreate()` method:

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    if (savedInstanceState != null) {
        seconds = savedInstanceState.getInt("seconds");
        running = savedInstanceState.getBoolean("running");
    }
    ...
}
```

We restored the state of the variables in the onCreate() method.

So what does this have to do with our current problem?

Fragments can lose state too

If the activity uses a fragment, **the fragment gets destroyed and recreated along with the activity**. This means that any local variables used by the fragment can also lose their state.

In our `WorkoutDetailFragment` code, we use a local variable called `workoutId` to store the ID of the workout the user clicks on in the `WorkoutListFragment` list view. When the user rotates the device, `workoutId` loses its current value and it's set to `0` by default. The fragment then displays details of the workout with an ID of `0`—the first workout in the list.

When you rotate the tablet, WorkoutDetailFragment loses the value of workoutId, and sets it back to its default value of 0.

Save the fragment's state...

You deal with this problem in a fragment in a similar way to how you deal with it in an activity.

You first override the fragment's `onSaveInstanceState()` method. This method works in a similar way to an activity's `onSaveInstanceState()` method. It gets called before the fragment gets destroyed, and it has one parameter: a `Bundle`. You use the `Bundle` to save the values of any variables whose state you need to keep.

In our case, we want to save the state of our `workoutId` variable, so we'd use code like this:

```
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putLong("workoutId", workoutId);
}
```

The onSaveInstanceState() method gets called before the fragment is destroyed.

Once you've saved the state of any variables, you can restore it when the fragment is recreated.

...then use onCreate() to restore the state

Just like an activity, a fragment has an `onCreate()` method that has one parameter, a `Bundle`. This is the `Bundle` to which you saved the state of your variables in the fragment's `onSaveInstanceState()` method, so you can use it to restore the state of those variables in your fragment's `onCreate()` method.

In our case, we want to restore the state of the `workoutId` variable, so we can use code like this:

```
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    if (savedInstanceState != null) {
        workoutId = savedInstanceState.getLong("workoutId");
    }
}
```

We can use this Bundle to get the previous state of the workoutId variable.

We'll show you the full code on the next page.

The updated code for WorkoutDetailFragment.java

We've updated our code for *WorkoutDetailFragment.java* to save the state of the `workoutId` variable before the fragment is destroyed, and restore it if the fragment is recreated. Here's our code; update your version of *WorkoutDetailFragment.java* to reflect our changes.

```
package com.hfad.workout;

import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            workoutId = savedInstanceState.getLong("workoutId");
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }
```

Add the onCreate() method.

Set the value of the workoutId.

The code continues on the next page.

WorkoutDetailFragment.java (continued)

```
    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putLong("workoutId", workoutId);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

Save the value of the workoutId in the savedInstanceState Bundle before the fragment gets destroyed. We're retrieving it in the onCreate() method.

Test drive the app

Now, when you run the app on a tablet and rotate the device, details of the workout the user selected continue to be displayed on the screen.

Your Android Toolbox

You've got Chapter 10 under your belt and now you've added fragments for larger interfaces to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Make apps look different on different devices by putting separate layouts in device-appropriate folders.
- Android keeps track of places you've visited within an app by adding them to the back stack as separate transactions. Pressing the Back button pops the last transaction off the back stack.
- Use a frame layout to add, replace, or remove fragments programmatically using fragment transactions.
- Begin the transaction by calling the `FragmentManager beginTransaction()` method. This creates a `FragmentTransaction` object.
- Add, replace, and delete fragments using the `FragmentTransaction add()`, `replace()`, and `remove()` methods.
- Add a transaction to the back stack using the `FragmentTransaction addToBackStack()` method.
- Commit a transaction using the `FragmentTransaction commit()` method. This applies all the updates in the transaction.
- Save the state of a fragment's variables in the `Fragment onSaveInstanceState()` method.
- Restore the state of a fragment's variables in the `Fragment onCreate()` method.

11 dynamic fragments

Nesting Fragments

So far you've seen how to create and use static fragments.

But what if you want your fragments to be more **dynamic**? Dynamic fragments have a lot in common with dynamic activities, but there are crucial differences you need to be able to deal with. In this chapter you'll see how to **convert dynamic activities** into **working dynamic fragments**. You'll find out how to use **fragment transactions** to help **maintain your fragment state**. Finally, you'll discover how to **nest one fragment inside another**, and how the **child fragment manager** helps you control unruly back stack behavior.

Adding dynamic fragments

In Chapters 9 and 10, you saw how to create fragments, how to include them in activities, and how to connect them together. To do this, we created a list fragment displaying a list of workouts, and a fragment displaying details of a single workout.

These fragments we've created so far have both been static. Once the fragments are displayed, their contents don't change. We may completely replace the fragment that's displayed with a new instance, but we can't update the contents of the fragment itself.

In this chapter we're going to look at how you deal with a fragment that's more dynamic. By this, we mean a fragment whose views gets updated after the fragment is displayed. To learn how to do, we're going to change the stopwatch *activity* we created in Chapter 4 into a stopwatch *fragment*. We're going to add our new stopwatch fragment to `WorkoutDetailFragment` so that it's displayed underneath the details of the workout.

We're only showing the tablet version of the app here, but the new stopwatch fragment will appear in the phone version too.

WorkoutDetailFragment displays details of the workout the user clicks on.

We're going to add a stopwatch fragment to WorkoutDetailFragment.

WorkoutListFragment contains a list of workouts.

These lines won't appear in the actual app. We've added them here to show you each of the fragments.

Here's what we're going to do

There are a number of steps we'll go through to change the app to display the stopwatch:

1. **Convert StopwatchActivity into StopwatchFragment.**
   We'll take the `StopwatchActivity` code we created in Chapter 4, and change it into fragment code. We'll also display it in a new temporary activity called `TempActivity` so that we can check that it works. We'll temporarily change the app so that `TempActivity` starts when the app gets launched.

2. **Test StopwatchFragment.**
   The `StopwatchActivity` included Start, Stop, and Reset buttons.We need to check that these still work when the stopwatch code is in a fragment.

   We also need to test what happens to `StopwatchFragment` when the user rotates the device.

3. **Add StopwatchFragment to WorkoutDetailFragment.**
   Once we're satisfied that `StopwatchFragment` works, we'll add it to `WorkoutDetailFragment`.

We'll start by adding StopwatchFragment to a new activity called TempActivity.

When we're satisfied that StopwatchFragment works OK in TempActivity, we'll add it to WorkoutDetailFragment.

Let's get started.

Do this!

We're going to update the Workout app in this chapter, so open your original Workout project from Chapter 9 in Android Studio.

The new version of the app

- **Convert stopwatch**
- Test stopwatch
- Add to fragment

We're going to change our app to get `StopwatchFragment` working in a new temporary activity called `TempActivity`. This will emable us to confirm that `StopwatchFragment` works before we add it to `WorkoutDetailFragment` later in the chapter.

Here's how the new version of the app will work:

1. **When the app gets launched, it starts TempActivity.** `TempActivity` uses *activity_temp.xml* for its layout, and contains a fragment, `StopwatchFragment`.

2. **StopwatchFragment displays a stopwatch with Start, Stop, and Reset buttons.**

All of the other activities and fragments we created in Chapters 9 and 10 will still exist in the project, but we're not going to do anything with them until later in the chapter.

Convert stopwatch
Test stopwatch
Add to fragment

Create TempActivity

We'll start by creating `TempActivity`. Create a new empty activity by switching to the Project view of Android Studio's explorer, highlighting the *com.hfad.workout* package in the *app/src/main/java* folder, going to the File menu and choosing New...→Activity→Empty Activity. Name the activity "TempActivity", name the layout "activity_temp", make sure the package name is `com.hfad.workout`, and **check the Backwards Compatibility (AppCompat) checkbox**.

If prompted for the activity's source language, select the option for Java.

We're going to change our app so that, when it's launched, it starts `TempActivity` instead of `MainActivity`. To do this, we need to move `MainActivity`'s launcher intent filter to `TempActivity` instead. Open the file *AndroidManifest.xml* in the *app/src/main* folder, then make the following changes:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.workout">

    <application
        ...
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DetailActivity" />
        <activity android:name=".TempActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

This bit specifies that it's the main activity of the app.

This says the activity can be used to launch the app.

We'll update `TempActivity` on the next page.

TempActivity needs to extend AppCompatActivity

All of the fragments we're using in this app come from the Support Library. As we said back in Chapter 9, all activities that use Support Library fragments must extend the `FragmentActivity` class or one of its subclasses such as `AppCompatActivity`. If they don't, the code will break.

All of the other activities we've created in this app extend `AppCompatActivity`, so we'll make `TempActivity` extend this class too. Here's our code for *TempActivity.java*. Update your version of the code so that it matches ours below:

```
package com.hfad.workout;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class TempActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_temp);
    }
}
```

The activity extends AppCompatActivity.

We'll add a new stopwatch fragment

We're going to add a new stopwatch fragment called *StopwatchFragment.java* that uses a layout called *fragment_stopwatch.xml*. We're going to base the fragment on the stopwatch activity we created back in Chapter 4.

We already know that activities and fragments behave in similar ways, but we also know that a fragment is a distinct type of object—a fragment is not a subclass of activity. **Is there some way we could rewrite that stopwatch activity code so that it works like a fragment?**

Fragments and activities have similar lifecycles...

To understand how to rewrite an activity as a fragment, we need to think a little about the similarities and differences between them. If we look at the lifecycles of fragments and activities, we'll see that they're very similar:

Lifecycle method	Activity	Fragment
`onAttach()`		✓
`onCreate()`	✓	✓
`onCreateView()`		✓
`onActivityCreated()`		✓
`onStart()`	✓	✓
`onPause()`	✓	✓
`onResume()`	✓	✓
`onStop()`	✓	✓
`onDestroyView()`		✓
`onRestart()`	✓	
`onDestroy()`	✓	✓
`onDetach()`		✓

...but the methods are slightly different

Fragment lifecycle methods are almost the same as activity lifecycle methods, but there's one major difference: activity lifecycle methods are **protected** and fragment lifecycle methods are **public**. And we've already seen that the ways that activities and fragments create a layout from a layout resource file are different.

Also, in a fragment, we can't call methods like `findViewById()` directly. Instead, we need to find a reference to a `View` object, and then call the view's `findViewById()` method.

With these similarities and differences in mind, it's time you started to write some code...

Sharpen your pencil

This is the code for `StopwatchActivity` we wrote earlier. You're going to convert this code into a fragment called `StopwatchFragment`. With a pencil, make the changes you need. Keep the following things in mind:

- Instead of a layout file called *activity_stopwatch.xml*, it will use a layout called *fragment_stopwatch.xml*.
- Make sure the access restrictions on the methods are correct.
- How will you specify the layout?
- The `runTimer()` method won't be able to call `findViewById()`, so you might want to pass a `View` object into `runTimer()`.

```
public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
        runTimer();
    }

    @Override
    protected void onPause() {
        super.onPause();
        wasRunning = running;
        running = false;
    }
```

The number of seconds that have passed

running says whether the stopwatch is running.
wasRunning says whether the stopwatch was running before the stopwatch was paused.

If the activity was destroyed and recreated, restore the state of the variables from the savedInstanceState Bundle.

Start the runTimer() method.

Stop the stopwatch if the activity is paused.

```
    @Override
    protected void onResume() {
        super.onResume();
        if (wasRunning) {
            running = true;
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
        savedInstanceState.putBoolean("wasRunning", wasRunning);
    }

    public void onClickStart(View view) {
        running = true;
    }

    public void onClickStop(View view) {
        running = false;
    }

    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }

    private void runTimer() {
        final TextView timeView = (TextView)findViewById(R.id.time_view);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                int hours = seconds/3600;
                int minutes = (seconds%3600)/60;
                int secs = seconds%60;
                String time = String.format(Locale.getDefault(),
                        "%d:%02d:%02d", hours, minutes, secs);
                timeView.setText(time);
                if (running) {
                    seconds++;
                }
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

Start the stopwatch if the activity is resumed.

Save the activity's state before the activity is destroyed.

Start, stop, or reset the stopwatch depending on which button is clicked.

Use a Handler to post code to increment the number of seconds and update the text view every second.

Sharpen your pencil Solution

This is the code for `StopwatchActivity` we wrote earlier. You're going to convert this code into a fragment called `StopwatchFragment`. With a pencil, make the changes you need. Keep the following things in mind:

- Instead of a layout file called *activity_stopwatch.xml*, it will use a layout called *fragment_stopwatch.xml*.
- Make sure the access restrictions on the methods are correct.
- How will you specify the layout?
- The `runTimer()` method won't be able to call `findViewById()`, so you might want to pass a `View` object into `runTimer()`.

This is the new name.

```
public class StopwatchActivity StopwatchFragment extends Activity Fragment {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

    @Override
    protected public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
        runTimer();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View layout = inflater.inflate(R.layout.fragment_stopwatch, container, false);
        runTimer(layout);
        return layout;
    }

    @Override
    protected public void onPause() {
        super.onPause();
        wasRunning = running;
        running = false;
    }
```

We're extending Fragment, not Activity.

This method needs to be public.

You don't set a fragment's layout in its onCreate() method.

We're not calling runTimer() yet because we've not set the layout—we don't have any views yet.

We can leave this code in the onCreate() method.

We set the fragment's layout in the onCreateView() method.

Pass the layout view to the runTimer() method.

This method needs to be public.

```
    @Override
    protected public void onResume() {
        super.onResume();
        if (wasRunning) {
            running = true;
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
        savedInstanceState.putBoolean("wasRunning", wasRunning);
    }

    public void onClickStart(View view) {
        running = true;
    }

    public void onClickStop(View view) {
        running = false;
    }

    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }

    private void runTimer(View view ) {
        final TextView timeView = (TextView) view.findViewById(R.id.time_view);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                int hours = seconds/3600;
                int minutes = (seconds%3600)/60;
                int secs = seconds%60;
                String time = String.format(Locale.getDefault(),
                        "%d:%02d:%02d", hours, minutes, secs);
                timeView.setText(time);
                if (running) {
                    seconds++;
                }
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

This method needs to be public.

The runTimer() method now takes a View.

Use the view parameter to call findViewById().

Convert stopwatch
Test stopwatch
Add to fragment

The StopwatchFragment.java code

We'll add `StopwatchFragment` to our Workout project so that we can use it in our app. You do this in the same way you did in Chapter 9. Highlight the *com.hfad.workout* package in the *app/src/main/java* folder, then go to File→New...→Fragment→Fragment (Blank). Give the fragment a name of "StopwatchFragment", give it a layout name of "fragment_stopwatch", and uncheck the options for including fragment factory methods and interface callbacks.

If prompted for the fragment's source language, select the option for Java.

When you click on the Finish button, Android Studio creates a new fragment for you in a file called *StopwatchFragment.java* in the *app/src/main/java* folder. Replace the fragment code Android Studio gives you with the following code (this is the code you updated in the exercise on the previous page):

```
package com.hfad.workout;

import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import java.util.Locale;

public class StopwatchFragment extends Fragment {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
    }
```

The number of seconds that have passed

running says whether the stopwatch is running.
wasRunning says whether the stopwatch was running before the stopwatch was paused.

Restore the state of the variables from the savedInstanceState Bundle.

The code continues on the next page.

Convert stopwatch
Test stopwatch
Add to fragment

StopwatchFragment.java (continued)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.fragment_stopwatch, container, false);
    runTimer(layout);
    return layout;
}
```

Set the fragment's layout and start the runTimer() method, passing in the layout.

```
@Override
public void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}
```

If the fragment's paused, record whether the stopwatch was running and stop it.

```
@Override
public void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}
```

If the stopwatch was running before it was paused, set it running again.

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}
```

Put the values of the variables in the Bundle before the activity is destroyed. These are used when the user turns the device.

```
public void onClickStart(View view) {
    running = true;
}
```

This code needs to run when the user clicks on the Start button.

The code continues on the next page.

StopwatchFragment.java (continued)

```
    public void onClickStop(View view) {
        running = false;
    }

    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }

    private void runTimer(View view) {
        final TextView timeView = (TextView) view.findViewById(R.id.time_view);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                int hours = seconds/3600;
                int minutes = (seconds%3600)/60;
                int secs = seconds%60;
                String time = String.format(Locale.getDefault(),
                        "%d:%02d:%02d", hours, minutes, secs);
                timeView.setText(time);
                if (running) {
                    seconds++;
                }
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

This code needs to run when the user clicks on the Stop button.

This code needs to run when the user clicks on the Reset button.

Putting the code in a Handler means it can run in the background thread.

Display the number of seconds that have passed in the stopwatch.

If the stopwatch is running, increment the number of seconds.

Run the Handler code every second.

That's all the Java code we need for our `StopwatchFragment`. The next thing we need to do is say what the fragment should look like by updating the layout code Android Studio gave us.

Convert stopwatch
Test stopwatch
Add to fragment

The StopwatchFragment layout

We'll use the same layout for `StopwatchFragment` as we used in our original Stopwatch app. To do so, replace the contents of *fragment_stopwatch.xml* with the code below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/time_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textAppearance="@android:style/TextAppearance.Large"
        android:textSize="56sp" />

    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="8dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />
```

The number of hours, minutes, and seconds that have passed.

The Start button

The Stop button

The Reset button code is on the next page.

The StopwatchFragment layout (continued)

- **Convert stopwatch**
- Test stopwatch
- Add to fragment

```
        <Button
            android:id="@+id/reset_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="8dp"
            android:onClick="onClickReset"
            android:text="@string/reset" />
    </LinearLayout>
```

The Reset button

The StopwatchFragment layout uses String values

The XML code in *fragment_stopwatch.xml* uses string values for the text on the Start, Stop, and Reset buttons. We need to add these to *strings.xml*:

```
    ...
        <string name="start">Start</string>
        <string name="stop">Stop</string>
        <string name="reset">Reset</string>
    ...
```

These are the button labels.

The Stopwatch fragment looks just like it did when it was an activity. The difference is that we can now use it in other activities and fragments.

The stopwatch looks the same as it did when it was an activity. But because it's now a fragment, we can reuse it in different places.

The next thing we need to do is display it in `TempActivity`'s layout.

Convert stopwatch
Test stopwatch
Add to fragment

Add StopwatchFragment to TempActivity's layout

The simplest way of adding `StopwatchFragment` to `TempActivity`'s layout is to use the `<fragment>` element. Using the `<fragment>` element means that we can add the fragment directly into the layout instead of writing fragment transaction code.

Here's our code for *activity_temp.xml*. Replace the code that's currently in that file with this updated code:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.StopwatchFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

This adds the fragment to the activity.

That's everything we need to see `StopwatchFragment` running. Let's take it for a test drive.

Test drive the app

When we run the app, `TempActivity` is displayed. It contains `StopwatchFragment`. The stopwatch is set to `0`.

When we run the app, TempActivity starts, not MainActivity. TempActivity displays StopwatchFragment as expected.

The next thing we'll do is check that `StopwatchFragment`'s buttons work OK.

The app crashes if you click on a button

Convert stopwatch
Test stopwatch
Add to fragment

When you click on any one of the buttons in the Workout app's new stopwatch, the app crashes:

This is what happened when we clicked on the Start button in StopwatchFragment.

When we converted the stopwatch activity into a fragment, we didn't change any of the code relating to the buttons. We know this code worked great when the stopwatch was in an activity, so why should it cause the app to crash in a fragment?

Here's the error output from Android Studio. Can you see what may have caused the problem?

Yikes.

```
04-13 11:56:43.623 10583-10583/com.hfad.workout E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.hfad.workout, PID: 10583
    java.lang.IllegalStateException: Could not find method onClickStart(View) in a
    parent or ancestor Context for android:onClick attribute defined on view class
    android.support.v7.widget.AppCompatButton with id 'start_button'
        at android.support.v7.app.AppCompatViewInflater$DeclaredOnClickListener.
            resolveMethod(AppCompatViewInflater.java:327)
        at android.support.v7.app.AppCompatViewInflater$DeclaredOnClickListener.
            onClick(AppCompatViewInflater.java:284)
        at android.view.View.performClick(View.java:5609)
        at android.view.View$PerformClick.run(View.java:22262)
        at android.os.Handler.handleCallback(Handler.java:751)
        at android.os.Handler.dispatchMessage(Handler.java:95)
        at android.os.Looper.loop(Looper.java:154)
        at android.app.ActivityThread.main(ActivityThread.java:6077)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.
            run(ZygoteInit.java:865)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:755)
```

Let's look at the StopwatchFragment layout code

In the layout code for the `StopwatchFragment`, we're binding the buttons to methods in the same way that we did for an activity, by using the `android:onClick` attribute to say which method should be called when each button is clicked:

We're using the same layout for the stopwatch now that it's a fragment as we did when it was an activity.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="8dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />

    <Button
        android:id="@+id/reset_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="8dp"
        android:onClick="onClickReset"
        android:text="@string/reset" />
</LinearLayout>
```

We're using the android:onClick attributes in the layout to say which methods should be called when each button is clicked.

This worked OK when we were using an activity, so why should we have a problem now that we're using a fragment?

The onClick attribute calls methods in the activity, not the fragment

There's a big problem with using the `android:onClick` attribute to say which method should be called when a view is clicked. The attribute specifies which method should be called in the **current activity**. This is fine when the views are in an *activity*'s layout. But when the views are in a *fragment*, this leads to problems. Instead of calling methods in the fragment, Android calls methods in the parent activity. If it can't find the methods in this activity, the app crashes. That's what Android Studio's error message was trying to tell us.

It's not just buttons that have this problem. The `android:onClick` attribute can be used with any views that are subclasses of the `Button` class. This includes checkboxes, radio buttons, switches, and toggle buttons.

Now we *could* move the methods out of the fragment and into the activity, but that approach has a major disadvantage. It would mean that the fragment is no longer self-contained—if we wanted to reuse the fragment in another activity, we'd need to include the code in *that* activity too. Instead, we'll deal with it in the fragment.

How to make button clicks call methods in the fragment

There are three things you need to do in order to get buttons in a fragment to call methods in the fragment instead of the activity:

1. **Remove references to android:onClick in the fragment layout.**
   Buttons attempt to call methods in the activity when the `onClick` attribute is used, so these need to be removed from the fragment layout.

2. **Optionally, change the onClick method signatures.**
   When we created our `onClickStart()`, `onClickStop()`, and `onClickReset()` methods, we made them public and gave them a single `View` parameter. This was so they'd get called when the user clicked on a button. As we're no longer using the `android:onClick` attribute in our layout, we can set our methods to private and remove the `View` parameter.

   This step's optional, but it's a good opportunity to tidy up our code.

3. **Bind the buttons to methods in the fragment by implementing an OnClickListener.**
   This will ensure that the right methods are called when the buttons are clicked.

Let's do this now in our `StopwatchFragment`.

Convert stopwatch
Test stopwatch
Add to fragment

1. Remove the onClick attributes from the fragment's layout

The first thing we'll do is remove the `android:onClick` lines of code from the fragment's layout. This will stop Android from trying to call methods in the activity when the buttons are clicked:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="8dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />

    <Button
        android:id="@+id/reset_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="8dp"
        android:onClick="onClickReset"
        android:text="@string/reset" />
</LinearLayout>
```

Remove the onClick attributes for each of the buttons in the stopwatch.

The next thing we'll do is tidy up our `onClickStart()`, `onClickStop()`, and `onClickReset()` code.

2. Change the onClick... method signatures

Convert stopwatch
Test stopwatch
Add to fragment

Back in Chapter 4, when we created our `onClickStart()`, `onClickStop()`, and `onClickReset()` methods in `StopwatchActivity`, we had to give them a specific method signature like this:

```
public void onClickStart(View view) {
}
```

The methods had to be public.

The methods had to have a void return value.

The methods had to have a single parameter of type View.

The methods had to take this form so that they'd respond when the user clicked on a button. Behind the scenes, when you use the `android:onClick` attribute, Android looks for a public method with a void return value, and with a name that matches the method specified in the layout XML.

Now that our code is in a fragment and we're no longer using the `android:onClick` attribute in our layout code, we can change our method signatures like this:

```
private void onClickStart() {
}
```

Our methods no longer need to be public, so we can make them private.

We no longer need the View parameter.

So let's update our fragment code. Change the `onClickStart()`, `onClickStop()`, and `onClickReset()` methods in *StopwatchFragment.java* to match ours:

```
...
    public private void onClickStart(View view) {
        running = true;
    }

    public private void onClickStop(View view) {
        running = false;
    }

    public private void onClickReset(View view) {
        running = false;
        seconds = 0;
    }
...
```

Change the methods to private.

Remove the View parameters.

3. Make the fragment implement OnClickListener

To make the buttons call methods in `StopwatchFragment` when they are clicked, we'll make the fragment implement the `View.OnClickListener` interface like this:

This turns the fragment into an OnClickListener.

```
public class StopwatchFragment extends Fragment implements View.OnClickListener {
    ...
}
```

This turns `StopwatchFragment` into a type of `View.OnClickListener` so that it can respond when views are clicked.

You tell the fragment how to respond to clicks by implementing the `View.OnClickListener` `onClick()` method. This method gets called whenever a view in the fragment is clicked.

```
@Override
public void onClick(View v) {
    ...
}
```

You must override the onClick() method in your fragment code.

The `onClick()` method has a single `View` parameter. This is the view that the user clicks on. You can use the view's `getId()` method to find out which view the user clicked on, and then decide how to react.

Code Magnets

See if you can complete the `StopwatchFragment` `onClick()` method. You need to call the `onClickStart()` method when the Start button is clicked, the `onClickStop()` method when the Stop button is clicked, and the `onClickReset()` method when the Reset button is clicked.

```
@Override
public void onClick(View v) {

    switch (..........: ..................) {
        case R.id.start_button:
            onClickStart();
            break;

        case ...................................:

            ...........................();
            break;
        case R.id.reset_button:

            ...........................();
    }
}
```

getId() | R.id.stop_button | getName() | onClickReset | onClickStop | v | true | View | true | true

Code Magnets Solution

See if you can complete the `StopwatchFragment onClick()` method. You need to call the `onClickStart()` method when the Start button is clicked, the `onClickStop()` method when the Stop button is clicked, and the `onClickReset()` method when the Reset button is clicked.

```
@Override
public void onClick(View v) {

    switch (v.getId()) {
        case R.id.start_button:
            onClickStart();
            break;

        case R.id.stop_button:
            onClickStop();
            break;
        case R.id.reset_button:
            onClickReset();

    }
}
```

You didn't need to use these magnets.

`getName()` `true` `true` `true` `View`

The StopwatchFragment onClick() method

We need to make a few changes to *StopwatchFragment.java*; we'll show you the changes one at a time, then the fully updated code a couple of pages ahead.

Here's the code to implement the `StopwatchFragment onClick()` method so that the correct method gets called when each button is clicked:

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.start_button:
            onClickStart();
            break;
        case R.id.stop_button:
            onClickStop();
            break;
        case R.id.reset_button:
            onClickReset();
            break;
    }
}
```

This is the View the user clicked on.

Check which View was clicked.

If the Start button was clicked, call the onClickStart() method.

If the Stop button was clicked, call the onClickStop() method.

If the Reset button was clicked, call the onClickReset() method.

There's just one more thing we need to do to get our buttons working: attach the listener to the buttons in the fragment.

Convert stopwatch
Test stopwatch
Add to fragment

Attach the OnClickListener to the buttons

To make views respond to clicks, you need to call each view's `setOnClickListener()` method. The `setOnClickListener()` method takes an `OnClickListener` object as a parameter. Because `StopwatchFragment` implements the `OnClickListener` interface, we can use the keyword `this` to pass the fragment as the `OnClickListener` in the `setOnClickListener()` method.

As an example, here's how you attach the `OnClickListener` to the Start button:

Get a reference to the button.

```
Button startButton = (Button) layout.findViewById(R.id.start_button);
startButton.setOnClickListener(this);
```

Attach the listener to the button.

The call to each view's `setOnClickListener()` method needs to be made after the fragment's views have been created. This means they need to go in the `StopwatchFragment` `onCreateView()` method like this:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.fragment_stopwatch, container, false);
    runTimer(layout);
    Button startButton = (Button)layout.findViewById(R.id.start_button);
    startButton.setOnClickListener(this);
    Button stopButton = (Button)layout.findViewById(R.id.stop_button);
    stopButton.setOnClickListener(this);
    Button resetButton = (Button)layout.findViewById(R.id.reset_button);
    resetButton.setOnClickListener(this);
    return layout;
}
```

This attaches the listener to each of the buttons.

We'll show you the full `StopwatchFragment` code on the next page.

The StopwatchFragment code

Here's the revised code for *StopwatchFragment.java*; update your version to match ours:

```
package com.hfad.workout;

import java.util.Locale;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.widget.Button;

public class StopwatchFragment extends Fragment implements View.OnClickListener {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View layout = inflater.inflate(R.layout.fragment_stopwatch, container, false);
        runTimer(layout);
        Button startButton = (Button)layout.findViewById(R.id.start_button);
        startButton.setOnClickListener(this);
        Button stopButton = (Button)layout.findViewById(R.id.stop_button);
        stopButton.setOnClickListener(this);
        Button resetButton = (Button)layout.findViewById(R.id.reset_button);
        resetButton.setOnClickListener(this);
        return layout;
    }
```

We're using the Button class, so we'll import it.

The fragment needs to implement the View.OnClickListener interface.

We're not changing the onCreate() method.

Update the onCreateView() method to attach the listener to the buttons.

The code continues on the next page.

The StopwatchFragment code (continued)

- Convert stopwatch
- **Test stopwatch**
- Add to fragment

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.start_button:
            onClickStart();
            break;
        case R.id.stop_button:
            onClickStop();
            break;
        case R.id.reset_button:
            onClickReset();
            break;
    }
}
```

As we're implementing the OnClickListener interface, we need to override the onClick() method.

Call the appropriate method in the fragment for the button that was clicked.

```
@Override
public void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}

@Override
public void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}
```

We've not changed these methods.

The code continues on the next page.

The StopwatchFragment code (continued)

```
    private void onClickStart() {
        running = true;
    }

    private void onClickStop() {
        running = false;
    }

    private void onClickReset() {
        running = false;
        seconds = 0;
    }
```

We've changed these methods so they're private. We've also removed the View parameter, as we no longer needed it.

We've not changed this method.

```
    private void runTimer(View view) {
        final TextView timeView = (TextView) view.findViewById(R.id.time_view);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                int hours = seconds/3600;
                int minutes = (seconds%3600)/60;
                int secs = seconds%60;
                String time = String.format(Locale.getDefault(),
                        "%d:%02d:%02d", hours, minutes, secs);
                timeView.setText(time);
                if (running) {
                    seconds++;
                }
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

Those are all the code changes needed for *StopwatchFragment.java*.
Let's see what happens when we run the app.

Test drive the app

- Convert stopwatch
- **Test stopwatch**
- Add to fragment

When we run the app, the stopwatch is displayed as before. This time, however, the Start, Stop, and Reset buttons work.

Now that we've got the buttons working, the next thing we need to test is what happens when we rotate the device.

Rotating the device resets the stopwatch

Convert stopwatch
Test stopwatch
Add to fragment

There's still one more problem we need to sort out. When we rotate our device, the stopwatch gets reset back to 0.

We encountered a similar problem when we first created `StopwatchActivity` back in Chapter 4. `StopwatchActivity` lost the state of any instance variables when it was rotated because activities are destroyed and recreated when the device is rotated. We solved this problem by saving and restoring the state of any instance variables used by the stopwatch.

This time, the problem isn't due to the code in `StopwatchFragment`. Instead, it's because of how we're adding `StopwatchFragment` to `TempActivity`.

Convert stopwatch
Test stopwatch
Add to fragment

Use <fragment> for static fragments...

When we added `StopwatchFragment` to `TempActivity`, we did it by adding a `<fragment>` element to its layout like this:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.StopwatchFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

We did this because it was the simplest way to display our fragment in an activity and see it working.

As we said back in Chapter 9, the `<fragment>` element is a placeholder for where the fragment's layout should be inserted. When Android creates the activity's layout, it replaces the `<fragment>` element with the fragment's user interface.

When you rotate the device, Android recreates the activity. If your activity contains a `<fragment>` element, *it reinserts a new version of the fragment each time the activity is recreated*. The old fragment is discarded, and any instance variables are set back to their original values. In this particular example, this means that the stopwatch is set back to 0.

...but dynamic fragments need a fragment transaction

The `<fragment>` element works well for fragments that display static data. If you have a fragment that's dynamic, like our stopwatch, you need to add the fragment using a fragment transaction instead.

We're going to change `TempActivity` so that we no longer display `StopwatchFragment` using a `<fragment>`. Instead, we'll use a fragment transaction. To do this, we need to make changes to *activity_temp.xml* and *TempActivity.java*.

Change activity_temp.xml to use a FrameLayout

As you learned back in Chapter 10, when you want to add a fragment to an activity using a fragment transaction, you first need to add a placeholder for the fragment in the activity's layout. We did this in Chapter 10 by adding a frame layout to the layout, and giving it an ID so we could refer to it in our Java code.

We need to do the same thing with *activity_temp.xml*. We'll replace the `<fragment>` element with a frame layout, and give the frame layout an ID of `stopwatch_container`. Update your version of *activity_temp.xml* so that it reflects ours:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="com.hfad.workout.StopwatchFragment"
    android:id="@+id/stopwatch_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Replace the fragment with a FrameLayout.

Delete this line.

Add a fragment transaction to TempActivity.java

Once you've added the frame layout to your activity's layout, you can create the fragment transaction that will add the fragment to the frame layout.

We want to add `StopwatchFragment` to `TempActivity` as soon as `TempActivity` gets created. We only want to add a new fragment, however, if one hasn't previously been added to it. We don't want to override any existing fragment.

To do this, we'll add code to `TempActivity`'s `onCreate()` method that checks whether the `savedInstanceState` `Bundle` parameter is null.

If `savedInstanceState` is null, this means that `TempActivity` is being created for the first time. In that case, we need to add `StopwatchFragment` to the activity.

If `savedInstanceState` is not null, that means that `TempActivity` is being recreated after having been destroyed. In that situation, we don't want to add a new instance of `StopwatchFragment` to the activity, as it would overwrite an existing fragment.

Pool Puzzle

Your **job** is to take code snippets from the pool and place them into the blank lines in *TempActivity.java*. You may **not** use the same code snippet more than once, and you won't need to use all the code snippets. Your **goal** is to create a fragment transaction that will add an instance of `StopwatchFragment` to `TempActivity`.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_temp);
    if (savedInstanceState == null) {
        StopwatchFragment stopwatch = new StopwatchFragment();
        FragmentTransaction ft = ..................................................;
        ft.add(R.id.stopwatch_container, ..............);
        ft................................(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft. ...............................;
    }
}
```

Note: each snippet from the pool can only be used once!

Pool Puzzle Solution

Your **job** is to take code snippets from the pool and place them into the blank lines in *TempActivity.java*. You may **not** use the same code snippet more than once, and you won't need to use all the code snippets. Your **goal** is to create a fragment transaction that will add an instance of `StopwatchFragment` to `TempActivity`.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_temp);
    if (savedInstanceState == null) {
        StopwatchFragment stopwatch = new StopwatchFragment();
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.stopwatch_container, stopwatch);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }
}
```

This begins the fragment transaction. We need to use getSupportFragmentManager(), not getFragmentManager(), as we're using fragments from the Support Library.

Add the transaction to the back stack.

Add an instance of StopwatchFragment to TempActivity's layout.

Commit the transaction.

You didn't need to use these code snippets.

- Convert stopwatch
- **Test stopwatch**
- Add to fragment

The full code for TempActivity.java

We've added a fragment transaction to *TempActivity.java* that adds `StopwatchFragment` to `TempActivity`. Our full code is below. Update your version of *TempActivity.java* so that it matches ours.

```
package com.hfad.workout;

import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class TempActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_temp);
        if (savedInstanceState == null) {
            StopwatchFragment stopwatch = new StopwatchFragment();
            FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
            ft.add(R.id.stopwatch_container, stopwatch);
            ft.addToBackStack(null);
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        }
    }
}
```

You need to import the FragmentTransaction class from the Support Library.

We only want to add the fragment if the activity isn't being recreated after having been destroyed.

Begin the fragment transaction.

Add the stopwatch, and add the transaction to the back stack.

Set the fragment transition to fade in and out.

Commit the transaction. This applies the changes.

Those are all the code changes we need to add `StopwatchFragment` to `TempActivity` using a fragment transaction. Let's see what happens when we run the code.

test drive

Test drive the app

- Convert stopwatch
- **Test stopwatch**
- Add to fragment

When we run the app, the stopwatch is displayed as before. The Start, Stop, and Reset buttons all work, and when we rotate the app, the stopwatch keeps running.

At the beginning of the chapter, we said we'd first focus on getting `StopwatchFragment` working in a new temporary activity so that we could confirm it works OK. Now that we've achieved that, we can reuse it in `WorkoutDetailFragment`.

Add the stopwatch to WorkoutDetailFragment

We're going to add `StopwatchFragment` to `WorkoutDetailFragment` so that a stopwatch is displayed underneath details of the workout. The stopwatch will appear along with the workout details whenever the user chooses one of the workouts.

Here's how the app will work:

1. **When the app gets launched, it starts MainActivity.** `MainActivity` includes `WorkoutListFragment`, which displays a list of workouts.

2. **The user clicks on a workout and WorkoutDetailFragment is displayed.** `WorkoutDetailFragment` displays details of the workout, and contains `StopwatchFragment`.

3. **StopwatchFragment displays a stopwatch.**

We've simplified the app structure here, but these are the key points.

We'll go through the steps on the next page.

What we're going to do

Convert stopwatch
Test stopwatch
Add to fragment

There are just a couple of steps we need to go through in order to get the new version of the app up and running.

1. **Make the app start MainActivity when it launches.**
   Earlier in the chapter, we temporarily changed the app so that it would start `TempActivity`. We need to change the app so that it starts `MainActivity` again.

2. **Add StopwatchFragment to WorkoutDetailFragment.**
   We'll do this using a fragment transaction.

Let's get started.

Start MainActivity when the app launches

Earlier in the chapter, we updated *AndroidManifest.xml* to make the app start `TempActivity`. This was so that we could get `StopwatchFragment` working before adding it to `WorkoutDetailFragment`.

Now that `StopwatchFragment` is working, we need to start `MainActivity` again when the app launches. To do this, update *AndroidManifest.xml* with the following changes:

```
...
    <application
        ...
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DetailActivity" />
        <activity android:name=".TempActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
...
```

Add an intent filter to start MainActivity when the app is launched.

Remove the intent filter from TempActivity.

- ✓ Convert stopwatch
- ✓ Test stopwatch
- → Add to fragment

Add a FrameLayout where the fragment should appear

Next we need to add `StopwatchFragment` to `WorkoutDetailFragment`. We'll do this by adding a frame layout to *fragment_workout_detail.xml*, just as we did in *activity_temp.xml*. We'll then be able to add `StopwatchFragment` to `WorkoutDetailFragment` using a fragment transaction.

Here's our code for *fragment_workout_detail.xml*; update your code so that it matches ours:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/textTitle" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textDescription" />

    <FrameLayout
        android:id="@+id/stopwatch_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

All that's left to do is to add the fragment transaction to `WorkoutDetailFragment`.

Convert stopwatch
Test stopwatch
Add to fragment

So far, we've only used fragment transactions in activities

Earlier in the chapter, we added the following code to `TempActivity` to add `StopwatchFragment` to its layout:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_temp);
    if (savedInstanceState == null) {
        StopwatchFragment stopwatch = new StopwatchFragment();
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.stopwatch_container, stopwatch);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }
}
```

This code adds StopwatchFragment to TempActivity when TempActivity is created.

The above code worked well when we wanted to add `StopwatchFragment` to an activity. How will it need to change now that we want to add `StopwatchFragment` to a *fragment*?

Using fragment transactions in fragments uses most of the same code

The good news is that you can use nearly all of the same code when you want to use a fragment transaction inside a fragment. There's just one key difference: fragments don't have a method called `getSupportFragmentManager()`, so we need to edit this line of code:

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
```

In order to create the fragment transaction, we need to get a reference to a fragment manager. Fragments have *two* methods you can use for this purpose: **`getFragmentManager()`** and **`getChildFragmentManager()`**. So what's the difference between these two methods, and which one should we use in our app?

- Convert stopwatch
- Test stopwatch
- **Add to fragment**

Using getFragmentManager() creates extra transactions on the back stack

The `getFragmentManager()` method gets the fragment manager associated with the fragment's *parent activity*. Any fragment transaction you create using this fragment manager is added to the back stack as a separate transaction.

In our case, when someone clicks on a workout, we want the app to display the details of the workout and the stopwatch. `MainActivity` creates a transaction that displays `WorkoutDetailFragment`. If we use `getFragmentManager()` to create a transaction to display `StopwatchFragment`, this will be added to the back stack as a separate transaction.

The problem with using two transactions to display the workout and stopwatch is what happens when the user presses the Back button.

Suppose the user clicks on a workout. Details of the workout will be displayed, along with the stopwatch. If the user then clicks on the Back button, they will expect the screen to go back to how it looked before they selected a workout. But **the Back button simply pops the last transaction on the back stack**. That means if we create two transactions to add the workout detail and the stopwatch, when the user clicks the Back button, only the stopwatch will be removed. They have to click the Back button again to remove the workout details.

A transaction for WorkoutDetailFragment is added to the back stack, followed by a separate transaction for StopwatchFragment.

When the user hits the Back button, the StopwatchFragment transaction is popped off the back stack. The transaction for WorkoutDetailFragment stays on the back stack.

The user clicks an item in the list just once to display the workout details and stopwatch.

The user has to click the Back button **twice** to get back to where they started. Clicking the Back button once only removes the stopwatch.

Clearly this behavior is less than ideal. So what about `getChildFragmentManager()`?

Using getChildFragmentManager() creates nested transactions instead

- Convert stopwatch
- Test stopwatch
- **Add to fragment**

The `getChildFragmentManager()` method gets the fragment manager associated with the fragment's *parent fragment*. Any fragment transaction you create using this fragment manager is added to the back stack inside the parent fragment transaction, not as a separate transaction.

In our particular case, this means that the fragment transaction that displays `WorkoutDetailFragment` contains a second transaction that displays `StopwatchFragment`.

I display workout details, and I also display the stopwatch.

The transaction to add StopwatchFragment is nested inside the transaction to add WorkoutDetailFragment.

`WorkoutDetailFragment` and `StopwatchFragment` are still displayed when the user clicks on a workout, but the behavior is different when the user clicks on the Back button. As the two transactions are nested, *both* transactions are popped off the back stack when the user presses the Back button. The workout details and the stopwatch are both removed if the user presses the Back button once. That's what we want, so we'll use this method in our app.

This time the user has to press the Back button just once to undo both the workout detail and stopwatch transactions.

Convert stopwatch
Test stopwatch
Add to fragment

What getChildFragmentManager() fragment transaction code looks like

We've written code that will add `StopwatchFragment` to `WorkoutDetailFragment`. It creates a fragment transaction using the fragment manager returned by `getChildFragmentManager()`. Here's the code:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState == null) {
        StopwatchFragment stopwatch = new StopwatchFragment();
        FragmentTransaction ft = getChildFragmentManager().beginTransaction();
        ft.add(R.id.stopwatch_container, stopwatch);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    } else {
        workoutId = savedInstanceState.getLong("workoutId");
    }
}
```

We're using getChildFragmentManager() instead of getSupportFragmentManager(). Apart from that, the code is the same as we had earlier.

We need to add this code to *WorkoutDetailFragment.java*. We'll show you the full code on the next page.

there are no Dumb Questions

Q: I can see that the child fragment manager handles the case where I put one fragment inside another. But what if I put one fragment inside another, inside another, inside another...?

A: The transactions will all be nested within each other, leaving just a single transaction at the activity level. So the nested set of child transactions will be undone by a single Back button click.

Q: Fragments seem more complicated than activities. Should I use fragments in my apps?

A: That depends on your app and what you want to achieve. One of the major benefits of using fragments is that you can use them to support a wide range of different screen sizes. You can, say, choose to display fragments side by side on tablets and on separate screens on smaller devices. You'll also see in the next chapter that some UI designs require you to use fragments.

The full WorkoutDetailFragment.java code

Convert stopwatch
Test stopwatch
Add to fragment

Here's the full code for *WorkoutDetailFragment.java*. Update your version of the code to include our changes.

```
package com.hfad.workout;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
        if (savedInstanceState == null) {
            StopwatchFragment stopwatch = new StopwatchFragment();
            FragmentTransaction ft = getChildFragmentManager().beginTransaction();
            ft.add(R.id.stopwatch_container, stopwatch);
            ft.addToBackStack(null);
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        } else {
            workoutId = savedInstanceState.getLong("workoutId");
        }
    }
```

You need to import the FragmentTransaction class from the Support Library.

Delete this line.

We only want to add the fragment if the activity isn't being recreated after having been destroyed.

Begin the fragment transaction.

Add the stopwatch, and add the transaction to the back stack.

Set the fragment transition to fade in and out.

Commit the transaction.

The code continues on the next page.

The full code (continued)

- Convert stopwatch
- Test stopwatch
- **Add to fragment**

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putLong("workoutId", workoutId);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

We didn't change any of the methods on this page.

That's everything we need for our app. Let's take it for a test drive and check that it works OK.

Test drive the app

- Convert stopwatch
- Test stopwatch
- **Add to fragment**

We'll start by testing the app on a tablet.

When we start the app, `MainActivity` is displayed.

When we click on one of the workouts, details of that workout are displayed along with a stopwatch. If we click on a second workout and then click on the Back button, details of the first workout are displayed.

Test drive (continued)

- ☑ Convert stopwatch
- ☑ Test stopwatch
- ☑ **Add to fragment**

When we click on the stopwatch buttons, they all work as expected. When we rotate the app, the stopwatch maintains its state.

When we run the app on a phone, `WorkoutDetailFragment` is displayed inside a separate activity, `DetailActivity`. The stopwatch is still displayed underneath the workout details, and functions as expected.

Your Android Toolbox

You've got Chapter 11 under your belt and now you've added dynamic fragments to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Fragments can contain other fragments.
- If you use the `android:onClick` attribute in a fragment, Android will look for a method of that name in the fragment's parent activity.
- Instead of using the `android:onClick` attribute in a fragment, make the fragment implement the `View.OnClickListener` interface and implement its `onClick()` method.
- If you use the `<fragment>` element in your layout, the fragment gets recreated when you rotate the device. If your fragment is dynamic, use a fragment transaction instead.
- Fragments contain two methods for getting a fragment manager, `getFragmentManager()` and `getChildFragmentManager()`.
- `getFragmentManager()` gets a reference to the fragment manager associated with the fragment's parent activity. Any fragment transactions you create using this fragment manager are added to the back stack as extra transactions.
- `getChildFragmentManager()` gets a reference to the fragment manager associated with the fragment's parent fragment. Any fragment transactions you create using this fragment manager are nested inside the parent fragment transaction.

12 design support library

Swipe Right

Ever wondered how to develop apps with a rich, slick UI?

With the release of the **Android Design Support Library**, it became much easier to create apps with an intuitive UI. In this chapter, we'll show you around some of the highlights. You'll see how to add ***tabs*** so that your users can *navigate around your app more easily*. You'll discover how to ***animate your toolbars*** so that they can *collapse or scroll on a whim*. You'll find out how to add ***floating action buttons*** for common user actions. Finally, we'll introduce you to ***snackbars***, a way of displaying short, informative messages to the user that they can interact with.

The Bits and Pizzas app revisited

In Chapter 8, we showed you a sketch of the top-level screen of the Bits and Pizzas app. It contained a list of places in the app the user could go to. The first three options linked to category screens for pizzas, pasta, and stores, and the final option linked to a screen where the user could create an order.

So far you've seen how to add actions to the app bar. These are used for simple commands, such as Create Order or Send Feedback. But what about the category screens? As we want to use these for navigating through the app rather than taking an action, we'll take a different approach.

We're going to change the Bits and Pizzas app so that it uses **tab navigation**. We'll display a set of tabs underneath the toolbar, with each option on a different tab. When the user clicks on a tab, the screen for that option will be displayed. We'll also let the user swipe left and right between the different tabs.

The app structure

We're going to change `MainActivity` so that it uses tabs. The tabs will include options for Home, Pizzas, Pasta, and Stores, so that the user can easily navigate to the main sections of the app.

We'll create fragments for these different options; when the user clicks on one of the tabs, the fragment for that option will be displayed:

This is what the new version of the app will look like.

We'll go through the steps for how to do this on the next page.

Here's what we're going to do

There are three main steps we'll go through to get tabs working:

1. **Create the fragments.**
   We'll create basic versions of `TopFragment`, `PizzaFragment`, `PastaFragment`, and `StoresFragment` so that we can easily tell which fragment is displayed on each of the tabs.

2. **Enable swipe navigation between the fragments.**
   We'll update `MainActivity` so that the user can swipe between the different fragments.

3. **Add the tab layout.**
   Finally, we'll add a tab layout to `MainActivity` that will work in conjunction with the swipe navigation. The user will be able to navigate to each fragment by clicking on a tab, or swiping between them.

Do this!

We're going to update the Bits and Pizzas app in this chapter, so open your original Bits and Pizzas project from Chapter 8 in Android Studio.

We'll start by creating the fragments.

Create TopFragment

We'll use `TopFragment` to display content that will appear on the Home tab. For now, we'll display the text "Top fragment" so that we know which fragment is displayed. Highlight the *com.hfad.bitsandpizzas* package in the *app/src/main/java* folder, then go to File→New...→Fragment→Fragment (Blank). Name the fragment "TopFragment" and name its layout "fragment_top". Then replace the code for *TopFragment.java* with the code below:

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class TopFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_top, container, false);
    }
}
```

TopFragment.java is a fragment from the Support Library.

Add the following string resource to *strings.xml*; we'll use this in our fragment layout:

```
<string name="title_top">Top fragment</string>
```

Add this to strings.xml. We'll use it in the layout so we know when TopFragment is being displayed.

Then update the code for *fragment_top.xml* as follows:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.TopFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/title_top" />
</LinearLayout>
```

Create PizzaFragment

We'll use a `ListFragment` called `PizzaFragment` to display the list of pizzas. Highlight the *com.hfad.bitsandpizzas* package in the *app/src/main/java* folder, then go to File→New...→Fragment→Fragment (Blank). Name the fragment "PizzaFragment", and uncheck the option to create a layout. Why? Because list fragments don't need a layout—they use their own.

Don't choose the Fragment (List) option, as this generates more complex code.

Next, add a new string array resource called `"pizzas"` to *strings.xml* (this contains the names of the pizzas):

```
<string-array name="pizzas">
    <item>Diavolo</item>
    <item>Funghi</item>
</string-array>
```

Add the array of pizzas to strings.xml.

Then change the code for *PizzaFragment.java* so that it's a `ListFragment`. Its list view needs to be populated with the pizza names. Here's the updated code:

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class PizzaFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
                inflater.getContext(),
                android.R.layout.simple_list_item_1,
                getResources().getStringArray(R.array.pizzas));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

We'll use a ListFragment to display the list of pizzas.

The ArrayAdapter populates the ListFragment's ListView with the pizza names.

Create PastaFragment

We'll use a `ListFragment` called `PastaFragment` to display the list of pasta. Highlight the *com.hfad.bitsandpizzas* package in the *app/src/main/java* folder and create a new blank fragment named "PastaFragment". You can uncheck the option to create a layout, as list fragments use their own layouts.

Next, add a new string array resource called `"pasta"` to *strings.xml* (this contains the names of the pasta):

```
<string-array name="pasta">
    <item>Spaghetti Bolognese</item>
    <item>Lasagne</item>
</string-array>
```

Add the array of pasta to strings.xml.

Then change the code for *PastaFragment.java* so that it's a `ListFragment` that displays a list of the pasta names. Here's the updated code:

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class PastaFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
                inflater.getContext(),
                android.R.layout.simple_list_item_1,
                getResources().getStringArray(R.array.pasta));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Create StoresFragment

We'll use a `ListFragment` called `StoresFragment` to display the list of stores. Highlight the *com.hfad.bitsandpizzas* package in the *app/src/main/java* folder and create a new blank fragment named "StoresFragment." Uncheck the option to create a layout, as list fragments define their own layouts.

Next, add a new string array resource called `"stores"` to *strings.xml* (this contains the names of the stores):

```
<string-array name="stores">
    <item>Cambridge</item>
    <item>Sebastopol</item>
</string-array>
```

Add the array of stores to strings.xml.

Then change the code for *StoresFragment.java* so that it's a `ListFragment`. Its list view needs to be populated with the store names. Here's the updated code:

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class StoresFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
                inflater.getContext(),
                android.R.layout.simple_list_item_1,
                getResources().getStringArray(R.array.stores));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

We'll use a ListFragment to display the list of stores.

We've now added all the fragments we need, so let's move on to the next step.

Use a view pager to swipe through fragments

We want to be able to swipe through the different fragments we've just created. To do this, we'll use a **view pager**, which is a view group that allows you to swipe through different pages in a layout, each page containing a separate fragment.

The view pager will let us swipe between the different fragments.

You use a view pager by adding it to your layout, then writing activity code to control which fragments should be displayed. The `ViewPager` class comes from the v4 Support Library, which is included in the v7 AppCompat Support Library, so you also need to make sure you add one of these libraries to your project as a dependency. In our particular case, we already added the v7 AppCompat Support Library to our project in Chapter 8.

You can check which Support Libraries are included in your project in Android Studio by choosing Project Structure from the File menu, clicking on the app module, and then choosing Dependencies.

What view pager layout code looks like

You add a view pager to your layout using code like this:

The ViewPager class is found in the v4 Support Library (which is included in the v7 AppCompat Support Library).

```
<android.support.v4.view.ViewPager
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

You need to give the ViewPager an ID so that you can control its behavior in your activity code.

The above code defines the view pager, and gives it an ID of `pager`. Every view pager you create *must* have an ID so that you can get a reference to it in your activity code. Without this ID, you can't specify which fragments should appear on each page of the view pager.

We're going to add a view pager to `MainActivity`. We'll look at the full code for its layout on the next page.

Add a view pager to MainActivity's layout

We're going to add a view pager to `MainActivity`'s layout, and remove the text view that's already there. Open the file *activity_main.xml*, then update your code so that it matches ours below (we've bolded our changes):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.MainActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/toolbar" />

    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

</LinearLayout>
```

Add the ViewPager below the Toolbar.

We're no longer displaying a TextView in MainActivity, so delete these lines of code.

That's everything we need to add a view pager to our layout. To get our new view pager to display fragments, we need to write some activity code. We'll do that next.

Tell a view pager about its pages using a fragment pager adapter

To get a view pager to display a fragment on each of its pages, there are two key pieces of information you need to give it: the number of pages it should have, and which fragment should appear on each page. You do this be creating a **fragment pager adapter**, and adding it to your activity code.

A fragment pager adapter is a type of adapter that specializes in adding fragments to pages in a view pager. You generally use one when you want to have a small number of pages that are fairly static, as each fragment the user visits is kept in memory.

If you want your view pager to have a large number of pages, you would use a **fragment state pager adapter** instead. We're not covering it here, but the code is almost identical.

Fragment pager adapter code looks like this:

```
private class SectionsPagerAdapter extends FragmentPagerAdapter {

    public SectionsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public int getCount() {
        //The number of pages in the ViewPager
    }

    @Override
    public Fragment getItem(int position) {
        //The fragment to be displayed on each page
    }
}
```

We're setting this to private, as we're going to add it to MainActivity as an inner class.

You need to extend the FragmentPagerAdapter class.

You must have a constructor that takes a FragmentManager parameter.

You need to override the getCount() method to specify the number of pages in the view pager.

You need to say which fragment should appear on each page. The position gives the page number, starting at 0.

When you create a fragment pager adapter, there are two key methods you ***must*** override: `getCount()` and `getItem()`. You use `getCount()` to specify how many pages there should be in the view pager, and the `getItem()` to say which fragment should be displayed on each page.

We'll show you the code for the Bits and Pizzas fragment pager adapter on the next page.

The code for our fragment pager adapter

We want our view pager to have four pages. We'll display `TopFragment` on the first page, `PizzaFragment` on the second, `PastaFragment` on the third, and `StoresFragment` on the fourth.

To accomplish this, we're going to create a fragment pager adapter called `SectionsPagerAdapter`. Here's the code (we'll add it to *MainActivity.java* in a couple of pages):

```
private class SectionsPagerAdapter extends FragmentPagerAdapter {

    public SectionsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public int getCount() {
        return 4;
    }

    @Override
    public Fragment getItem(int position) {
        switch (position) {
            case 0:
                return new TopFragment();
            case 1:
                return new PizzaFragment();
            case 2:
                return new PastaFragment();
            case 3:
                return new StoresFragment();
        }
        return null;
    }
}
```

We'll have four pages in our ViewPager, one for each of the fragments we want to be able to swipe through.

The getCount() method specifies 4 pages, so the getItem() method should only request the fragments for these 4 page positions.

We want to display TopFragment first, so we'll return a new instance of it for position 0 of the ViewPager.

That's all the code we need for our `SectionsPagerAdapter`. Next we need to get our view pager to use it.

Attach the fragment pager adapter to the view pager

Finally, we need to attach our `SectionsPagerAdapter` to the view pager so that the view pager can use it. You attach a fragment pager adapter to a view pager by calling the `ViewPager setAdapter()` method, and passing it a reference to an instance of the fragment pager adapter.

Here's the code to attach the fragment pager adapter we created to the view pager:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    //Attach the SectionsPagerAdapter to the ViewPager
    SectionsPagerAdapter pagerAdapter =
                        new SectionsPagerAdapter(getSupportFragmentManager());
    ViewPager pager = (ViewPager) findViewById(R.id.pager);
    pager.setAdapter(pagerAdapter);
}
```

This attaches the FragmentPagerAdapter we created to the ViewPager.

We're using support fragments, so we need to pass our adapter a reference to the support fragment manager.

That's everything we need to be able to swipe through our fragments. We'll show you the full code for `MainActivity` on the next page.

there are no Dumb Questions

Q: When should I use tabs in my app?

A: Tabs work well when you want to give the user a quick way of navigating between a small number of sections or categories. You would generally put each one on a separate tab.

Q: What if I have a large number of categories? Can I still use tabs?

A: You can, but you may want to consider other forms of navigation such as navigation drawers. These are panels that slide out from the side of the screen. We'll show you how to create them in Chapter 14.

Q: You mentioned the fragment state pager adapter. What's that?

A: It's very similar to a fragment pager adapter, except that it also handles saving and restoring a fragment's state. It uses less memory than a fragment pager adapter, as when pages aren't visible, the fragment it displays may be destroyed. It's useful if your view pager has a large number of pages.

The full code for MainActivity.java

Here's our full code for *MainActivity.java*. Update your version of the code to match our changes (in bold):

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.content.Intent;
import android.support.v7.widget.ShareActionProvider;
import android.support.v4.view.MenuItemCompat;
import android.support.v4.view.ViewPager;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

public class MainActivity extends AppCompatActivity {

    private ShareActionProvider shareActionProvider;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        //Attach the SectionsPagerAdapter to the ViewPager
        SectionsPagerAdapter pagerAdapter =
                      new SectionsPagerAdapter(getSupportFragmentManager());
        ViewPager pager = (ViewPager) findViewById(R.id.pager);
        pager.setAdapter(pagerAdapter);
    }
```

We're using these extra classes so we need to import them.

Attach the FragmentPagerAdapter to the ViewPager.

The code continues on the next page.

The MainActivity.java code (continued)

None of the code on this page has changed.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    MenuItem menuItem = menu.findItem(R.id.action_share);
    shareActionProvider =
            (ShareActionProvider) MenuItemCompat.getActionProvider(menuItem);
    setShareActionIntent("Want to join me for pizza?");
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_create_order:
            Intent intent = new Intent(this, OrderActivity.class);
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

private void setShareActionIntent(String text) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, text);
    shareActionProvider.setShareIntent(intent);
}
```

The code continues on the next page.

The MainActivity.java code (continued)

```
    private class SectionsPagerAdapter extends FragmentPagerAdapter {

        public SectionsPagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public int getCount() {
            return 4;
        }

        @Override
        public Fragment getItem(int position) {
            switch (position) {
                case 0:
                    return new TopFragment();
                case 1:
                    return new PizzaFragment();
                case 2:
                    return new PastaFragment();
                case 3:
                    return new StoresFragment();
            }
            return null;
        }
    }
}
```

The FragmentPagerAdapter passes information to the ViewPager.

Say how many pages the ViewPager should contain.

Specify which fragment should appear on each page.

Now that we've updated our `MainActivity` code, let's take our app for a test drive and see what happens.

Test drive the app

When we run the app, `TopFragment` is displayed. When we swipe the screen to the left, `PizzaFragment` is displayed, followed by `PastaFragment` and `StoresFragment`. When we swipe the screen in the opposite direction starting from `StoresFragment`, `PastaFragment` is displayed, followed by `PizzaFragment` and `TopFragment`.

Now that we can swipe through the fragments in `MainActivity`, let's add tabs.

Add tab navigation to MainActivity

- [x] Add fragments
- [x] Add swiping
- [ ] Add tabs

We're going to add tabs to `MainActivity` as an additional way of navigating through our fragments. Each fragment will be displayed on a separate tab, and clicking on each tab will show that fragment. We'll also be able to swipe through the tabs using the existing view pager.

The tabs will be displayed underneath the toolbar.

The Home tab is highlighted, as that's the current tab.

The user will still be able to swipe through the fragments as before.

You use tabs by adding them to your layout, then writing activity code to link the tabs to the view pager. The classes we need to do this come from the **Android Design Support Library**, so you need to add this library to your project as a dependency. To do this, choose File→Project Structure in Android Studio, click on the app module, then choose Dependencies. When you're presented with the project dependencies screen, click on the "+" button at the bottom or right side of the screen. When prompted, choose the Library Dependency option, then select the Design Library from the list of possible libraries. Finally, use the OK buttons to save your changes.

We'll look at the Design Support Library in more detail later in the chapter.

We've added the Android Design Support Library to our project as a dependency.

- ☑ Add fragments
- ☑ Add swiping
- ☐ Add tabs

How to add tabs to your layout

You add tabs to your layout using two components from the Design Support Library: a **`TabLayout`** and an **`AppBarLayout`**. You use a `TabLayout` to add the tabs, and the `AppBarLayout` to group the tabs and your toolbar together.

The code to add tabs to your layout looks like this:

```
<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize" />

    <android.support.design.widget.TabLayout
        android:id="@+id/tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</android.support.design.widget.AppBarLayout>
```

The AppBarLayout comes from the Design Support Library.

This line applies a theme to the Toolbar and TabLayout so that they have a consistant appearance.

You include the Toolbar inside the AppBarLayout.

The TabLayout comes from the DesignSupportLibrary. You add it to the AppBarLayout.

The `Toolbar` and `TabLayout` elements both have IDs because you need to be able to reference them in your activity code in order to control their behavior.

The `AppBarLayout` contains both the `Toolbar` and the `TabLayout`. It's a type of vertical linear layout that's designed to work with app bars. The `android:theme` attribute is used to style the `Toolbar` and `TabLayout`. We've given ours a theme of `ThemeOverlay.AppCompat.Dark.ActionBar`.

On the next page we'll show you the code to add tabs to `MainActivity`'s layout.

Add tabs to MainActivity's layout

- ☑ Add fragments
- ☑ Add swiping
- ☐ **Add tabs**

Here's our code for *activity_main.xml*. Update your version of the code to match our changes (in bold):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

        <include
            layout="@layout/toolbar_main"
            android:id="@+id/toolbar" />
        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize" />

        <android.support.design.widget.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

Add an AppBarLayout.

We've decided to put our Toolbar code in activity_main.xml instead of including it from a separate file. This is so that we can show you the full code in one place. In practice, using the <include> still works.

The Toolbar goes inside the AppBarLayout.

Add a TabLayout inside the AppBarLayout.

- [x] Add fragments
- [x] Add swiping
- [ ] Add tabs

Link the tab layout to the view pager

Once you've added the tab layout, you need to write some activity code to control it. Most of the tab layout's behavior (such as which fragment appears on which tab) comes from the view pager you've already created. All you need to do is implement a method in the view pager's fragment pager adapter to specify the text you want to appear on each tab, then link the view pager to the tab layout.

We're going to add the text we want to appear on each of the tabs as String resources. Open the file *strings.xml*, then add the following Strings:

```
<string name="home_tab">Home</string>
<string name="pizza_tab">Pizzas</string>
<string name="pasta_tab">Pasta</string>
<string name="store_tab">Stores</string>
```

These Strings will be displayed on the tabs.

To add the text to each of the tabs, you need to implement the fragment pager adapter's `getPageTitle()` method. This takes one parameter, an `int` for the tab's position, and needs to return the text that should appear on that tab. Here's the code we need to add the above String resources to our four tabs (we'll add it to *MainActivity.java* on the next page):

```
@Override
public CharSequence getPageTitle(int position) {
    switch (position) {
        case 0:
            return getResources().getText(R.string.home_tab);
        case 1:
            return getResources().getText(R.string.pizza_tab);
        case 2:
            return getResources().getText(R.string.pasta_tab);
        case 3:
            return getResources().getText(R.string.store_tab);
    }
    return null;
}
```

This is a new method in the fragment pager adapter we created earlier.

These lines of code add the String resources to the tabs.

Finally, you need to attach the view pager to the tab layout. You do this by calling the `TabLayout` object's `setupWithViewPager()` method, and passing in a reference to the `ViewPager` object as a parameter:

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
tabLayout.setupWithViewPager(pager);
```

This line attaches the ViewPager to the TabLayout. The TabLayout uses the ViewPager to determine how many tabs there should be, and what should be on each tab.

That's everything we need to get our tabs working. We'll show you the full code for `MainActivity` on the next page.

The full code for MainActivity.java

- ☑ Add fragments
- ☑ Add swiping
- ☐ **Add tabs**

Here's our full code for *MainActivity.java*. Update your version of the code to match our changes (in bold):

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.content.Intent;
import android.support.v7.widget.ShareActionProvider;
import android.support.v4.view.MenuItemCompat;
import android.support.v4.view.ViewPager;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.design.widget.TabLayout;

public class MainActivity extends AppCompatActivity {

    private ShareActionProvider shareActionProvider;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        //Attach the SectionsPagerAdapter to the ViewPager
        SectionsPagerAdapter pagerAdapter =
                      new SectionsPagerAdapter(getSupportFragmentManager());
        ViewPager pager = (ViewPager) findViewById(R.id.pager);
        pager.setAdapter(pagerAdapter);

        //Attach the ViewPager to the TabLayout
        TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
        tabLayout.setupWithViewPager(pager);
    }
```

We're using the TabLayout class, so we need to import it.

This links the ViewPager to the TabLayout.

The code continues on the next page.

The MainActivity.java code (continued)

None of the code on this page has changed.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    MenuItem menuItem = menu.findItem(R.id.action_share);
    shareActionProvider =
            (ShareActionProvider) MenuItemCompat.getActionProvider(menuItem);
    setShareActionIntent("Want to join me for pizza?");
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_create_order:
            Intent intent = new Intent(this, OrderActivity.class);
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

private void setShareActionIntent(String text) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, text);
    shareActionProvider.setShareIntent(intent);
}
```

The code continues on the next page.

The MainActivity.java code (continued)

- ✓ Add fragments
- ✓ Add swiping
- → Add tabs

```
    private class SectionsPagerAdapter extends FragmentPagerAdapter {

        public SectionsPagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public int getCount() {
            return 4;
        }

        @Override
        public Fragment getItem(int position) {
            switch (position) {
                case 0:
                    return new TopFragment();
                case 1:
                    return new PizzaFragment();
                case 2:
                    return new PastaFragment();
                case 3:
                    return new StoresFragment();
            }
            return null;
        }

        @Override
        public CharSequence getPageTitle(int position) {
            switch (position) {
                case 0:
                    return getResources().getText(R.string.home_tab);
                case 1:
                    return getResources().getText(R.string.pizza_tab);
                case 2:
                    return getResources().getText(R.string.pasta_tab);
                case 3:
                    return getResources().getText(R.string.store_tab);
            }
            return null;
        }
    }
}
```

This method adds the text to the tabs.

Test drive the app

When we run the app, `MainActivity` includes a tab layout. We can swipe through the fragments as before, and we can also navigate to each fragment by clicking on the appropriate tab.

That's all the code you need to implement tab and swipe navigation. So what's next?

The Design Support Library helps you implement material design

So far, we've added tabs to our app to help the user navigate around the app. To do this, we've used two components from the Design Support Library: the `TabLayout` and `AppBarLayout`.

The Design Support Library was introduced as a way of making it easier for developers to use **material design** components in their apps. Material design was introduced with Lollipop as a way of giving a consistent look and feel to all Android apps. The idea is that a user can switch from a Google app like the Play Store to an app designed by a third-party developer and instantly feel comfortable and know what to do. It's inspired by paper and ink, and uses print-based design principles and movement to reflect how real-world objects (such as index cards and pieces of paper) look and behave.

Material Design

You can find the full (and evolving) specs for material design here:

https://material.io/guidelines/

The Design Support Library lets you do more than add tabs to your apps.

- **It lets you add floating action buttons (FABs).**
  These are special action buttons that float above the main screen.

This is a FAB.

- **It includes snackbars, a way of displaying interactive short messages to the user as an alternative to toasts.**
  Unlike a toast (which you learned about in Chapter 5), you can add actions to snackbars so that the user can interact with them.

This is a snackbar. It's like a toast, but more interactive.

- **You can use it to animate your toolbars.**
  You can make your toolbar scroll off the screen, or collapse, if the user scrolls content in another view.

- **It includes a navigation drawer layout.**
  This is a slide-out drawer you can use as an alternative to using tabs. We'll look at this feature in Chapter 14.

For the rest of this chapter, we're going to show you how to implement some of these features in the Bits and Pizzas app.

Here's what we'll do

We're going to add more goodness from the Design Support Library to the Bits and Pizzas app. Here are the steps we'll go through:

1. **Enable MainActivity's toolbar to scroll.**
We'll change `MainActivity` so that the toolbar scrolls up and down when the user scrolls the contents of the view pager we added earlier. To see this working, we'll add content we can scroll to `TopFragment`.

When the user scrolls this content, the toolbar will scroll up too.

2. **Add a collapsing toolbar to OrderActivity.**
We'll start by adding a plain collapsing toolbar to `OrderActivity`. The toolbar will collapse when the user scrolls `OrderActivity`'s contents. After we've got the plain collapsing toolbar working, we'll add an image to it.

This is a toolbar with an image. When the user scrolls the main content, we'll get it to collapse.

3. **Add a FAB to OrderActivity.**
We'll display a floating action button to the bottom-right corner.

4. **Make the FAB display a snackbar.**
The snackbar will appear at the bottom of the screen when the user clicks on the FAB. The FAB will move up when the snackbar appears, and move back down when the snackbar is no longer there.

We'll start by getting our toolbar to scroll in response to the user scrolling the content in the view pager.

Make the toolbar respond to scrolls

Scrolling toolbar
Collapsing toolbar
FAB
Snackbar

We're going to change our app so that `MainActivity`'s toolbar scrolls whenever the user scrolls content in `TopFragment`. To enable this, there are two things we need to do:

1. **Change MainActivity's layout to enable the toolbar to scroll.**
2. **Change TopFragment to include scrollable content.**

We'll start by changing `MainActivity`'s layout.

We'll get the toolbar to scroll when the user scrolls the content in TopFragment.

We'll add this scrollable content to TopFragment.

Use a CoordinatorLayout to coordinate animations between views

To get the toolbar to move when content in the fragment is scrolled, we'll add a **coordinator layout** to `MainActivity`. A coordinator layout is like a souped-up frame layout that's used to coordinate animations and transitions between different views. In this case, we'll use the coordinator layout to coordinate scrollable content in `TopFragment` and `MainActivity`'s toolbar.

You add a coordinator layout to an activity's layout using code like this:

```
<android.support.design.widget.CoordinatorLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

</android.support.design.widget.CoordinatorLayout>
```

The CoordinatorLayout comes from the Design Support Library.

You add any views whose behavior you want to coordinate inside the CoordinatorLayout.

Any views in your layout whose animations you want to coordinate must be included in the `<CoordinatorLayout>` element. In our case, we want to coordinate animations between the toolbar and the contents of the view pager, so these views need to be included in the coordinator layout.

A CoordinatorLayout allows the behavior of one view to affect the behavior of another.

Add a coordinator layout to MainActivity's layout

We're going to replace the linear layout in *activity_main.xml* with a coordinator layout. Here's our code; update your version to match our changes (in bold):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize" />

        <android.support.design.widget.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</android.support.design.widget.CoordinatorLayout>
</LinearLayout>
```

We're replacing the LinearLayout with a CoordinatorLayout.

Add the app namespace, as we'll need to use attributes from it over the next few pages.

Delete this line, as we're no longer using a LinearLayout.

We're replacing the LinearLayout with a CoordinatorLayout.

How to coordinate scroll behavior

As well as adding views to the coordinator layout, you need to say how you want them to behave. In our case, we want the toolbar to scroll in response to another view's scroll event. This means that we need to ***mark the view the user will scroll***, and ***tell the toolbar to respond to it***.

Mark the view the user will scroll

You mark the view the user will scroll by giving it an attribute of **`app:layout_behavior`** and setting it to the built-in String `"@string/appbar_scrolling_view_behavior"`. This tells the coordinator layout that you want views in the app bar layout to be able to respond when the user scrolls this view. In our case, we want the toolbar to scroll in response to the user scrolling the view pager's content, so we need to add the `app:layout_behavior` attribute to the `ViewPager` element:

```
<android.support.v4.view.ViewPager
    ...
    app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

You add this line to the ViewPager to tell the CoordinatorLayout you want to react to the user scrolling its content.

Tell the toolbar to respond to scroll events

You tell views in the app bar layout how to respond to scroll events using the **`app:layout_scrollFlags`** attribute. In our case, we're going to set the toolbar to scroll upward off the screen when the user scrolls the view pager content up, and quickly return to its original position when the user scrolls down. To do this, we need to set the `Toolbar app:layout_scrollFlags` attribute to `"scroll|enterAlways"`.

The `scroll` value allows the view to scroll off the top of screen. Without this, the toolbar would stay pinned to the top of the screen. The `enterAlways` value means that the toolbar quickly scrolls down to its original position when the user scrolls the corresponding view. The toolbar will still scroll down without this value, but it will be slower.

Here's the code we need to add to the toolbar to enable it to scroll:

```
<android.support.v7.widget.Toolbar
    ...
    app:layout_scrollFlags="scroll|enterAlways" />
```

This line tells the CoordinatorLayout (and AppBarLayout) how you want the Toolbar to react to the user scrolling content.

The toolbar MUST be contained within an app bar layout in order for it to scroll. The app bar layout and coordinator layout work together to enable the toolbar to scroll.

We'll look at the full code for `MainActivity`'s layout on the next page.

The code to enable to toolbar to scroll

Here's our updated code for *activity_main.xml*. Update your version of the code to match our changes (in bold):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.hfad.bitsandpizzas.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:layout_scrollFlags="scroll|enterAlways" />

        <android.support.design.widget.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />
</android.support.design.widget.CoordinatorLayout>
```

Add this line to enable the Toolbar to scroll. If you wanted the TabLayout to scroll too, you'd add the code to that element as well.

This line marks the view whose content you expect the user to scroll.

Those are all the changes we need to make to `MainActivity`. Next we'll add scrollable content to `TopFragment`.

Add scrollable content to TopFragment

We're going to change `TopFragment`'s layout so that it contains scrollable content. We'll add an image of one of the Bits and Pizzas restaurants, along with some text describing the company ethos.

Here's what the new version of `TopFragment` will look like:

We'll change TopFragment to include an image and some text. We want the user to be able to scroll the entire contents of the fragment.

We'll start by adding the String and image resources to our project.

Add String and image resources

We'll add the String resources first. Open *strings.xml*, then add the following:

```
<string name="company_name">Bits and Pizzas</string>
<string name="restaurant_image">Restaurant image</string>
<string name="home_text">Since we opened our doors in 2017, Bits and Pizzas
    has built its reputation as one of America's best Italian-Digital
    restaurants. Some people believe eating out is about being with your
    friends and family. We believe that good food is best enjoyed while
    staring at your phone.</string>
```

Next we'll add the restaurant image to the *drawable-nodpi* folder. First, switch to the Project view of Android Studio's explorer and check whether the *app/src/main/res/drawable-nodpi* folder exists in your project. If it's not already there, highlight the *app/src/main/res* folder, go to the File menu, choose the New... option, then click on the option to create a new Android resource directory. When prompted, choose a resource type of "drawable", name it "drawable-nodpi" and click on OK.

Once you have a *drawable-nodpi* folder, download the file *restaurant.jpg* from *https://git.io/v9oet*, and add it to the *drawable-nodpi* folder.

You need to add this image to the drawable-nodpi folder.

Use a nested scroll view to make layout content scrollable

We'll allow the user to scroll the contents of `TopFragment` using a **nested scroll view**. This kind of view works just like a normal scroll view, except that it enables *nested scrolling*. This is important because the coordinator layout *only* listens for nested scroll events. If you use a normal scroll view in your layout, the toolbar won't scroll when the user scrolls the content.

Another view that enables nested scrolling is the recycler view. You'll find out how to use this in the next chapter.

You add a nested scroll view to your layout using code like this:

```
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    ...

</android.support.v4.widget.NestedScrollView >
```

The NestedScrollView comes from the Design Support Library.

You add any views you want the user to be able to scroll to the NestedScrollView.

You add any views you want the user to be able to scroll to the nested scroll view. If you just have one view, you can add this to the nested scroll view directly. If you want to scroll multiple views, however, these must be added to a separate layout inside the scroll view. This is because a nested scroll view can only have one direct child. As an example, here's how you'd add two text views to a nested scroll view with the help of a linear layout:

```
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        ... >

        <TextView
            ... />

        <TextView
            ... />
    </LinearLayout>

</android.support.v4.widget.NestedScrollView >
```

We're just using a LinearLayout as an example—it could be some other sort of layout instead. The key point is that the NestedScrollView can only have one direct child. If you want to put more than one view in the NestedScrollView, in this case two TextViews, you must put them in another layout first.

Next we'll update `TopFragment`'s layout so it uses a nested scroll view.

Scrolling toolbar
Collapsing toolbar
FAB
Snackbar

How we'll structure TopFragment's layout

We're going to add a restaurant image and some text to `TopFragment`'s layout. Before we write the code, here's a breakdown of how we'll structure it.

1. We want the entire fragment to be scrollable. This means we need to put all the views in a nested scroll view.
2. We'll use two text views for the Bits and Pizzas company name and text. We'll put these in a vertical linear layout with a white background.
3. We want to display the linear layout containing the two text views on top of the image. We'll do this by putting them both in a frame layout.

Putting this together, we'll use a nested scroll view for our layout, and this will contain a frame layout. The frame layout will include two elements, an image view and a linear layout. The linear layout will contain two text views to display the company name and ethos.

On the next page we'll show you the full code for *fragment_top.xml*. Once you've updated your code, we'll take the app for a test drive.

The full code for fragment_top.xml

Here's the full code for *fragment_top.xml*; update your code to match ours:

```
<android.support.v4.widget.NestedScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.hfad.bitsandpizzas.TopFragment">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <ImageView android:id="@+id/info_image"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:scaleType="centerCrop"
            android:src="@drawable/restaurant"
            android:contentDescription="@string/restaurant_image" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="40dp"
            android:layout_marginLeft="16dp"
            android:layout_marginRight="16dp"
            android:padding="16dp"
            android:background="#FFFFFF"
            android:orientation="vertical">

            <TextView
                android:textSize="32sp"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/company_name" />

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/home_text" />
        </LinearLayout>
    </FrameLayout>
</android.support.v4.widget.NestedScrollView>
```

We want the whole fragment to be able to scroll.

We're using a FrameLayout because we want to position the text on top of the image.

We're using a LinearLayout to contain the text. We're giving it a white background, and the margins will add space around the edges.

Test drive the app

- [x] **Scrolling toolbar**
- [ ] **Collapsing toolbar**
- [ ] **FAB**
- [ ] **Snackbar**

When we run the app, `TopFragment` displays the new layout. When we scroll the content, the toolbar scrolls too.

When you scroll the content up, the toolbar scrolls up too.

By allowing the toolbar to scroll, you free up more space for content. An added bonus is that you don't have to write any activity or fragment code to control the toolbar's behavior. All of the functionality came from using widgets from the Design Support Library.

Add a collapsing toolbar to OrderActivity

A variant of allowing your toolbar to scroll is the **collapsing toolbar**. A collapsing toolbar is one that starts off large, shrinks when the user scrolls the screen content up, and grows again when the user scrolls the screen content back down. You can even add an image to it, which disappears when the toolbar reaches its minimum height, and becomes visible again as the toolbar expands:

Over the next few pages, we're going to change `OrderActivity` to include a collapsing toolbar.

First add some String resources

Before we get started, we need to add some String resources to *strings.xml* that we'll use in `OrderActivity`'s layout. Open *strings.xml*, then add the following resources:

```
<string name="order_name_hint">Please enter your name</string>
<string name="order_details_hint">Please enter your order</string>
```

We'll start updating the layout on the next page.

Scrolling toolbar
Collapsing toolbar
FAB
Snackbar

How to create a plain collapsing toolbar

You add a collapsing toolbar to your activity's layout using the collapsing toolbar layout from the Design Support Library. In order for it to work, you need to add the collapsing toolbar layout to an app bar layout that's included within a coordinator layout. The collapsing toolbar layout should contain the toolbar you want to collapse.

As the collapsing toolbar needs to respond to scroll events in a separate view, you also need to add scrollable content to the coordinator layout, for example using a nested scroll view.

Here's an overview of how you need to structure your layout file in order to use a collapsing toolbar:

```
<android.support.design.widget.CoordinatorLayout
    ... >

    <android.support.design.widget.AppBarLayout
        ... >

        <android.support.design.widget.CollapsingToolbarLayout
            ... >

            <android.support.v7.widget.Toolbar
                ... />

        </android.support.design.widget.CollapsingToolbarLayout>
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.widget.NestedScrollView
        ...>

        ...

    </android.support.v4.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>
```

You add the CollapsingToolbarLayout to an AppBarLayout, which sits inside a CoordinatorLayout. The CollapsingToolbarLayout contains the Toolbar.

The scrollable content goes here.

In addition to structuring your layout in a particular way, there are some key attributes you need to use to get your collapsing toolbar to work properly. We'll look at these next.

Scrolling toolbar
Collapsing toolbar
FAB
Snackbar

Nested scroll view attributes

As before, you need to tell the coordinator layout which view you expect the user to scroll. You do this by setting the nested scroll view's `layout_behavior` attribute to `"@string/appbar_scrolling_view_behavior"`:

```
<android.support.v4.widget.NestedScrollView
    ...
    app:layout_behavior="@string/appbar_scrolling_view_behavior" >
```

This is the same as when we created a scrolling toolbar.

Collapsing toolbar layout attributes

You want the collapsing toolbar layout to collapse and expand in response to scroll events, so you need to set its `layout_scrollFlags` attribute to control this behavior. In our case, we want the collapsing toolbar layout to collapse until it's the size of a standard toolbar, so we'll set the attribute to a value of `"scroll|exitUntilCollapsed"`:

```
<android.support.design.widget.CollapsingToolbarLayout
    ...
    app:layout_scrollFlags="scroll|exitUntilCollapsed" >
```

This means we want the toolbar to collapse until it's done collapsing.

App bar layout attributes

As before, you apply a theme to your app bar layout to control the appearance of its contents. You also need to specify a height for the contents of the app bar layout. This is the maximum height the collapsing toolbar will be able to expand to. In our case, we'll apply a theme of `"@style/ThemeOverlay.AppCompat.Dark.ActionBar"` as before and set the height to 300dp:

```
<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >
```

This is the maximum height of the collapsing toolbar.

Toolbar attributes

If you have items on your toolbar such as an Up button, these may scroll off the screen as the toolbar collapses. You can prevent this from happening by setting the `layout_collapseMode` attribute to `"pin"`:

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    app:layout_collapseMode="pin" />
```

This pins anything that's on the toolbar, such as the Up button, to the top of the screen.

The full code to add a collapsing toolbar to activity_order.xml

Here's how to add a collapsing toolbar to `OrderActivity`'s layout. Replace your existing code for *activity_order.xml* with the code below:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/coordinator"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

        <android.support.design.widget.CollapsingToolbarLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_scrollFlags="scroll|exitUntilCollapsed" >

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin" />

        </android.support.design.widget.CollapsingToolbarLayout>
    </android.support.design.widget.AppBarLayout>
```

We've added an ID to the CoordinatorLayout, as we'll need it later in the chapter.

This is the CollapsingToolbarLayout. It needs to be within an AppBarLayout.

The CollapsingToolbarLayout contains a toolbar.

The code continues on the next page.

The activity_order.xml code (continued)

```
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding = "16dp" >

        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/order_name_hint" />

        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/order_details_hint" />

    </LinearLayout>

</android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>
```

The NestedScrollView contains the content we want the user to be able to scroll.

We're using a LinearLayout to position the scrollable content.

We're using EditTexts to add some content we can scroll to the layout.

Let's see what happens when we run the app.

Test drive the app

- [x] Scrolling toolbar
- [ ] Collapsing toolbar
- [ ] FAB
- [ ] Snackbar

When we run the app, `OrderActivity` displays the new layout, including the collapsing toolbar. The collapsing toolbar starts off large, and collapses as we scroll the content.

This is the toolbar when it's collapsed.

This is the toolbar when it's expanded.

You can add images to collapsing toolbars too

The collapsing toolbar we've created so far is quite plain. It has a plain background, which grows and shrinks as we scroll the content in the activity.

We can improve this by adding an image to the collapsing toolbar. We'll display the image when the collapsing toolbar is large, and when it shrinks we'll display a standard toolbar instead:

This is the same collapsing toolbar, except we're going to add an image to it.

How to add an image to a collapsing toolbar

We're going to update the collapsing toolbar we just created so that it includes an image. For convenience, we'll use the same image that we added to `TopFragment`.

You add an image to a collapsing toolbar by adding an `ImageView` to the `CollapsingToolBarLayout`, specifying the image you want to use. As an optional extra, you can add a parallax effect to the `ImageView` so that the image scrolls at a different rate than the rest of the toolbar. You do this by adding a `layout_collapseMode` attribute to the `ImageView` with a value of `"parallax"`.

We want to use a drawable named "restaurant" for our image. Here's the code we need:

```
<android.support.design.widget.CollapsingToolbarLayout
... >

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:src="@drawable/restaurant"
        android:contentDescription="@string/restaurant_image"
        app:layout_collapseMode="parallax" />

    <Toolbar
        ... >

</android.support.design.widget.CollapsingToolbarLayout>
```

We're cropping the image to fit inside the AppBarLayout.

This line is optional. It adds parallax animation so the image moves at a different rate than the scrollable content.

By default, when the toolbar is collapsed, it will continue to display the image as its background. To get the toolbar to revert to a plain background color when it's collapsed, you add a `contentScrim` attribute to the `CollapsingToolbarLayout`, setting it to the value of the color. We want our toolbar to have the same background color as before, so we'll set it to `"?attr/colorPrimary"`:

```
<android.support.design.widget.CollapsingToolbarLayout
    ...
    app:layout_scrollFlags="scroll|exitUntilCollapsed"
    app:contentScrim="?attr/colorPrimary" >
```

This line changes the toolbar back to its default color when it's collapsed.

Those are all the changes we need, so we'll update the code on the next page, and then take it for a test drive.

The updated code for activity_order.xml

Here's the updated code for *activity_order.xml* to add an image to the collapsing toolbar (update your version to match our changes in bold):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/coordinator"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

        <android.support.design.widget.CollapsingToolbarLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_scrollFlags="scroll|exitUntilCollapsed"
            app:contentScrim="?attr/colorPrimary" >

            <ImageView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="centerCrop"
                android:src="@drawable/restaurant"
                android:contentDescription="@string/restaurant_image"
                app:layout_collapseMode="parallax" />

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin" />

        </android.support.design.widget.CollapsingToolbarLayout>
    </android.support.design.widget.AppBarLayout>
```

This line changes the toolbar background when it's collapsed.

These lines add an image to the collapsing toolbar. It uses snazzy parallax animation.

The code continues on the next page.

The activity_order.xml code (continued)

```
    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" >

        ...

    </android.support.v4.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>
```

Let's see what the app looks like when we run it.

Test drive the app

When we run the app, `OrderActivity`'s collapsing toolbar includes an image. As the toolbar collapses, the image fades, and the toolbar's background changes to its original color. When the toolbar is expanded again, the image reappears.

The toolbar changes color when it's collapsed.

The image appears on the toolbar.

FABs and snackbars

Scrolling toolbar
Collapsing toolbar
FAB
Snackbar

There are two final additions we're going to make to `OrderActivity` from the Design Support Library: a FAB and a snackbar.

A **FAB** is a **floating action button**. It's a circled icon that floats above the user interface, for example in the bottom-right corner of the screen. It's used to promote actions that are so common or important that you want to make them obvious to the user.

This is a FAB that appears in the Google Calendar app. It floats in the bottom-right corner of the screen, and you use it to add events.

A **snackbar** is like a toast except that you can interact with it. It's a short message that appears at the bottom of the screen that's used to give the user information about an operation. Unlike with a toast, you can add actions to a snackbar, such as an action to undo an operation.

This is the snackbar that appears in the Chrome app when you've just closed a web page. You can reopen the page by clicking on the Undo action in the snackbar.

We'll add a FAB and snackbar to OrderActivity

We're going to add a FAB to `OrderActivity`. When the user clicks on the FAB, we'll display a snackbar that shows a message to the user. In the real world, you'd want to use the FAB to perform an action such as saving the user's pizza order, but we're just going to focus on showing you how to add the widgets to your app.

Here's what the new version of `OrderActivity` will look like:

Scrolling toolbar
Collapsing toolbar
FAB
Snackbar

Add the icon for the FAB

We'll start by adding an icon to our project to display on the FAB. You can either create your own icon from scratch or use one of the icons provided by Google: *https://design.google.com/icons/*.

We're going to use the "done" icon `ic_done_white_24dp`, and we'll add a version of it to our project's *drawable** folders, one for each screen density. Android will decide at runtime which version of the icon to use depending on the screen density of the device.

We're going to use this icon on our FAB.

First, switch to the Project view of Android Studio's explorer, highlight the *app/src/main/res* folder in your project, then create folders called *drawable-hdpi*, *drawable-mdpi*, *drawable-xhdpi*, *drawable-xxhdpi*, and *drawable-xxxhdpi* if they don't already exist. Then go to *http://tinyurl.com/HeadFirstAndroidDoneIcons*, and download *ic_done_white_24dp.png* Bits and Pizzas images. Add the image in the *drawable-hdpi* folder to the *drawable-hdpi* folder in your project, then repeat this process for the other folders.

How to add a FAB to your layout

You add a FAB to your layout using code like this:

```
<android.support.design.widget.CoordinatorLayout ...>

    ...

    <android.support.design.widget.FloatingActionButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end|bottom"
        android:layout_margin="16dp"
        android:src="@drawable/ic_done_white_24dp"
        android:onClick="onClickDone" />

</android.support.design.widget.CoordinatorLayout>
```

The code for adding a FAB is similar to the code for adding an ImageButton. That's because FloatingActionButton is a subclass of ImageButton.

If you're using the FAB in an activity, you can use the onClick attribute to specify which method should be called when it's clicked.

The above code adds a FAB to the bottom-end corner of the screen, with a margin of 16dp. It uses the `src` attribute to set the FAB's icon to the `ic_done_white_24dp` drawable. We're also using the FAB's `onClick` attribute to specify that the `onClickDone()` method in the layout's activity will get called when the user clicks on the FAB. We'll create this method later.

You usually use a FAB inside a `CoordinatorLayout`, as this means that you can coordinate movement between the different views in your layout. In our case, it means that the FAB will move up when the snackbar appears.

On the next page we'll show you the code for `OrderActivity`'s layout.

The material design guidelines recommend using no more than one FAB per screen.

The updated code for activity_order.xml

Scrolling toolbar
Collapsing toolbar
FAB
Snackbar

Here's the updated code for *activity_order.xml* (update your version to match our changes in bold):

We've not changed any of the code on this page.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/coordinator"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

        <android.support.design.widget.CollapsingToolbarLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_scrollFlags="scroll|exitUntilCollapsed"
            app:contentScrim="?attr/colorPrimary" >

            <ImageView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="centerCrop"
                android:src="@drawable/restaurant"
                android:contentDescription="@string/restaurant_image"
                app:layout_collapseMode="parallax" />

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin" />

        </android.support.design.widget.CollapsingToolbarLayout>
    </android.support.design.widget.AppBarLayout>
```

The code continues on the next page.

The activity_order.xml code (continued)

```
<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior" >

    <LinearLayout
        ...
    </LinearLayout>

</android.support.v4.widget.NestedScrollView>

<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    android:layout_margin="16dp"
    android:src="@drawable/ic_done_white_24dp"
    android:onClick="onClickDone" />

</android.support.design.widget.CoordinatorLayout>
```

We're adding the FAB to the CoordinatorLayout so that it will move out the way when a snackbar gets displayed.

Add the onClickDone() method to OrderActivity

Now that we've added a FAB to `OrderActivity`'s layout, we need to write some activity code to make the FAB do something when it's clicked. You do this in the same way that you would for a button, by adding the method described by the FAB's `onClick` attribute to your activity code.

In our case, we've given the `onClick` attribute a value of `"onClickDone"`, so this means we need to add an `onClickDone()` method to *OrderActivity.java*:

```
public void onClickDone(View view) {
    //Code that runs when the FAB is clicked
}
```

You don't have to add this method now; you can wait until we show you the full code a few pages ahead.

Now we're going to write some code to display a snackbar when the user clicks on the FAB.

How to create a snackbar

- [x] Scrolling toolbar
- [x] Collapsing toolbar
- [x] FAB
- [ ] **Snackbar**

As we said earlier in the chapter, a snackbar is a bar that appears at the bottom of the screen that displays a short message to the user. It's similar to a toast, except that you can interact with it.

To create a snackbar, you call the `Snackbar.make()` method. This method takes three parameters: the `View` you want to hold the snackbar, the text you want to display, and an `int` duration. As an example, here's the code for a snackbar that appears on the screen for a short duration:

```
CharSequence text = "Hello, I'm a Snackbar!";
int duration = Snackbar.LENGTH_SHORT;
Snackbar snackbar = Snackbar.make(findViewById(R.id.coordinator, text, duration);
```

If you want to display a String resource, you can pass in the resource ID instead of the text.

In the above code, we've used a view called `coordinator` to hold the snackbar. This view will usually be your activity's coordinator layout so that it can coordinate the snackbar with other views.

We've set the snackbar's duration to `LENGTH_SHORT`, which shows the snackbar for a short period of time. Other options are `LENGTH_LONG` (which shows it for a long duration) and `LENGTH_INDEFINITE` (which shows it indefinitely). With any of these options, the user is able to swipe away the snackbar so that it's no longer displayed.

You can add an action to the snackbar by calling its `setAction()` method. This can be useful if, for example, you want the user to be able to undo an operation they've just performed. The `setAction()` method takes two parameters: the text that should appear for the action, and a `View.onClickListener()`. Any code you want to run when the user clicks on the action should appear in the listener's `onClick()` event:

You pass the setAction() method the text that should appear for the action, and a View.OnClickListener.

```
snackbar.setAction("Undo", new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Code to run when the user clicks on the Undo action
    }
});
```

You need to specify what should happen if the user clicks on the Undo action.

Once you've finished creating the snackbar, you display it using its `show()` method:

```
snackbar.show();
```

Pool Puzzle

Your **goal** is to make `OrderActivity`'s `onClickDone()` method display a snackbar. The snackbar should include an action, "Undo", which shows a toast when clicked. Take code snippets from the pool and place them into the blank lines in the code. You may **not** use the same snippet more than once, and you won't need to use all the snippets.

```
public void onClickDone(View view) {
    CharSequence text = "Your order has been updated";
    int duration =                                    ;
    Snackbar snackbar = Snackbar.              (findViewById(R.id.coordinator),              ,              );
    snackbar.setAction("Undo", new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Toast toast = Toast.              OrderActivity.this, "Undone!",                              );
            toast.              ;
        }
    });
    snackbar.              ;
}
```

Note: each thing from the pool can only be used once!

Pool Puzzle Solution

Your **goal** is to make `OrderActivity`'s `onClickDone()` method display a snackbar. The snackbar should include an action, "Undo", which shows a toast when clicked. Take code snippets from the pool and place them into the blank lines in the code. You may **not** use the same snippet more than once, and you won't need to use all the snippets.

```
public void onClickDone(View view) {
    CharSequence text = "Your order has been updated";
    int duration = Snackbar . LENGTH_SHORT ;
    Snackbar snackbar = Snackbar.make (findViewById(R.id.coordinator), text , duration );
    snackbar.setAction("Undo", new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Toast toast = Toast.makeText OrderActivity.this, "Undone!",Toast . LENGTH_SHORT );
            toast. show() ;
        }
    });
    snackbar. show() ;
}
```

The full code for OrderActivity.java

Here's our full code for *OrderActivity.java*, including the code to add a snackbar with an action. Update your version of the code to match our changes (in bold):

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.support.v7.app.ActionBar;
import android.view.View;
import android.support.design.widget.Snackbar;
import android.widget.Toast;

public class OrderActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_order);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }

    public void onClickDone(View view) {
        CharSequence text = "Your order has been updated";
        int duration = Snackbar.LENGTH_SHORT;
        Snackbar snackbar = Snackbar.make(findViewById(R.id.coordinator), text, duration);
        snackbar.setAction("Undo", new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast toast = Toast.makeText(OrderActivity.this, "Undone!", Toast.LENGTH_SHORT);
                toast.show();
            }
        });
        snackbar.show();
    }
}
```

We're using these new classes, so you need to import them.

This method gets called when the user clicks on the FAB.

Create the snackbar.

Add an action to the snackbar.

If the user clicks on the snackbar's action, display a toast.

Display the snackbar.

test drive

Test drive the app

- Scrolling toolbar
- Collapsing toolbar
- FAB
- **Snackbar**

When we run the app, a FAB is displayed in `OrderActivity`. When we click on the FAB, a snackbar is displayed and the FAB moves up to accommodate it. When we click on the Undo action on the snackbar, a toast is displayed.

As you can see, snackbars have a lot in common with toasts, as they're both used to display messages to the user. But if you want the user to be able to *interact* with the information you're showing them, choose a snackbar.

Your Android Toolbox

You've got Chapter 12 under your belt and now you've added the Design Support Library to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Enable swipe navigation using a **view pager**.
- You tell a view pager about its pages by implementing a **fragment pager adapter**.
- Use the fragment pager adapter's `getCount()` method to tell the view pager how many pages it should have. Use its `getItem()` method to tell it which fragment should appear on each page.
- Add tab navigation by implementing a **tab layout**. Put the toolbar and tab layout inside an **app bar layout** in your layout code, then attach the tab layout to the view pager in your activity code.
- The tab layout comes from the **Android Design Support Library**. This library helps you implement the **material design guidelines** in your app.
- Use a **coordinator layout** to coordinate animations between views.
- Add scrollable content the coordinator layout can coordinate using a **nested scroll view**.
- Use a **collapsing toolbar layout** to add a toolbar that collapses and grows in response to user scroll actions.
- Use a **FAB** (floating action button) to promote common or important user actions.
- A **snackbar** lets you display short messages that the user can interact with.

13 recycler views and card views

Get Recycling

You've already seen how the humble list view is a key part of most apps. But compared to some of the *material design* components we've seen, it's somewhat plain. In this chapter, we'll introduce you to the **recycler view**, a more advanced type of list that gives you *loads more flexibility* and *fits in with the material design ethos*. You'll see how to create **adapters** tailored to your data, and how to completely change the look of your list with *just two lines of code*. We'll also show you how to use **card views** to give your data a *3D material design* appearance.

There's still work to do on the Bits and Pizzas app

In Chapter 12, we updated the Bits and Pizzas app to include components from the Design Support Library, including a tab layout, FAB, and collapsing toolbar. We added these to help users navigate to places in the app more easily, and to implement a consistent material design look and feel. If you recall, material design is inspired by paper and ink, and uses print-based design principles and movement to reflect how real-world objects (such as index cards and pieces of paper) look and behave. But there's one key area we didn't look at: lists.

We're currently using list views in `PizzaFragment`, `PastaFragment`, and `StoresFragment` to display the available pizzas, pasta, and stores. These lists are very plain compared with the rest of the app, and could do with some work to give them the same look and feel.

Another disadvantage of list views is that they don't implement nested scrolling. In Chapter 12, we made `MainActivity`'s toolbar scroll in response to the user scrolling content in the activity's fragments. This currently works for `TopFragment`, as it uses a nested scroll view. As none of the other fragments use nested scrolling, however, the toolbar remains fixed when the user tries to scroll their content.

To address these issues, we're going to change `PizzaFragment` to use a **recycler view**. This is a more advanced and flexible version of a list view that implements nested scrolling. Instead of displaying just the names of each pizza in a list view, we'll use a recycler view to display its name and image:

This is the current PizzaFragment. It includes a list of pizzas, but it looks quite plain.

This is the recycler view we're going to add to PizzaFragment.

When you scroll the recycler view, the toolbar moves up. This matches the behavior of TopFragment.

Recycler views from 10,000 feet

Before we dive into the code, let's take a look at how recycler views work. As a recycler view is more flexible than a list view, it takes a lot more setting up.

Like list views, recycler views efficiently manage a small number of views to give the appearance of a large collection of views that extend beyond the screen. They allow you greater flexibility about how the data is displayed than list views do.

A recycler view accesses its data using an **adapter**. Unlike a list view, however, it doesn't use any of the built-in Android adapters such as array adapters. Instead, ***you have to write an adapter of your own that's tailored to your data***. This includes specifying the type of data, creating views, and binding the data to the views.

Items are positioned in a recycler view using a **layout manager**. There are a number of built-in layout managers you can use that allow you to position items in a linear list or grid.

Here's a diagram of all those elements put together:

In our particular case, we're going to create a recycler view to display pizza names and images. We'll go through the steps for how to do this on the next page.

Here's what we're going to do

There are five main steps we'll go through to get the recycler view working:

1. **Add the pizza data to the project.**
We'll add images of the pizzas to the app, along with a new `Pizza` class. This class will be the recycler view's data source.

2. **Create a card view for the pizza data.**
We're going to make each pizza in the recycler view look as though it's displayed on a separate card. To do this, we'll use a new type of view called a *card view*.

3. **Create a recycler view adapter.**
As we said on the previous page, when you use a recycler view you need to write your own adapter for it. Our adapter needs to take the pizza data and bind each item to a card view. Each card will then be able to be displayed in the recycler view.

4. **Add a recycler view to PizzaFragment.**
After we've created the adapter, we'll add the recycler view to `PizzaFragment`. We'll make it use the adapter, and use a layout manager to display pizza data in a two-column grid.

5. **Make the recycler view respond to clicks.**
We'll create a new activity, `PizzaDetailActivity`, and get it to start when the user clicks on one of the pizzas. We'll display details of the pizza in the activity.

These are card views displaying pizza data.

This is the recycler view.

This is PizzaDetailActivity.

Do this!

We're going to update the Bits and Pizzas app in this chapter, so open your Bits and Pizzas project in Android Studio.

The first thing we'll do is add the pizza data.

Add the pizza data

We'll start by adding the pizza images to the Bits and Pizzas project. Download the files *diavolo.jpg* and *funghi.jpg* from *https://git.io/v9oet*, then add them to the folder *app/src/main/res/drawable-nodpi*. This folder should already exist in your project, as we added an image to it in Chapter 12.

These are the pizza images.

Add the Pizza class

We'll get our data from a `Pizza` class, which we need to add. The class defines an array of two pizzas, where each pizza has a name and image resource ID. Switch to the Project view of Android Studio's explorer, highlight the `com.hfad.bitsandpizzas` package in the *app/src/main/java* folder, then go to File→New...→Java class. When prompted, name the class "Pizza" and make sure the package name is `com.hfad.bitsandpizzas`. Finally, replace the code in *Pizza.java* with the following:

In a real app, we might use a database for this. We're using a Java class here for simplicity.

```
package com.hfad.bitsandpizzas;

public class Pizza {
    private String name;
    private int imageResourceId;

    public static final Pizza[] pizzas = {
            new Pizza("Diavolo", R.drawable.diavolo),
            new Pizza("Funghi", R.drawable.funghi)
    };

    private Pizza(String name, int imageResourceId) {
        this.name = name;
        this.imageResourceId = imageResourceId;
    }

    public String getName() {
        return name;
    }

    public int getImageResourceId() {
        return imageResourceId;
    }
}
```

Each Pizza has a name and image resource ID. The image resource ID refers to the pizza images we added to the project above.

The Pizza constructor

These are getters for the private variables.

Display the pizza data in a card

The next thing we need to do is define a layout for the pizza data. This layout will be used by the recycler view's adapter to determine how each item in the recycler view should look. We're going to use a **card view** for this layout.

A card view is a type of frame layout that lets you display information on virtual cards. Card views can have rounded corners and shadows to make it look as though they're positioned above their background. If we use a card view for our pizza data, each pizza will look as though it's displayed on a separate card in the recycler view.

Add the CardView and RecyclerView Support Libraries

Card views and recycler views come from the CardView and RecyclerView v7 Support Libraries, respectively, so before we can go any further, you need to add them to your project as dependencies.

In Android Studio go to File→Project Structure. In the Project Structure window, click on the "app" option and switch to the Dependencies tab. Then click on the "+" button at the bottom or right side of the screen, choose the "Library dependency" option, and add the CardView Library. Repeat these steps to add the RecyclerView-v7 Library, then click on the OK button to save your changes.

Make sure you add both libraries.

Now that you've added the Support Libraries, we'll create a card view that we can use for our pizza data.

How to create a card view

We're going to create a card view that displays an image with a caption. We'll use it here for the name and image of individual pizzas, but you could also use the same layout for different categories of data such as pasta or stores.

You create a card view by adding a `<CardView>` element to a layout. If you want to use the card view in a recycler view (as we do here), you need to create a new layout file for the card view. Do this by highlighting the *app/src/main/res/layout* folder, and choosing File→New→Layout resource file. When prompted, name the layout "card_captioned_image".

You add a card view to your layout using code like this:

```
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_margin="4dp"
    card_view:cardElevation="2dp"
    card_view:cardCornerRadius="4dp">

    ...

</android.support.v7.widget.CardView>
```

This defines the CardView.

Setting the card's elevation gives it a drop shadow.

This gives the CardView rounded corners.

You add any views you want to be displayed to the CardView.

In the above code, we've added an extra namespace of:

```
xmlns:card_view="http://schemas.android.com/apk/res-auto"
```

so that we can give the card rounded corners and a drop shadow to make it look higher than its background. You add rounded corners using the `card_view:cardCornerRadius` attribute, and the `card_view:cardElevation` attribute sets its elevation and adds drop shadows.

Once you've defined the card view, you need to add any views you want to display to it. In our case, we'll add a text view and image view to display the name and image of the pizza. We'll show you the full code for this on the next page.

The full card_captioned_image.xml code

Here's the full code for *card_captioned_image.xml* (update your version of the file to match ours):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_margin="5dp"
    card_view:cardElevation="2dp"
    card_view:cardCornerRadius="4dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <ImageView android:id="@+id/info_image"
            android:layout_height="0dp"
            android:layout_width="match_parent"
            android:layout_weight="1.0"
            android:scaleType="centerCrop"/>

        <TextView
            android:id="@+id/info_text"
            android:layout_marginLeft="4dp"
            android:layout_marginBottom="4dp"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"/>
    </LinearLayout>
</android.support.v7.widget.CardView>
```

The card view will be as wide as its parent allows, and 200dp high.

We've put the ImageView and TextView in a LinearLayout, as the CardView can only have one direct child.

The image will be as wide as the CardView allows. We're using centerCrop to make sure the image scales uniformly.

The CardView contains an ImageView and a TextView.

This is what the CardView will look like when data's been added to it. We'll do this via the recycler view's adapter.

Note that the above layout doesn't explicitly mention pizza data. This means we can use the same layout for any data items that consist of a caption and an image, such as pasta.

Now that we've created a layout for the card views, we'll move on to creating the recycler view's adapter.

How our recycler view adapter will work

As we said earlier, when you use a recycler view in your app, you need to create a recycler view adapter. That's because unlike a list view, recycler views don't use any of the built-in adapters that come with Android. While writing your own adapter may seem like hard work, on the plus side it gives you more flexibility than using a built-in one.

The adapter has two main jobs: to create each of the views that are visible within the recycler view, and to bind each view to a piece of data. In our case, the recycler view needs to display a set of cards, each containing a pizza image and caption. This means that the adapter needs to create each card and bind data to it.

We'll create the recycler view adapter over the next few pages. Here are the steps we'll go through to create it:

1. **Specify what data the adapter should work with.**
   We want the adapter to work with the pizza data. Each pizza has a name and image resource ID, so we'll pass the adapter an array of pizza names, and an array of image resource IDs.

2. **Define the views the adapter should populate.**
   We want to use the data to populate a set of pizza cards defined by *card_captioned_image.xml*. We then need to create a set of these cards that will be displayed in the recycler view, one card per pizza.

3. **Bind the data to the cards.**
   Finally, we need to display the pizza data in the cards. To make that happen, we need to populate the `info_text` text view with the name of the pizza, and the `info_image` image view with the pizza's image.

We'll start by adding a `RecyclerView.Adapter` class to our project.

there are no Dumb Questions

Q: Why doesn't Android provide ready-made adapters for recycler views?

A: Because recycler view adapters don't just specify the data that will appear. They also specify the views that will be used for each item in the collection. That means that recycler view adapters are both more powerful, and less general, than list view adapters.

Add a recycler view adapter

You create a recycler view adapter by extending the `RecyclerView.Adapter` class and overriding various methods; we'll cover these over the next few pages. You also need to define a `ViewHolder` as an inner class, which tells the adapter which views to use for the data items.

We're going to create a recycler view adapter called `CaptionedImagesAdapter`. In Android Studio, highlight the `com.hfad.bitsandpizzas` package in the *app/src/main/java* folder, then go to File→New...→Java class. When prompted, name the class "CaptionedImagesAdapter" and make sure the package name is `com.hfad.bitsandpizzas`. Then replace the code in *CaptionedImagesAdapter.java* with the following:

```
package com.hfad.bitsandpizzas;

import android.support.v7.widget.RecyclerView;

class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{

    public static class ViewHolder extends RecyclerView.ViewHolder {
        //Define the view to be used for each data item
    }
}
```

We're extending the RecyclerView class, so we need to import it.

The ViewHolder is used to specify which views should be used for each data item.

You define the ViewHolder as an inner class. We'll complete this later in the chapter.

As you can see, the `ViewHolder` inner class you define is a key part of the adapter. We've left the `ViewHolder` class empty for now, but we'll come back to it later in the chapter.

Before we look in more detail at view holders, we'll tell the adapter what sort of data it should use by adding a constructor.

Don't worry if Android Studio gives you error messages when you add the above code to your project.

It's just warning you that the code isn't complete yet. We still need to override various methods in our adapter code to tell it how to behave, and we'll do this over the next few pages.

Tell the adapter what data it should work with...

When you define a recycler view adapter, you need to tell it what sort of data it should use. You do this by defining a constructor that includes the data types you want the adapter to use as parameters.

In our case, we want the adapter to take String captions and `int` image IDs. We'll therefore add `String[]` and `int[]` parameters to the constructor, and save the arrays as private variables. Here's the code that does this; you can either update your version of *CaptionedImagesAdapter.java* now, or wait until we show you the full adapter code later in the chapter.

```
class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{

    private String[] captions;
    private int[] imageIds;
...
    public CaptionedImagesAdapter(String[] captions, int[] imageIds){
        this.captions = captions;
        this.imageIds = imageIds;
    }
}
```

We'll use these variables to hold the pizza data.

We'll pass the data to the adapter using its constructor.

...and implement the getItemCount() method

We also need to tell the adapter how many data items there are. You do this by overriding the `RecyclerViewAdapter getItemCount()` method. This returns an `int` value, the number of data items. We can derive this from the number of captions we pass the adapter. Here's the code:

```
class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
...
    @Override
    public int getItemCount(){
        return captions.length;
    }
}
```

The length of the captions array equals the number of data items in the recycler view.

Next we'll define the adapter's view holder.

Define the adapter's view holder

The view holder is used to define what view or views the recycler view should use for each data item it's given. You can think of it as a holder for the views you want the recycler view to display. In addition to views, the view holder contains extra information that's useful to the recycler view, such as its position in the layout.

In our case, we want to display each item of pizza data on a card, which means we need to specify that the adapter's view holder uses a card view. Here's the code to do this (we'll show you the full adapter code later in the chapter):

Each of our ViewHolders will contain a CardView. We created the layout for this CardView earlier in the chapter.

```
...
import android.support.v7.widget.CardView;

class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
...
    public static class ViewHolder extends RecyclerView.ViewHolder {

        private CardView cardView;

        public ViewHolder(CardView v) {
            super(v);
            cardView = v;
        }
    }
}
```

We're using the CardView class, so we need to import it.

Our recycler view needs to display CardViews, so we specify that our ViewHolder contains CardViews. If you want to display another type of data in the recycler view, you define it here.

When you create a view holder, you must call the `ViewHolder` super constructor using:

```
super(v);
```

This is because the `ViewHolder` superclass includes metadata such as the item's position in the recycler view, and you need this information for the adapter to work properly.

Now that we've defined our view holders, we need to tell the adapter how to construct one. We'll do this by overriding the adapter's `onCreateViewHolder()` method.

Override the onCreateViewHolder() method

The `onCreateViewHolder()` method gets called when the recycler view requires a new view holder. The recycler view calls the method repeatedly when the recycler view is first constructed to build the set of view holders that will be displayed on the screen.

The method takes two parameters: a `ViewGroup` parent object (the recycler view itself) and an `int` parameter called `viewType`, which is used if you want to display different kinds of views for different items in the list. It returns a view holder object. Here's what the method looks like:

```
@Override
public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(
        ViewGroup parent, int viewType){
    //Code to instantiate the ViewHolder
}
```

This method gets called when the recycler view needs to create a view holder.

We need to add code to the method to instantiate the view holder. To do this, we need to call the `ViewHolder`'s constructor, which we defined on the previous page. The constructor takes one parameter, a `CardView`. We'll create the `CardView` from the *card_captioned_image.xml* layout we created earlier in the chapter using this code:

Get a LayoutInflater object.

```
CardView cv = (CardView) LayoutInflater.from(parent.getContext())
        .inflate(R.layout.card_captioned_image, parent, false);
```

Use the LayoutInflator to turn the layout into a CardView. This is nearly identical to code you've already seen in the onCreateView() of fragments.

Here's the full code for the `onCreateViewHolder()` method (we'll add this to the adapter later):

```
@Override
public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(
                ViewGroup parent, int viewType){
    CardView cv = (CardView) LayoutInflater.from(parent.getContext())
            .inflate(R.layout.card_captioned_image, parent, false);
    return new ViewHolder(cv);
}
```

Specify what layout to use for the contents of the ViewHolder.

Now that the adapter can create view holders, we need to get it to populate the card views they contain with data.

Add the data to the card views

You add data to the card views by implementing the adapter's `onBindViewHolder()` method. This gets called whenever the recycler view needs to display data in a view holder. It takes two parameters: the view holder the data needs to be bound to, and the position in the data set of the data that needs to be bound.

Our card view contains two views, an image view with an ID of `info_image`, and a text view with an ID of `info_text`. We'll populate these with data from the `captions` and `imageIds` arrays. Here's the code that will do that:

Each CardView contains a TextView and ImageView. We need to populate these with the caption and image of each pizza.

We're using these extra classes, so we need to import them.

```
...
import android.widget.ImageView;
import android.widget.TextView;
import android.graphics.drawable.Drawable;
import android.support.v4.content.ContextCompat;

class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{

    private String[] captions;
    private int[] imageIds;

    ...

    @Override
    public void onBindViewHolder(ViewHolder holder, int position){
        CardView cardView = holder.cardView;
        ImageView imageView = (ImageView)cardView.findViewById(R.id.info_image);
        Drawable drawable =
                ContextCompat.getDrawable(cardView.getContext(), imageIds[position]);
        imageView.setImageDrawable(drawable);
        imageView.setContentDescription(captions[position]);
        TextView textView = (TextView)cardView.findViewById(R.id.info_text);
        textView.setText(captions[position]);
    }
}
```

We added these variables earlier. They contain the captions and image resource IDs of the pizzas.

The recycler view calls this method when it wants to use (or reuse) a view holder for a new piece of data.

Display the image in the ImageView.

Display the caption in the TextView.

That's all the code we need for our adapter. We'll show you the full code over the next couple of pages.

The full code for CaptionedImagesAdapter.java

Here's our complete code for the adapter. Update your version of *CaptionedImagesAdapter.java* so that it matches ours.

```
package com.hfad.bitsandpizzas;

import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.CardView;
import android.view.ViewGroup;
import android.view.LayoutInflater;
import android.widget.ImageView;
import android.widget.TextView;
import android.graphics.drawable.Drawable;
import android.support.v4.content.ContextCompat;

class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{

    private String[] captions;
    private int[] imageIds;

    public static class ViewHolder extends RecyclerView.ViewHolder {

        private CardView cardView;

        public ViewHolder(CardView v) {
            super(v);
            cardView = v;
        }
    }

    public CaptionedImagesAdapter(String[] captions, int[] imageIds){
        this.captions = captions;
        this.imageIds = imageIds;
    }
```

These are the classes we're using, so we need to import them.

We're using these variables for the captions and image resource IDs.

Each ViewHolder will display a CardView.

Pass data to the adapter in its constructor.

The code continues on the next page.

The full CaptionedImagesAdapter.java code (continued)

```
    @Override
    public int getItemCount(){
        return captions.length;
    }

    @Override
    public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(
            ViewGroup parent, int viewType){
        CardView cv = (CardView) LayoutInflater.from(parent.getContext())
                .inflate(R.layout.card_captioned_image, parent, false);
        return new ViewHolder(cv);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position){
        CardView cardView = holder.cardView;
        ImageView imageView = (ImageView)cardView.findViewById(R.id.info_image);
        Drawable drawable =
                 ContextCompat.getDrawable(cardView.getContext(), imageIds[position]);
        imageView.setImageDrawable(drawable);
        imageView.setContentDescription(captions[position]);
        TextView textView = (TextView)cardView.findViewById(R.id.info_text);
        textView.setText(captions[position]);
    }
}
```

The number of data items

Use the layout we created earlier for the CardViews.

Populate the CardView's ImageView and TextView with data.

That's all the code we need for our adapter. So what's next?

Create the recycler view

So far we've created a card view layout that displays captioned images, and an adapter that creates the cards and populates them with data. The next thing we need to do is create the recycler view, which will pass pizza data to the adapter so that it can populate the cards with the pizza images and captions. The recycler view will then display the cards.

We're going to add the recycler view to our existing `PizzaFragment`. Whenever the user clicks on the Pizzas tab in `MainActivity`, the pizzas will be displayed:

This is what the recycler view in PizzaFragment will look like. It will display the pizza cards in a two-column grid.

Add a layout for PizzaFragment

Before we can add the recycler view, we need to add a new layout file to our project for `PizzaFragment` to use. This is because we intially created `PizzaFragment` as a `ListFragment`, and these define their own layout.

To add the layout file, highlight the *app/src/main/res/layout* folder in Android Studio, and choose File→New→Layout resource file. When prompted, name the layout "fragment_pizza".

Add the RecyclerView to PizzaFragment's layout

You add a recycler view to a layout using the `<RecyclerView>` element from the RecyclerView Support Library.

Our `PizzaFragment` layout only needs to display a single recycler view, so here's the full code for *fragment_pizza.xml* (update your version of the code to match ours):

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pizza_recycler"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical" />
```

This defines the RecyclerView.

We've given the recycler view an ID so that we can refer to it in our Java code.

This adds a vertical scrollbar.

You add scrollbars to the recycler view using the `android:scrollbars` attribute. We've set this to `"vertical"` because we want our recycler view to be able to scroll vertically. We've also given the recycler view an ID so that we can get a reference to it in our `PizzaFragment` code; we need this in order to control its behavior.

Now that we've added a recycler view to `PizzaFragment`'s layout, we need to update our fragment code to get the recycler view to use the adapter we created.

Get the recycler view to use the adapter

To get the recycler view to use the adapter, there are two things we need to do: tell the adapter what data to use, then attach the adapter to the recycler view. We can tell the adapter what data to use by passing it the pizza names and image resource IDs via its constructor. We'll then use the `RecyclerView` `setAdapter()` method to assign the adapter to the recycler view.

The code to do this is all code that you've seen before, so we'll show you the full `PizzaFragment` code on the next page.

Pizza data
Card view
Adapter
Recycler view
Clicks

The full PizzaFragment.java code

Here's our full code for *PizzaFragment.java* (update your version of the code to match our changes):

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.support.v7.widget.RecyclerView;

public class PizzaFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
                inflater.getContext(),
                android.R.layout.simple_list_item_1,
                getResources().getStringArray(R.array.pizzas));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
        RecyclerView pizzaRecycler = (RecyclerView)inflater.inflate(
                             R.layout.fragment_pizza, container, false);

        String[] pizzaNames = new String[Pizza.pizzas.length];
        for (int i = 0; i < pizzaNames.length; i++) {
            pizzaNames[i] = Pizza.pizzas[i].getName();
        }

        int[] pizzaImages = new int[Pizza.pizzas.length];
        for (int i = 0; i < pizzaImages.length; i++) {
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
        }

        CaptionedImagesAdapter adapter = new CaptionedImagesAdapter(pizzaNames, pizzaImages);
        pizzaRecycler.setAdapter(adapter);
        return pizzaRecycler;
    }
}
```

We're changing PizzaFragment to be a Fragment, not a ListFragment.

We're no longer using an ArrayAdapter, so delete this line.

We need to import the RecyclerView class.

Change this from ListFragment to Fragment.

Delete these lines, as they're no longer necessary.

Use the layout we updated on the previous page.

Add the pizza names to an array of Strings, and the pizza images to an array of ints.

Pass the arrays to the adapter.

There's just one more thing we need to do: specify how the views in the recycler view should be arranged.

A recycler view uses a layout manager to arrange its views

One of the ways in which a recycler view is more flexible than a list view is in how it arranges its views. A list view displays its views in a single vertical list, but a recycler view gives you more options. You can choose to display views in a linear list, a grid, or a staggered grid.

You specify how to arrange the views using a **layout manager**. The layout manager positions views inside a recycler view: the type of layout manager you use determines how items are positioned. Here are some examples:

We don't cover how to do it, but you can also write your own layout managers. If you search for "android recyclerview layoutmanager" you'll find many third-party ones you can use in your code, from carousels to circles.

Linear layout manager

This arranges items in a vertical or horizontal list.

Grid layout manager

This arranges items in a grid.

Staggered grid layout manager

This arranges unevenly sized items in a staggered grid.

On the next page, we'll show you how to specify which layout manager to use in your recycler view.

Specify the layout manager

You tell the recycler view which layout manager it should use by creating a new instance of the type of layout manager you want to use, then attaching it to the recycler view.

Linear layout manager

To tell the recycler view that you want it to display its views in a linear list, you'd use the following code:

```
LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());
pizzaRecycler.setLayoutManager(layoutManager);
```

This needs to be a Context. If you use this code in an activity, you use "this" instead of getActivity().

The `LinearLayoutManager` constructor takes one parameter, a `Context`. If you're using the code in an activity, you'd normally use `this` to pass it the current activity (a context). The above code uses `getActivity()` instead, as our recycler view is in a fragment.

Grid layout manager

You use similar code to specify a grid layout manager, except that you need to create a new `GridLayoutManager` object instead. The `GridLayoutManager` takes two parameters in its constructor: a `Context`, and an `int` value specifying the number of columns the grid should have.

This says that the GridLayoutManager should be two columns wide.

```
GridLayoutManager layoutManager = new GridLayoutManager(getActivity(), 2);
```

You can also change the orientation of the grid. To do this, you add two more parameters to the constructor: the orientation, and whether you want the views to appear in reverse order.

Gives the GridLayoutManager a horizontal orientation.

```
GridLayoutManager layoutManager =
        new GridLayoutManager(getActivity(), 1, GridLayoutManager.HORIZONTAL, false);
```

If you wanted to display the list in reverse order, you'd set this to true.

Staggered grid layout manager

You tell the recycler view to use a staggered grid layout manager by creating a new `StaggeredGridLayoutManager` object. Its constructor takes two parameters: an `int` value for the number of columns or rows, and an `int` value for its orientation. As an example, here's how you'd specify a staggered grid layout oriented vertically with two rows:

This gives the staggered grid layout a vertical orientation.

```
StaggeredGridLayoutManager layoutManager =
            new StaggeredGridLayoutManager(2, StaggeredGridLayoutManager.VERTICAL);
```

Let's add a layout manager to our recycler view.

The full PizzaFragment.java code

We're going to use a `GridLayoutManager` to display the pizza data in a grid. Here's the full code for *PizzaFragment.java*, update your version of the code to match our changes (in bold):

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.GridLayoutManager;

public class PizzaFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        RecyclerView pizzaRecycler = (RecyclerView)inflater.inflate(
                R.layout.fragment_pizza, container, false);

        String[] pizzaNames = new String[Pizza.pizzas.length];
        for (int i = 0; i < pizzaNames.length; i++) {
            pizzaNames[i] = Pizza.pizzas[i].getName();
        }

        int[] pizzaImages = new int[Pizza.pizzas.length];
        for (int i = 0; i < pizzaImages.length; i++) {
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
        }

        CaptionedImagesAdapter adapter = new CaptionedImagesAdapter(pizzaNames, pizzaImages);
        pizzaRecycler.setAdapter(adapter);
        GridLayoutManager layoutManager = new GridLayoutManager(getActivity(), 2);
        pizzaRecycler.setLayoutManager(layoutManager);
        return pizzaRecycler;
    }
}
```

We're using this class, so we need to import it.

We're going to display the CardViews in a grid with two columns, so we're using a GridLayoutManager.

Next we'll examine what happens when the code runs, then take our app for a test drive.

What happens when the code runs

1. **The user clicks on the Pizzas tab in MainActivity.**
   `PizzaFragment` is displayed, and its `onCreateView()` method runs.

2. **The PizzaFragment onCreateView() method creates a new CaptionedImagesAdapter.**
   The method passes the names and images of the pizzas to the adapter using the adapter's constructor, and sets the adapter to the recycler view.

3. **The PizzaFragment onCreateView() method creates a GridLayoutManager and assigns it to the recycler view.**
   The `GridLayoutManager` means that the views will be displayed in a grid. As the recycler view has a vertical scrollbar, the list will be displayed vertically.

The story continues

- ☑ Pizza data
- ☑ Card view
- ☑ Adapter
- ➔ ☐ **Recycler view**
- ☐ Clicks

4 **The adapter creates a view holder for each of the CardViews the recycler view needs to display.**

5 **The adapter then binds the pizza names and images to the text view and image view in each card view.**

Let's run the app and see how it looks.

- [x] Pizza data
- [x] Card view
- [x] Adapter
- [x] Recycler view
- [ ] Clicks

Test drive the app

When we run the app, `MainActivity` is displayed. When we click on or swipe to the Pizzas tab, the pizzas are displayed in a grid. When we scroll the pizza data, `MainActivity`'s toolbar responds.

When you click on the Pizzas tab, PizzaFragment is displayed. It contains this grid of card views populated with pizza data.

When you scroll the recycler view, the toolbar scrolls too. This is because recycler views allow nested scrolling, and in Chapter 12 we told the toolbar to respond to nested scroll events.

As you can see, adding a recycler view is more involved than adding a list view, but it gives you a lot more flexibility. Most of the work comes from having to write a bespoke recycler view adapter, but you can reuse it elsewhere in your app. As an example, suppose you wanted to display pasta cards in a recycler view. You would use the same adapter we created earlier, but pass it pasta data instead of pizzas.

Before we move on, have a go at the following exercise.

RecyclerView Magnets

Use the magnets on this page and the next to create a new recycler view for the pasta dishes. The recycler view should contain a grid of card views, each one displaying the name and image of a pasta dish.

This is the code for the Pasta class.

```
package com.hfad.bitsandpizzas;

public class Pasta {
    private String name;
    private int imageResourceId;

    public static final..............[] pastas = {
            new Pasta("Spaghetti Bolognese", R.drawable.spag_bol),
            new Pasta("Lasagne", R.drawable.lasagne)
    };

    private Pasta(String name, int imageResourceId) {
        this.name = name;
        this.imageResourceId = imageResourceId;
    }

    public String ....................{
        return name;
    }

    public int ....................................{
        return imageResourceId;
    }
}
```

`getName()` `Pasta` `android:scrollbars` `RecyclerView` `android.support.v7.widget.RecyclerView` `=` `"vertical"` `getImageResourceId()`

This is the code for the layout.

```
< ..................................................................
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pasta_recycler"

    ..................................................................
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

This is the code for PastaFragment.java.

```
...

public class PastaFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        RecyclerView pastaRecycler = (RecyclerView)inflater.inflate(

                ..............................................., container, false);

        String[] pastaNames = new String[Pasta.pastas.length];
        for (int i = 0; i < pastaNames.length; i++) {
            pastaNames[i] = Pasta.pastas[i].getName();
        }

        int[] pastaImages = new int[Pasta.pastas.length];
        for (int i = 0; i < pastaImages.length; i++) {
            pastaImages[i] = Pasta.pastas[i].getImageResourceId();
        }

       ........................................adapter =

                    new ........................................ (pastaNames, ............................);
        pastaRecycler.setAdapter(adapter);

       ...................................... layoutManager = new ......................................(getActivity(), 2);
        pastaRecycler.setLayoutManager(layoutManager);
        return pastaRecycler;
    }
}
```

CaptionedImagesAdapter

ArrayAdapter

CaptionedImagesAdapter

GridLayout

GridLayoutManager

R.layout.fragment_pasta

pastaImages

GridLayout

GridLayoutManager

ArrayAdapter

RecyclerView Magnets Solution

Use the magnets on this page and the next to create a new recycler view for the pasta dishes. The recycler view should contain a grid of card views, each one displaying the name and image of a pasta dish.

```
package com.hfad.bitsandpizzas;

public class Pasta {
    private String name;
    private int imageResourceId;

    public static final [Pasta] [] pastas = {
            new Pasta("Spaghetti Bolognese", R.drawable.spag_bol),
            new Pasta("Lasagne", R.drawable.lasagne)
    };

    private Pasta(String name, int imageResourceId) {
        this.name = name;
        this.imageResourceId = imageResourceId;
    }

    public String [getName()] {
        return name;
    }

    public int [getImageResourceId()] {
        return imageResourceId;
    }
}
```

It's an array of Pasta objects.

These methods are used by PastaFragment.java.

RecyclerView

This is a spare magnet.

Add the recycler view to the layout.

```
< [android.support.v7.widget.RecyclerView]
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pasta_recycler"
    [android:scrollbars] [=] ["vertical"]
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Add vertical scrollbars.

```
...

public class PastaFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        RecyclerView pastaRecycler = (RecyclerView)inflater.inflate(
                                 R.layout.fragment_pasta, container, false);

        String[] pastaNames = new String[Pasta.pastas.length];
        for (int i = 0; i < pastaNames.length; i++) {
            pastaNames[i] = Pasta.pastas[i].getName();
        }

        int[] pastaImages = new int[Pasta.pastas.length];
        for (int i = 0; i < pastaImages.length; i++) {
            pastaImages[i] = Pasta.pastas[i].getImageResourceId();
        }

        CaptionedImagesAdapter adapter =
                    new CaptionedImagesAdapter(pastaNames, pastaImages);
        pastaRecycler.setAdapter(adapter);

        GridLayoutManager layoutManager = new GridLayoutManager(getActivity(), 2);
        pastaRecycler.setLayoutManager(layoutManager);
        return pastaRecycler;
    }
}
```

Use this layout.

We're using the CaptionedImagesAdapter we wrote earlier.

Pass the pasta names and images to the adapter.

Use the GridLayoutManager to display the card views in a grid.

You didn't need to use these magnets.

GridLayout

GridLayout

ArrayAdapter

ArrayAdapter

Make the recycler view respond to clicks

- [x] Pizza data
- [x] Card view
- [x] Adapter
- [x] Recycler view
- [ ] **Clicks**

So far, we've added a recycler view to `PizzaFragment`, and created an adapter to populate it with pizza data.

The next thing we need to do is get the recycler view to respond to clicks. We'll create a new activity, `PizzaDetailActivity`, which will start when the user clicks on one of the pizzas. The name and image of the pizza the user selects will be displayed in the activity:

Before we can get the recycler view to respond to clicks, we need to create `PizzaDetailActivity`.

Create PizzaDetailActivity

To create `PizzaDetailActivity`, click on the `com.hfad.bitsandpizzas` package in the Bits and Pizzas folder structure, then go to File→New...→Activity→Empty Activity. Name the activity "PizzaDetailActivity", name the layout "activity_pizza_detail", make sure the package name is `com.hfad.bitsandpizzas`, and **check the Backwards Compatibility (AppCompat) option**.

If prompted for the activity's source language, select the option for Java.

Now let's update `PizzaDetailActivity`'s layout. Open *activity_pizza_detail.xml*, and update it with the code below, which adds a text view and image view to the layout that we'll use to display details of the pizza:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.PizzaDetailActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/toolbar" />

    <TextView
        android:id="@+id/pizza_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <ImageView
        android:id="@+id/pizza_image"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:adjustViewBounds="true"/>
</LinearLayout>
```

We'll add a toolbar to the activity.

We'll put the name of the pizza in the TextView.

We'll put the pizza image in the ImageView.

We'll look at what we need the code for *PizzaDetailActivity.java* to do on the next page.

What PizzaDetailActivity needs to do

- Pizza data
- Card view
- Adapter
- Recycler view
- **Clicks**

There are a couple of things that we need `PizzaDetailActivity` to do:

- `PizzaDetailActivity`'s main purpose is to display the name and image of the pizza the user has selected. To do this, we'll get the selected pizza's ID from the intent that starts the activity. We'll pass this to `PizzaDetailActivity` from `PizzaFragment` when the user clicks on one of the pizzas in the recycler view.
- We'll enable the `PizzaDetailActivity`'s Up button so that when the user clicks on it, they'll get returned to `MainActivity`.

Update AndroidManifest.xml to give PizzaDetailActivity a parent

We'll start by updating *AndroidManifest.xml* to specify that `MainActivity` is the parent of `PizzaDetailActivity`. This means that when the user clicks on the Up button in `PizzaDetailActivity`'s app bar, `MainActivity` will be displayed. Here's our version of *AndroidManifest.xml* (update your version to match our changes in bold):

```
<manifest ...>
    <application
        ...>
        <activity
            android:name=".MainActivity">
            ...
        </activity>
        <activity
            android:name=".OrderActivity"
            ...
        </activity>
        <activity
            android:name=".PizzaDetailActivity"
            android:parentActivityName=".MainActivity">
        </activity>
    </application>
</manifest>
```

This sets MainActivity as PizzaDetailActivity's parent.

Next, we'll update *PizzaDetailActivity.java*. You've already seen how to do everything we need, so we're just going to show you the full code.

The code for PizzaDetailActivity.java

Here's the full code for *PizzaDetailActivity.java*; update your version of the code to match ours:

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.widget.ImageView;
import android.widget.TextView;
import android.support.v4.content.ContextCompat;

public class PizzaDetailActivity extends AppCompatActivity {

    public static final String EXTRA_PIZZA_ID = "pizzaId";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pizza_detail);

        //Set the toolbar as the activity's app bar
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);

        //Display details of the pizza
        int pizzaId = (Integer)getIntent().getExtras().get(EXTRA_PIZZA_ID);
        String pizzaName = Pizza.pizzas[pizzaId].getName();
        TextView textView = (TextView)findViewById(R.id.pizza_text);
        textView.setText(pizzaName);
        int pizzaImage = Pizza.pizzas[pizzaId].getImageResourceId();
        ImageView imageView = (ImageView)findViewById(R.id.pizza_image);
        imageView.setImageDrawable(ContextCompat.getDrawable(this, pizzaImage));
        imageView.setContentDescription(pizzaName);
    }
}
```

We're using these classes, so we need to import them.

We'll use this constant to pass the ID of the pizza as extra information in the intent.

Enable the Up button.

Use the pizza ID to populate the TextView and ImageView.

From the intent, get the pizza the user chose.

Get a recycler view to respond to clicks

Next, we need to get items in the recycler view to respond to clicks so that we can start `PizzaDetailActivity` when the user clicks on a particular pizza.

When you create a navigation list with a list view, you can respond to click events within the list by giving the list view an `OnItemClickListener`. The list view then listens to each of the views that it contains, and if any of them are clicked, the list view calls its `OnItemClickListener`. That means that you can respond to list item clicks with very little code.

List views are able to do this because they inherit a bunch of functionality from a deep hierarchy of superclasses. Recycler views, however, don't have such a rich set of built-in methods, as they don't inherit from the same superclasses. Here's a class hierarchy diagram for the `ListView` and `RecyclerView` classes:

While this gives recycler views more flexibility, it also means that with a recycler view you have to do a lot more of the work yourself. So how do we get our recycler view to respond to clicks?

You can listen for view events from the adapter

To get your recycler view to respond to click events, you need access to the views that appear inside it. These views are all created inside the recycler view's adapter. When a view appears onscreen, the recycler view calls the `CaptionedImagesAdapter`'s `onBindViewHolder()` method to make the card view match the details of the list item.

When the user clicks on one of the pizza cards in the recycler view, we want to start `PizzaDetailActivity`, passing it the position of the pizza that was clicked. That means you *could* put some code inside the adapter to start an activity like this:

```
class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
...
    @Override
    public void onBindViewHolder(ViewHolder holder, final int position){
        final CardView cardView = holder.cardView;
        ImageView imageView = (ImageView)cardView.findViewById(R.id.info_image);
        Drawable drawable =
                ContextCompat.getDrawable(cardView.getContext(), imageIds[position]);
        imageView.setImageDrawable(drawable);
        imageView.setContentDescription(captions[position]);
        TextView textView = (TextView)cardView.findViewById(R.id.info_text);
        textView.setText(captions[position]);
        cardView.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(cardView.getContext(), PizzaDetailActivity.class);
                intent.putExtra(PizzaDetailActivity.EXTRA_PIZZA_ID, position);
                cardView.getContext().startActivity(intent);
            }
        });
    }
}
```

Don't update the adapter code just yet. This is just an example.

Adding this code to the adapter would start PizzaDetailActivity when a CardView is clicked.

But just because you *could* write this code, it doesn't necessarily mean that you should.

BRAIN POWER

You *could* respond to a click event by adding code to your adapter class. But can you think of a reason why you *wouldn't* want to do that?

Keep your adapters reusable

If you deal with click events in the `CaptionedImagesAdapter` class, *you'll limit how that adapter can be used*. Think about the app we're building. We want to display lists of pizzas, pasta, and stores. In each case, we'll probably want to display a list of captioned images. If we modify the `CaptionedImagesAdapter` class so that clicks always send the user to an activity that displays details of a single pizza, we won't be able to use the `CaptionedImagesAdapter` for the pasta and stores lists. We'll have to create a separate adapter for each one.

Decouple your adapter with an interface

Instead of that approach, we'll keep the code that starts the activity outside of the adapter. When someone clicks on an item in the list, we want the adapter to call the fragment that contains the list, and then the fragment code can fire off an intent to the next activity. That way we can reuse `CaptionedImagesAdapter` for the pizzas, pasta, and stores lists, and in each case leave it to the fragments to decide what happens in response to a click.

We're going to use a similar pattern to the one that allowed us to decouple a fragment from an activity in Chapter 9. We'll create a `Listener` interface inside `CaptionedImagesAdapter` like this:

```
interface Listener {
    void onClick(int position);
}
```

We'll call the listener's `onClick()` method whenever one of the card views in the recycler view is clicked. We'll then add code to `PizzaFragment` so that it implements the interface; this will allow the fragment to respond to clicks and start an activity.

This is what will happen at runtime:

1. A user will click on a card view in the recycler view.
2. The `Listener`'s `onClick()` method will be called.
3. The `onClick()` method will be implemented in `PizzaFragment`. Code in this fragment will start `PizzaDetailActivity`.

Let's start by adding code to *CaptionedImagesAdapter.java*.

Add the interface to the adapter

We've updated our *CaptionedImagesAdapter.java* code to add the `Listener` interface and call its `onClick()` method whenever one of the card views is clicked. Apply the changes below (in bold) to your code, then save your work:

```
package com.hfad.bitsandpizzas;

import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.CardView;
import android.view.ViewGroup;
import android.view.LayoutInflater;
import android.widget.ImageView;
import android.widget.TextView;
import android.graphics.drawable.Drawable;
import android.support.v4.content.ContextCompat;
import android.view.View;

class CaptionedImagesAdapter extends
        RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{

    private String[] captions;
    private int[] imageIds;
    private Listener listener;

    interface Listener {
        void onClick(int position);
    }

    public static class ViewHolder extends RecyclerView.ViewHolder {

        private CardView cardView;

        public ViewHolder(CardView v) {
            super(v);
            cardView = v;
        }
    }

    public CaptionedImagesAdapter(String[] captions, int[] imageIds){
        this.captions = captions;
        this.imageIds = imageIds;
    }
```

We're using this extra class, so we need to import it.

Add the Listener as a private variable.

Add the interface.

The code continues on the next page.

The CaptionedImagesAdapter.java code (continued)

```
    @Override
    public int getItemCount(){
        return captions.length;
    }

    public void setListener(Listener listener){
        this.listener = listener;
    }

    @Override
    public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(
            ViewGroup parent, int viewType){
        CardView cv = (CardView) LayoutInflater.from(parent.getContext())
                .inflate(R.layout.card_captioned_image, parent, false);
        return new ViewHolder(cv);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, final int position){
        CardView cardView = holder.cardView;
        ImageView imageView = (ImageView)cardView.findViewById(R.id.info_image);
        Drawable drawable =
                ContextCompat.getDrawable(cardView.getContext(), imageIds[position]);
        imageView.setImageDrawable(drawable);
        imageView.setContentDescription(captions[position]);
        TextView textView = (TextView)cardView.findViewById(R.id.info_text);
        textView.setText(captions[position]);
        cardView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (listener != null) {
                    listener.onClick(position);
                }
            }
        });
    }
}
```

Activityies and fragments will use this method to register as a listener.

You need to change the position variable to final, as it's used in an inner class.

Add the listener to the CardView.

When the CardView is clicked, call the Listener onClick() method.

Now that we've added a listener to the adapter, we need to implement it in `PizzaFragment`.

Implement the listener in PizzaFragment.java

We'll implement `CaptionedImagesAdapter`'s `Listener` interface in `PizzaFragment` so that when a card view in the recycler view is clicked, `PizzaDetailActivity` will be started. Here's the updated code; update your version of the code to match ours (our changes are in bold):

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.GridLayoutManager;
import android.content.Intent;
```

We're using an Intent to start the activity, so import this class.

```
public class PizzaFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        RecyclerView pizzaRecycler = (RecyclerView)inflater.inflate(
                R.layout.fragment_pizza, container, false);

        String[] pizzaNames = new String[Pizza.pizzas.length];
        for (int i = 0; i < pizzaNames.length; i++) {
            pizzaNames[i] = Pizza.pizzas[i].getName();
        }

        int[] pizzaImages = new int[Pizza.pizzas.length];
        for (int i = 0; i < pizzaImages.length; i++) {
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
        }
```

The code continues on the next page.

The PizzaFragment.java code (continued)

```
        CaptionedImagesAdapter adapter =
                        new CaptionedImagesAdapter(pizzaNames, pizzaImages);
        pizzaRecycler.setAdapter(adapter);
        GridLayoutManager layoutManager = new GridLayoutManager(getActivity(), 2);
        pizzaRecycler.setLayoutManager(layoutManager);

        adapter.setListener(new CaptionedImagesAdapter.Listener() {
            public void onClick(int position) {
                Intent intent = new Intent(getActivity(), PizzaDetailActivity.class);
                intent.putExtra(PizzaDetailActivity.EXTRA_PIZZA_ID, position);
                getActivity().startActivity(intent);
            }
        });
        return pizzaRecycler;
    }
}
```

This implements the Listener onClick() method. It starts PizzaDetailActivity, passing it the ID of the pizza the user chose.

That's all the code we need to make views in the recycler view respond to clicks. By taking this approach, we can use the same adapter and card view for different types of data that is composed of an image view and text view.

Let's see what happens when we run the code.

- [x] Pizza data
- [x] Card view
- [x] Adapter
- [x] Recycler view
- [x] **Clicks**

Test drive the app

When we run the app and click on the Pizzas tab, `PizzaFragment` is displayed. When we click on one of the pizzas, `PizzaDetailActivity` starts, and details of that pizza are displayed.

The card view responds to clicks, and displays `PizzaDetailActivity`.

Your Android Toolbox

You've got Chapter 13 under your belt and now you've added recycler views and card views to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Card views and recycler views have their own Support Libraries.
- Add a card view to a layout using the `<android.support.v7.widget.CardView>` element.
- Give the card view rounded corners using the `cardCornerRadius` attribute. This requires a namespace of `"http://schemas.android.com/apk/res-auto"`.
- Give the card view a drop shadow using the `cardElevation` attribute. This requires a namespace of `"http://schemas.android.com/apk/res-auto"`.
- Recycler views work with adapters that are subclasses of `RecyclerView.Adapter`.
- When you create your own `RecyclerView.Adapter`, you must define the view holder and implement the `onCreateViewHolder()`, `onBindViewHolder()`, and `getItemCount()` methods.
- You add a recycler view to a layout using the `<android.support.v7.widget.RecyclerView>` element. You give it a scrollbar using the `android:scrollbars` attribute.
- Use a layout manager to specify how items in a recycler view should be arranged. A `LinearLayoutManager` arranges items in a linear list, a `GridLayoutManager` arranges items in a grid, and a `StaggeredGridLayoutManager` arranges items in a staggered grid.

14 navigation drawers

Going Places

You've already seen how tabs help users navigate your apps.

But if you need a *large number* of them, or want to *split them into sections*, the **navigation drawer** is your new BFF. In this chapter, we'll show you how to create a navigation drawer that *slides out from the side of your activity at a single touch*. You'll learn how to give it a header using a **navigation view**, and provide it with a **structured set of menu items** to take the user to all the major hubs of your app. Finally, you'll discover how to set up a **navigation view listener** so that the drawer *responds to the slightest touch and swipe*.

Tab layouts allow easy navigation...

In Chapter 12, we introduced you to the tab layout as a way of making it easy for users to navigate around your app. In that chapter we added a Home screen tab to the Bits and Pizzas app, along with tabs for the Pizzas, Pasta, and Stores categories:

These are the tabs we created in Chapter 12.

Tab layouts work well if you have a small number of category screens that are all at the same level in the app hierarchy. But what if you want to use a large number of tabs, or group the tabs into sections?

...but navigation drawers let you show more options

If you want users to be able to navigate through a large number of options, or group them into sections, you might prefer to use a **navigation drawer**. This is a slide-out panel that contains links to other parts of the app that you can group into different sections. As an example, the Gmail app uses a navigation drawer that contains sections such as email categories, recent labels, and all labels:

The main email categories are at the top of the drawer.

Labels that have recently been clicked on are displayed in a separate section.

Finally, here's a long list of all the email labels.

This is the Gmail app. It contains a navigation drawer that slides over the app's main content. The drawer gives you lots of options you can use to navigate to different parts of the app.

When you click on an item in the navigation drawer, the drawer closes and the content for that option is displayed here.

We're going to create a navigation drawer for a new email app

We're going to create a navigation drawer for a new email app called CatChat. The navigation drawer will contain a header (including an image and some text) and a set of options. The main options will be for the user's inbox, draft messages, sent items, and trash. We'll also include a separate support section for help and feedback options:

The navigation drawer is composed of several different components. We'll go through these on the next page.

Navigation drawers deconstructed

You implement a navigation drawer by adding a **drawer layout** to your activity's layout. This defines a drawer you can open and close, and it needs to contain two views:

1. **A view for the main content.**
   This is usually a layout containing a toolbar and a frame layout, which you use to display fragments.

2. **A view for the drawer contents.**
   This is usually a navigation view, which controls most of the drawer's behavior.

When the drawer's closed, the drawer layout looks just like a normal activity. It displays the layout for its main content:

Here the drawer's closed, so it looks like a plain old activity.

The main content of the activity usually is composed of a toolbar and a frame layout that's used to display fragments.

When you open the navigation drawer, it slides over the activity's main content to display the drawer's contents. This is usually a navigation view, which displays a drawer header image and a list of options. When you click on one of these options, it either starts a new activity or displays a fragment in the activity's frame layout:

The drawer's contents are defined by a navigation view.

The drawer slides over the main content when it opens.

Here's what we're going to do

We're going to create a navigation drawer for the CatChat app.
There are four main steps we'll go through to do this:

1. **Create basic fragments and activities for the app's contents.**
When the user clicks on one of the options in the navigation drawer, we want to display the fragment or activity for that option. We'll create the fragments `InboxFragment`, `DraftsFragment`, `SentItemsFragment`, and `TrashFragment`, and activities `HelpActivity` and `FeedbackActivity`.

These are the fragments.

These are the activities.

2. **Create the drawer's header.**
We'll build a layout, *nav_header.xml*, for the drawer's header. It will contain an image and text.

3. **Create the drawer's options.**
We'll build a menu, *menu_nav.xml*, for the options the drawer will display.

4. **Create the navigation drawer.**
We'll add the navigation drawer to the app's main activity, and get it to display the header and options. We'll then write activity code to control the drawer's behavior.

The drawer's header and options

We'll create this navigation drawer.

Let's get started.

Fragments/activities
Header
Options
Drawer

Create the CatChat project

Before we begin, we need a new project for the CatChat app. Create a new Android project with an empty activity for an application named "CatChat" with a company domain of "hfad.com", making the package name `com.hfad.catchat`. The minimum SDK should be API level 19 so that it works with most devices. Specify an activity called "MainActivity" and a layout called "activity_main", and **make sure that you check the Backwards Compatibility (AppCompat) option.**

Add the v7 AppCompat and Design Support Libraries

We're going to use components and themes from the v7 AppCompat and Design Support Libraries in this chapter, so we need to add them to our project as dependencies. To do this, choose File→Project Structure in Android Studio, click on the app module, then choose Dependencies. When you're presented with the project dependencies screen, click on the "+" button at the bottom or right side of the screen. When prompted, choose the Library Dependency option, then select the Design Library from the list of possible libraries. Repeat these steps for the v7 AppCompat Support Library if Android Studio hasn't already added it for you. Finally, use the OK buttons to save your changes.

Next, we'll create four basic fragments for the app's inbox, drafts, sent messages, and trash. We'll use these fragments later in the chapter when we write the code for the navigation drawer.

Fragments/activities
Header
Options
Drawer

Create InboxFragment

We'll display `InboxFragment` when the user clicks on the inbox option in the navigation drawer. Highlight the `com.hfad.catchat` package in the *app/src/main/java* folder, then go to File→New...→Fragment→Fragment (Blank). Name the fragment "InboxFragment" and name its layout "fragment_inbox". Then update the code for *InboxFragment.java* to match our code below:

This is what InboxFragment looks like.

```
package com.hfad.catchat;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class InboxFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_inbox, container, false);
    }
}
```

All the fragments use the Fragment class from the Support Library.

And here's the code for *fragment_inbox.xml* (update your version of this code too):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.catchat.InboxFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Inbox" />
</LinearLayout>
```

InboxFragment's layout just contains a TextView. We're adding this text so we can easily tell when it's displayed.

Create DraftsFragment

When the user clicks on the drafts option in the navigation drawer, we'll show `DraftsFragment`. Select the `com.hfad.catchat` package in the *app/src/main/java* folder, and create a new blank fragment named "DraftsFragment" with a layout called of "fragment_drafts". Then replace the code for *DraftsFragment.java* with ours below:

```
package com.hfad.catchat;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class DraftsFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_drafts, container, false);
    }
}
```

Next replace the code for *fragment_drafts.xml* too:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.catchat.DraftsFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Drafts" />
</LinearLayout>
```

Create SentItemsFragment

We'll show `SentItemsFragment` when the user clicks on the sent items option in the navigation drawer. Highlight the `com.hfad.catchat` package in the *app/src/main/java* folder, and create a new blank fragment named "SentItemsFragment" with a layout called "fragment_sent_items". Then update the code for *SentItemsFragment.java* to match our code below:

```
package com.hfad.catchat;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class SentItemsFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_sent_items, container, false);
    }
}
```

And here's the code for *fragment_sent_items.xml* (update your version):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.catchat.SentItemsFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Sent items" />
</LinearLayout>
```

Create TrashFragment

When the user clicks on the trash option in the navigation drawer, we'll show `TrashFragment`. Highlight the `com.hfad.catchat` package in the *app/src/main/java* folder, and create a new blank fragment named "TrashFragment" with a layout called of "fragment_trash". Then replace the code for *TrashFragment.java* with ours below:

```
package com.hfad.catchat;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class TrashFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_trash, container, false);
    }
}
```

Next replace the code for *fragment_trash.xml* too:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.catchat.TrashFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Trash" />
</LinearLayout>
```

We've now created all the fragments we need. Next, we'll create a toolbar we can include in our activities.

Fragments/activities
Header
Options
Drawer

Create a toolbar layout

We're going to add a toolbar in a separate layout so that we can include it in each activity's layout (we'll create our activities soon). Switch to the Project view of Android Studio's explorer, select the *app/src/res/main/layout* folder, then go to the File menu and choose New → Layout resource file. When prompted, name the layout file "toolbar_main", then click on OK.

Next, open *toolbar_main.xml*, and replace the code Android Studio has created for you with the following:

```
<android.support.v7.widget.Toolbar
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />
```

This is the same toolbar code we've used in previous chapters.

Before we can use the toolbar in any of our activities, we need to change the theme used by your activity. We'll do this in app's style resource.

First, open *AndroidManifest.xml*, and make sure that the value of the theme attribute is set to `"@style/AppTheme"`. Android Studio may have set this value for you; if not, you'll need to update it to match ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            ...
        </activity>
    </application>
</manifest>
```

Android Studio may have already added this value for you.

We'll update the `AppTheme` style on the next page.

Update the app's theme

Next, we'll update the `AppTheme` style so that it uses a theme of `"Theme.AppCompat.Light.NoActionBar"`. We'll also override some of the colors that are used in the original theme.

First, open the *app/src/main/res/values* folder and check that Android Studio has created a file for you called *styles.xml*. If this file doesn't exist, you'll need to create it. To do this, select the *values* folder, then go to the File menu and choose New → "Values resource file". When prompted, name the file "styles", then click on OK.

Next, update *styles.xml* so that it matches ours:

This theme removes the default app bar (we're replacing it with a toolbar).

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

Android Studio may have added these colors for you.

The `AppTheme` style uses color resources, and these need to be included in *colors.xml*. First, make sure that Android Studio has created this file for you in the *app/src/main/res/values* folder (if it hasn't, you'll need to create it yourself). Then update *colors.xml* so that it matches our code below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Add these colors if Android Studio hasn't already done it for you.

Now that we've set up a style so that we can use a toolbar, we'll create two activities for the help and feedback options in the navigation drawer. We'll display these activities when the user selects the appropriate option.

Create HelpActivity

We'll start by creating `HelpActivity`. Select the `com.hfad.catchat` package in Android Studio, then go to the File menu and choose New. Select the option to create a new empty activity, and give it a name of "HelpActivity", with a layout name of "activity_help". Make sure the package name is `com.hfad.catchat`, and **check the Backwards Compatibility (AppCompat) checkbox.** Then update *activity_help.xml* so that it matches ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.catchat.HelpActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/toolbar" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Help" />
</LinearLayout>
```

We're adding a toolbar and "Help" text to HelpActivity.

Next update *HelpActivity.java* to match our version:

```
package com.hfad.catchat;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;

public class HelpActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

The activity needs to extend AppCompatActivity because we're using an AppCompat theme.

Create FeedbackActivity

Finally, select the `com.hfad.catchat` package again and create an empty activity called "FeedbackActivity", with a layout name of "activity_feedback". Make sure the package name is `com.hfad.catchat`, and **check the Backwards Compatibility (AppCompat) checkbox.** Then update *activity_feedback.xml* so that it matches ours below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.catchat.FeedbackActivity">

    <include
        layout="@layout/toolbar_main"
        android:id="@+id/toolbar" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Feedback" />
</LinearLayout>
```

Then update *FeedbackActivity.java* to match this version:

```
package com.hfad.catchat;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;

public class FeedbackActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_feedback);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

This activity needs to extend AppCompatActivity as well.

We need to build a navigation drawer

We've now added all the fragments and activities to our project that the options in the navigation drawer will link to. Next, we'll create the navigation drawer itself.

The navigation drawer comprises two separate components:

- **A navigation drawer header.**
  This is a layout that appears at the top of the navigation drawer. It usually consists of an image with some text, for example a photo of the user and their email account.

This is the header we'll create. It consists of an image and two pieces of text.

- **A set of options.**
  You define a set of options to be displayed in the navigation drawer underneath the header. When the user clicks on one of these options, the screen for that option is displayed as a fragment within the navigation drawer's activity, or as a new activity.

The navigation drawer will contain these options.

We're going to build these components, then use them in `MainActivity` to build a navigation drawer. We'll start with the navigation drawer header.

Create the navigation drawer's header

The navigation drawer's header comprises a simple layout that you add to a new layout file. We're going to use a new file called *nav_header.xml*. Create this file by selecting the *app/src/main/res/layout* folder in Android Studio, and choosing File→New→Layout resource file. When prompted, name the layout "nav_header".

Our layout is composed of an image and two text views. This means we need to add an image file to our project as a drawable, and two String resources. We'll start with the image file.

Add the image file

To add the image file, first switch to the Project view of Android Studio's explorer if you haven't already done so, and check whether the *app/src/main/res/drawable* folder exists in your project. If it's not already there, select the *app/src/main/res* folder in your project, go to the File menu, choose the New... option, and then click on the option to create a new Android resource directory. When prompted, choose a resource type of drawable, name it "drawable", and click on OK.

Once you've created the *drawable* folder, download the file *kitten_small.jpg* from *https://git.io/v9oet*, and add it to the *drawable* folder.

Add the String resources

Next, we'll add two String resources, which we'll use for the text views. Open the file *app/src/main/res/values/strings.xml*, then add the following resource:

```
<resources>
    ...
    <string name="app_name">CatChat</string>
    <string name="user_name">spot@catchat.com</string>
</resources>
```

Android Studio may have already added this String by default.

Now that you've added the resources, we can write the layout code. You're already familiar with the code we need to do this, so we're going to give you the full code on the next page.

Fragments/activities
Header
Options
Drawer

The full nav_header.xml code

Here's our full code for *nav_header.xml*; update your version of the file to match ours:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="180dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark" >

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/kitten_small" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:gravity="bottom|start"
        android:layout_margin="16dp" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/app_name"
            android:textAppearance="@style/TextAppearance.AppCompat.Body1" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/user_name" />
    </LinearLayout>
</FrameLayout>
```

We're explicitly setting the height of the layout to 180dp so that it doesn't take up too much space in the drawer.

The image background is quite dark, so we're using this line to make the text light.

This LinearLayout will appear on top of the ImageView. We're using it to display text at the bottom of the image.

This is a built-in style that makes the text look slightly bolder. It comes from the AppCompat Support Library.

Now that we've created the drawer's header, we'll create its list of options.

The drawer gets its options from a menu

The navigation drawer gets its list of options from a menu resource file. The code to do this is similar to that needed to add a set of options to an app bar.

Before we look at the code to add the options to the navigation drawer, we need to add a menu resource file to our project. To do this, select the *app/src/main/res* folder in Android Studio, go to the File menu, and choose New. Then select the option to create a new Android resource file. You'll be prompted for the name of the resource file and the type of resource. Give it a name of "menu_nav", give it a resource type of "Menu", and make sure that the Directory name is "menu". When you click on OK, Android Studio will create the file for you.

Next we'll add String resources for the titles of our menu items so that we can use them later in the chapter. Open *strings.xml* and add the following resources:

```
<resources>
    ...
    <string name="nav_inbox">Mesagez</string>
    <string name="nav_drafts">Draftz</string>
    <string name="nav_sent">Sent mesagez</string>
    <string name="nav_trash">In da trash</string>
    <string name="nav_support">Support</string>
    <string name="nav_help">Halp</string>
    <string name="nav_feedback">Giv us feedback</string>
</resources>
```

Next we can start constructing our menu code.

We need to create a menu with two sections

As we said earlier, we want to split the items in our navigation drawer into two sections. The first section will contain options for the main places in the app the user will want to visit: her inbox, draft messages, sent items, and trash. We'll then add a separate support section for help and feedback options.

Let's start by adding the main options.

Fragments/activities
Header
Options
Drawer

Add items in the order you want them to appear in the drawer

When you design a set of options for a navigation drawer, you generally put the items the user is most likely to want to click on at the top of the list. In our case, these options are for the inbox, draft messages, sent items, and trash.

You add items to the menu resource file in the order in which you want them to appear in the drawer. For each item, you specify an ID so you can refer to it in your Java code, and a title for the text you want to appear. You can also specify an icon that will appear alongside the item's text. As an example, here's the code to add an "inbox" item:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/nav_inbox"
        android:icon="@android:drawable/sym_action_email"
        android:title="@string/nav_inbox" />

    ...

</menu>
```

You need to give the item an ID so that your activity code can respond to it being clicked.

This is a built-in drawable you can use to display an email icon.

This is the text that appears in the navigation drawer.

In the above code, we're using one of Android's built-in icons: `"@android:drawable/sym_action_email".` Android comes with a set of built-in icons that you can use in your apps. The `"@android:drawable"` part tells Android you want to use one of these icons. You can see the full list of available icons when you start typing the icon name in Android Studio:

How to group items together

Fragments/activities
Header
Options
Drawer

As well as adding menu items individually, you can add them as part of a group. You define a group using the `<group>` element like this:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group>
        ...
    </group>
</menu>
```

Any items you want to include in the group go here.

This is useful if you want to apply an attribute to an entire group of items. As an example, you can highlight which item in the drawer the user has selected by setting the group's `android:checkableBehavior` attribute to `"single"`. This behavior is helpful when you intend to display screens for the items as fragments inside the navigation drawer's activity (in our case `MainActivity`), as it makes it easy to tell which option is currently selected:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        ...
    </group>
</menu>
```

This means that a single item in the group will be highlighted (the option the user selects).

You can highlight an item in the navigation drawer by default by setting its `android:checked` attribute to `"true"`. As an example, here's how you highlight the inbox item:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_inbox"
            android:icon="@android:drawable/sym_action_email"
            android:title="@string/nav_inbox"
            android:checked="true" />
     ...
    </group>
</menu>
```

This highlights the item in the navigation drawer by default.

We'll show you the full code for the first four menu items on the next page.

We'll use a group for the first section

We're going to add the inbox, drafts, sent messages, and trash options to our menu resource file as a group, and highlight the first item by default. We're using a group for these items because the screen for each option is a fragment, which we'll display in `MainActivity`.

Here's our code; update your version of *menu_nav.xml* to match ours.

The code on this page adds these four items.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_inbox"
            android:icon="@android:drawable/sym_action_email"
            android:title="@string/nav_inbox"
            android:checked="true" />
        <item
            android:id="@+id/nav_drafts"
            android:icon="@android:drawable/ic_menu_edit"
            android:title="@string/nav_drafts" />
        <item
            android:id="@+id/nav_sent"
            android:icon="@android:drawable/ic_menu_send"
            android:title="@string/nav_sent" />
        <item
            android:id="@+id/nav_trash"
            android:icon="@android:drawable/ic_menu_delete"
            android:title="@string/nav_trash" />
    </group>

</menu>
```

Add this group and the four items it contains to your menu resource file so they'll appear in the navigation drawer.

That's the first group of items sorted. We'll deal with the remaining items next.

Add the support section as a submenu

The second set of items in the navigation drawer forms a separate section. There's a heading of "Support," along with help and feedback options for the user to click on.

To create this section, we'll start by adding the Support heading as a separate item. As it's a heading, we only need to give it a title; it doesn't need an icon, and we're not assigning it an ID as we don't need it to respond to clicks:

```
...
    <item android:title="@string/nav_support">
    </item>
...
```

This adds a Support heading to the navigation drawer.

We want the help and feedback options to appear within the Support section, so we'll add them as separate items in a submenu inside the support item:

```
...
    <item android:title="@string/nav_support">
        <menu>
            <item
                android:id="@+id/nav_help"
                android:icon="@android:drawable/ic_menu_help"
                android:title="@string/nav_help"/>
            <item
                android:id="@+id/nav_feedback"
                android:icon="@android:drawable/sym_action_email"
                android:title="@string/nav_feedback" />
        </menu>
    </item>
...
```

This defines a submenu inside the Support item.

The above code adds these two items.

Note that we haven't put these items inside a group, so if the user clicks one of them, it won't be highlighted in the navigation drawer. This is because the help and feedback options will be displayed in new activities, not as fragments in the navigation drawer's activity.

We'll show you the full menu code on the next page.

Fragments/activities
Header
Options
Drawer

The full menu_nav.xml code

Here's the full code for *menu_nav.xml*; update your version of the code to match ours:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_inbox"
            android:icon="@android:drawable/sym_action_email"
            android:title="@string/nav_inbox" />
        <item
            android:id="@+id/nav_drafts"
            android:icon="@android:drawable/ic_menu_edit"
            android:title="@string/nav_drafts" />
        <item
            android:id="@+id/nav_sent"
            android:icon="@android:drawable/ic_menu_send"
            android:title="@string/nav_sent" />
        <item
            android:id="@+id/nav_trash"
            android:icon="@android:drawable/ic_menu_delete"
            android:title="@string/nav_trash" />
    </group>

    <item android:title="@string/nav_support">
        <menu>
            <item
                android:id="@+id/nav_help"
                android:icon="@android:drawable/ic_menu_help"
                android:title="@string/nav_help"/>
            <item
                android:id="@+id/nav_feedback"
                android:icon="@android:drawable/sym_action_email"
                android:title="@string/nav_feedback" />
        </menu>
    </item>
</menu>
```

These are the main options.

This is the support section.

The code on this page creates the full menu.

Now that we've added a menu and navigation drawer header layout, we can create the actual drawer.

How to create a navigation drawer

- Fragments/activities
- Header
- Options
- **Drawer**

You create a navigation drawer by adding a drawer layout to your activity's layout as its root element. The drawer layout needs to contain two things: a view or view group for the activity's content as its first element, and a navigation view that defines the drawer as its second:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
        ...
    </LinearLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:headerLayout="@layout/nav_header"
        app:menu="@menu/menu_nav" />
</android.support.v4.widget.DrawerLayout>
```

The DrawerLayout defines the drawer.

You give it an ID so you can refer to it in your activity code.

The DrawerLayout's first view is a layout for the activity's main content. You see it when the drawer is closed.

The NavigationView defines the drawer's contents.

This attaches the drawer to the start edge of the activity (the left for left-to-right languages).

This is the layout for the drawer's header.

This is the menu resource file containing the drawer's options.

There are two key `<NavigationView>` attributes that you use to control the drawer's appearance: `headerLayout` and `menu`.

The `app:headerLayout` attribute specifies the layout that should be used for the navigation drawer's header (in this case *nav_header.xml*). This attribute is optional.

You use the `app:menu` attribute to say which menu resource file contains the drawer's options (in this case *menu_drawer.xml*). If you don't include this attribute, your navigation drawer won't include any items.

- ✓ Fragments/activities
- ✓ Header
- ✓ Options
- → Drawer

The full code for activity_main.xml

We're going to add a navigation drawer to `MainActivity`'s layout that uses the header layout and menu we created earlier in the chapter. The layout's main content will comprise a toolbar and frame layout. We'll use the frame layout later in the chapter to display fragments.

Here's our full code for *activity_main.xml*; update your code to match ours:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <include
            layout="@layout/toolbar_main"
            android:id="@+id/toolbar" />

        <FrameLayout
            android:id="@+id/content_frame"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </LinearLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:headerLayout="@layout/nav_header"
        app:menu="@menu/menu_nav" />
</android.support.v4.widget.DrawerLayout>
```

The layout's root element is a DrawerLayout.

It has an ID so we can refer to it in our activity code later.

This is for the drawer's main content.

The activity's main content is composed of a Toolbar, and a FrameLayout in which we'll display fragments.

The NavigationView defines the drawer's appearance and much of its behavior. We're giving it an ID, as we'll need to refer to it in our activity code.

We're using the layout we created earlier as the drawer's header, and the menu resource file for the list of options.

Before we run the app to see how the navigation drawer's looking, we'll update `MainActivity` to display `InboxFragment` in the frame layout when the activity gets created.

Add InboxFragment to MainActivity's frame layout

- ✓ Fragments/activities
- ✓ Header
- ✓ Options
- → ☐ Drawer

When we created our menu resource file, we set the inbox option to be highlighted by default. We'll therefore display `InboxFragment` in `MainActivity`'s frame layout when the activity is created so that it matches the drawer's contents. We'll also set the toolbar as the activity's app bar so that it displays the app's title.

Here's our code for *MainActivity.java*; replace your version of the code to match ours:

```
package com.hfad.catchat;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        Fragment fragment = new InboxFragment();
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.content_frame, fragment);
        ft.commit();
    }
}
```

Make sure the activity extends the AppCompatActivity class, as we're using an AppCompat theme and support fragments.

Set the Toolbar as the activity's app bar.

Use a fragment transaction to display an instance of InboxFragment.

Let's see what happens when we run the app.

- ☑ Fragments/activities
- ☑ Header
- ☑ Options
- ☐ **Drawer**

Test drive the app

When we run the app, `InboxFragment` is displayed in `MainActivity`. When you swipe the app from the left side of the screen (in left-to-right languages like English) the navigation drawer is displayed. The navigation drawer contains a header layout, and the list of options we defined in our menu resource file. The first option is automatically highlighted:

In right-to-left languages, the drawer will appear on the right side of the screen instead.

Clicking on the drawer options doesn't do anything yet, as we haven't written any code in `MainActivity` to control how the drawer operates. We'll do that next.

What the activity code needs to do

Fragments/activities
Header
Options
Drawer

There are three things we need our activity code to do:

1. **Add a drawer toggle.**
   This provides a visual sign to the user that the activity contains a navigation drawer. It adds a "burger" icon to the toolbar, and you can click on this icon to open the drawer.

This is the "burger" icon. Clicking on it opens the navigation drawer.

2. **Make the drawer respond to clicks.**
   When the user clicks on one of the options in the navigation drawer, we'll display the appropriate fragment or activity and close the drawer.

When the user clicks on one of the main options, we'll display the fragment for that option and close the drawer. The option will be highlighted in the navigation drawer the next time we open it.

When the user clicks on one of these options, we'll start the appropriate activity.

3. **Close the drawer when the user presses the Back button.**
   If the drawer's open, we'll close it when the user clicks on the Back button. If the drawer's already closed, we'll get the Back button to function as normal.

We'll start by adding the drawer toggle.

Add a drawer toggle

The first thing we'll do is add a drawer toggle so that we can open the navigation drawer by clicking on an icon in the toolbar.

We'll start by creating two String resources to describe the "open drawer" and "close drawer" actions; these are required for accessibility purposes. Add the two Strings below to *strings.xml*:

```
<string name="nav_open_drawer">Open navigation drawer</string>
<string name="nav_close_drawer">Close navigation drawer</string>
```

You create the drawer toggle in the activity's `onCreate()` method by creating a new instance of the **`ActionBarDrawerToggle`** class and adding it to the drawer layout. We'll show you the code for this first, then add it to `MainActivity` later in the chapter.

The `ActionBarDrawerToggle` constructor takes five parameters: the current activity, the drawer layout, the toolbar, and the IDs of two String resources for opening and closing the drawer (the String resources we added above):

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
...
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this,
                                                         drawer,
                                                         toolbar,
                                                         R.string.nav_open_drawer,
                                                         R.string.nav_close_drawer);
```

The current activity

The activity's DrawerLayout

The activity's toolbar

This adds the burger icon to your toolbar.

These Strings are needed for accessibility.

Once you've created the drawer toggle, you add it to the drawer layout by calling the `DrawerLayout addDrawerListener()` method, passing the toggle as a parameter:

```
drawer.addDrawerListener(toggle);
```

Finally, you call the toggle's `syncState()` method to synchronize the icon on the toolbar with the state of the drawer. This is because the icon changes when you click on it to open the drawer:

```
toggle.syncState();
```

We'll add the drawer toggle to `MainActivity`'s `onCreate()` method in a few pages.

Respond to the user clicking items in the drawer

Next, we'll get `MainActivity` to respond to items in the navigation drawer being clicked by getting the activity to implement a **`NavigationView.OnNavigationItemSelectedListener`** interface. Doing this means that whenever an item is clicked, a new method we'll create in `MainActivity`, `onNavigationItemSelected()`, will get called. We'll use this method to display the screen for the appropriate option.

First, we'll get `MainActivity` to implement the interface using the code below. This code turns `MainActivity` into a listener for the navigation view:

```
...
import android.support.design.widget.NavigationView;

public class MainActivity extends AppCompatActivity
                          implements NavigationView.OnNavigationItemSelectedListener {
    ...
}
```

Implementing this interface means that your activity can respond to the user clicking options in the navigation drawer.

Next we need to register the listener, `MainActivity`, with the navigation view so that it will be notified when the user clicks on one of the options in the drawer. We'll do this by getting a reference to the navigation view in the activity's `onCreate()` method, and calling its `setNavigationItemSelectedListener()` method:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
}
```

This registers the activity as a listener on the navigation view so it will be notified if the user clicks on an item.

Finally, we need to implement the `onNavigationItemSelected()` method.

Implement the onNavigationItemSelected() method

The `onNavigationItemSelected()` method gets called when the user clicks on one of the items in the navigation drawer. It takes one parameter, the `MenuItem` that was clicked, and returns a boolean to indicate whether the item in the drawer should be highlighted:

This method gets called whenever an item in the drawer is clicked. Its parameter is the clicked item.

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    //Code to handle navigation clicks
}
```

The code in this method needs to display the appropriate screen for the clicked item. If the item is an activity, the code needs to start it with an intent. If the item is a fragment, it needs to be displayed in `MainActivity`'s frame layout using a fragment transaction.

When you display fragments by clicking on an item in a navigation drawer, you don't generally add the transaction to the back stack as we did previously. This is because when the user clicks on the Back button, they don't expect to revisit every option they clicked on in the drawer. Instead, you use code like this:

```
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
ft.replace(R.id.content_frame, fragment);
ft.commit();
```

This is the same fragment transaction code you've seen before except that we're not adding the transaction to the activity's back stack.

Finally, you need to close the drawer. To do this, you get a reference to the drawer layout, and call its `closeDrawer()` method:

```
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
drawer.closeDrawer(GravityCompat.START);
```

We're using GravityCompat.START because we've attached the drawer to the activity's start edge. If we'd attached it to the end edge, we'd use GravityCompat.END instead.

This closes the drawer so that it slides back to the activity's start edge.

You now know everything you need in order to write the code for the `onNavigationItemSelected()` method, so have a go at the following exercise.

Code Magnets

When the user clicks on an item in the navigation drawer, we need to display the appropriate screen for that item. If it's a fragment, we need to display it in the `content_frame` frame layout. If it's an activity, we need to start it. Finally, we need to close the navigation drawer.

See if you can complete the code below and on the next page. You won't need to use all of the magnets.

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {

    int id = item.                    ;
    Fragment fragment = null;
    Intent intent = null;

    switch(           ){
        case R.id.nav_drafts:

            fragment =                                        ;

                              ;
        case R.id.nav_sent:

            fragment =                                                ;

                              ;
        case R.id.nav_trash:

            fragment =                                        ;

                              ;
        case R.id.nav_help:

            intent = new Intent(              ,                                  );

                              ;
```

```
        case R.id.nav_feedback:

            intent = new Intent(..............,.......................................);

            ..............;
        default:

            fragment =.........................................;
    }

    if (....................!= null) {

        FragmentTransaction ft = getSupportFragmentManager().....................................;

        ft.replace(R.id.content_frame,.....................);

        ft.....................;
    } else {

        startActivity(...................);
    }

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

    drawer.......................(.....................................);
    return true;
}
```

closeDrawer
GravityCompat.START
fragment
FeedbackActivity.class
new
this
beginTransaction()
break
break
intent
DraftsFragment()
break
break
HelpActivity.class
TrashFragment()
break
InboxFragment()
getItemId()
id
HelpActivity
SentItemsFragment()
new
START
new
this
commit()
new
FeedbackActivity
fragment

Code Magnets Solution

When the user clicks on an item in the navigation drawer, we need to display the appropriate screen for that item. If it's a fragment, we need to display it in the `content_frame` frame layout. If it's an activity, we need to start it. Finally, we need to close the navigation drawer.

See if you can complete the code below and on the next page. You won't need to use all of the magnets.

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {

    int id = item.[getItemId()];
    Fragment fragment = null;
    Intent intent = null;

    switch([id]){
        case R.id.nav_drafts:

            fragment = [new] [DraftsFragment()];

            [break];
        case R.id.nav_sent:

            fragment = [new] [SentItemsFragment()];

            [break];
        case R.id.nav_trash:

            fragment = [new] [TrashFragment()];

            [break];
        case R.id.nav_help:

            intent = new Intent([this], [HelpActivity.class]);

            [break];
```

Get the ID of the item that was selected.

Save an instance of the fragment we want to display in the fragment variable.

Construct an intent to start HelpActivity if the Help option's clicked.

```
        case R.id.nav_feedback:

            intent = new Intent( [this] , [FeedbackActivity.class] );

            [break] ;
        default:

            fragment = [new] [InboxFragment()] ;
    }

    if ( [fragment] != null) {

        FragmentTransaction ft = getSupportFragmentManager().[beginTransaction()] ;

        ft.replace(R.id.content_frame, [fragment] );

        ft.[commit()] ;
    } else {

        startActivity( [intent] );
    }

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

    drawer.[closeDrawer] ( [GravityCompat.START] );
    return true;
}
```

If the feedback option is clicked, we need to start FeedbackActivity.

Display InboxFragment by default, as it's the first option in the drawer.

If we need to display a fragment, use a fragment transaction.

If we need to display an activity, use the intent we constructed to start it.

Finally, close the drawer.

We'll add this code to MainActivity.java in a couple of pages.

You didn't need to use these magnets.

HelpActivity

FeedbackActivity

START

Close the drawer when the user presses the Back button

Finally, we'll override what happens when the Back button's pressed. If the user presses the Back button when the navigation drawer's open, we'll close the drawer. If the drawer's already closed we'll get the Back button to function as normal.

To do this, we'll implement the activity's `onBackPressed()` method, which gets called whenever the user clicks on the Back button. Here's the code:

This gets called when the Back button gets pressed.

```
@Override
public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}
```

If the drawer is currently open, close it.

Otherwise, call up to the superclass onBackPressed() method.

That's everything we need for `MainActivity`. We'll show you the full code over the next couple of pages, and then take it for a test drive.

there are no Dumb Questions

Q: Do you have to use a navigation view for your drawer contents?

A: No, but it's *much* easier if you do. Before the Android Design Library came out, it was common practice to use a list view instead. This approach is still possible, but it requires a lot more code.

Q: Can your activity contain more than one navigation drawer?

A: Your activity can have one navigation drawer per vertical edge of its layout. To add a second navigation drawer, add an extra navigation view to your drawer layout underneath the first.

- Fragments/activities
- Header
- Options
- **Drawer**

The full MainActivity.java code

Here's the full code for *MainActivity.java*; update your version of the code to match ours:

```
package com.hfad.catchat;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarDrawerToggle;
import android.support.design.widget.NavigationView;
import android.view.MenuItem;
import android.content.Intent;
import android.support.v4.view.GravityCompat;

public class MainActivity extends AppCompatActivity
                        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this,
                                                                 drawer,
                                                                 toolbar,
                                                                 R.string.nav_open_drawer,
                                                                 R.string.nav_close_drawer);

        drawer.addDrawerListener(toggle);
        toggle.syncState();
```

We're using these extra classes so we need to import them.

Implementing this interface means the activity can listen for clicks.

Add a drawer toggle.

The code continues on the next page.

MainActivity.java (continued)

- ☑ Fragments/activities
- ☑ Header
- ☑ Options
- ☐ **Drawer**

```
    NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);

    Fragment fragment = new InboxFragment();
    FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
    ft.add(R.id.content_frame, fragment);
    ft.commit();
}

@Override
public boolean onNavigationItemSelected(MenuItem item) {
    int id = item.getItemId();
    Fragment fragment = null;
    Intent intent = null;

    switch(id){
        case R.id.nav_drafts:
            fragment = new DraftsFragment();
            break;
        case R.id.nav_sent:
            fragment = new SentItemsFragment();
            break;
        case R.id.nav_trash:
            fragment = new TrashFragment();
            break;
        case R.id.nav_help:
            intent = new Intent(this, HelpActivity.class);
            break;
        case R.id.nav_feedback:
            intent = new Intent(this, FeedbackActivity.class);
            break;
        default:
            fragment = new InboxFragment();
    }
```

Register the activity with the navigation view as a listener.

This method gets called when the user clicks on one of the items in the drawer.

The code continues on the next page.

MainActivity.java (continued)

```
        if (fragment != null) {
            FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
            ft.replace(R.id.content_frame, fragment);
            ft.commit();
        } else {
            startActivity(intent);
        }

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        drawer.closeDrawer(GravityCompat.START);
        return true;
    }

    @Override
    public void onBackPressed() {
        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        if (drawer.isDrawerOpen(GravityCompat.START)) {
            drawer.closeDrawer(GravityCompat.START);
        } else {
            super.onBackPressed();
        }
    }
}
```

Display the appropriate fragment or activity, depending on which option in the drawer the user selects.

Close the drawer when the user selects one of the options.

When the user presses the Back button, close the drawer if it's open.

Let's see what happens when we run the code.

Test drive the app

- Fragments/activities
- Header
- Options
- **Drawer**

When we run the app, a drawer toggle icon is displayed in the toolbar. Clicking on this icon opens the navigation drawer. When we click on one of the first four options, the fragment for that option is displayed in `MainActivity` and the drawer closes; the option for that item is highlighted the next time we open the drawer. When we click on one of the last two options, the activity for that option is started.

We've created a fully operational navigation drawer.

Your Android Toolbox

You've got Chapter 14 under your belt and now you've added navigation drawers to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Use a navigation drawer if you want to provide the user with a large number of shortcuts, or group them into sections.
- Create a navigation drawer by adding a **drawer layout** to your activity's layout. The drawer layout's first element needs to be a view that defines the activity's main content, usually a layout containing a `Toolbar` and `FrameLayout`. Its second element defines the contents of the drawer, usually a `NavigationView`.
- The **`NavigationView`** comes from the Design Support Library. It controls most of the drawer's behavior.
- You add a header to your drawer by creating a layout for it, and adding the header's resource ID to the navigation view's `headerLayout` attribute.
- You add items to the drawer by creating a menu resource, and adding the menu's resource ID to the navigation view's `menu` attribute.
- Add items to the menu resource in the order in which you want them to appear in the drawer.
- If you want to highlight which item in the drawer the user selects, add the menu items to a group and set the group's `checkableBehavior` attribute to `"single"`.
- Use an **`ActionBarDrawerToggle`** to display a "burger" icon in the activity's toolbar. This provides a visual sign that the activity has a navigation drawer. Clicking on it opens the drawer.
- Respond to the user clicking on items in the drawer by making your activity implement the **`NavigationView.OnNavigationItemSelectedListener`** interface. Register the activity with the navigation view as a listener, then implement the `onNavigationItemSelected()` method.
- Close the navigation drawer using the `DrawerLayout closeDrawer()` method.

15 SQLite databases

Fire Up the Database

If you're recording high scores or saving tweets, your app will need to store data. And on Android you usually keep your data safe inside a **SQLite database**. In this chapter, we'll show you how to **create a database, add tables to it, and prepopulate it with data**, all with the help of the friendly **SQLite helper**. You'll then see how you can cleanly roll out **upgrades** to your database structure, and how to **downgrade** it if you need to undo any changes.

Back to Starbuzz

Back in Chapter 7, we created an app for Starbuzz Coffee. The app allows the user to navigate through a series of screens so that she can see the drinks available at Starbuzz.

The Starbuzz database gets its drink data from a `Drink` class containing a selection of drinks available at Starbuzz. While this made building the first version of the app easier, there's a better way of storing and persisting data.

Over the next two chapters, we're going to change the Starbuzz app so that it gets its data from a SQLite database. In this chapter, we'll see how to create the database, and in the next chapter, we'll show you how to connect activities to it.

Android uses SQLite databases to persist data

All apps need to store data, and the main way you do that in Androidville is with a **SQLite database**. Why SQLite?

- **It's lightweight.**
  Most database systems need a special database server process in order to work. SQLite doesn't; a SQLite database is just a file. When you're not using the database, it doesn't use up any processor time. That's important on a mobile device, because we don't want to drain the battery.

- **It's optimized for a single user.**
  Our app is the only thing that will talk to the database, so we shouldn't have to identify ourselves with a username and password.

- **It's stable and fast.**
  SQLite databases are amazingly stable. They can handle database transactions, which means if you're updating several pieces of data and mess up, SQLite can roll the data back. Also, the code that reads and writes the data is written in optimized C code. Not only is it fast, but it also reduces the amount of processor power it needs.

We're going to go through the basics of SQLite in this chapter.

If you plan on doing a lot of database heavy lifting in your apps, we suggest you do more background reading on SQLite and SQL.

Where's the database stored?

Android automatically creates a folder for each app where the app's database can be stored. When we create a database for the Starbuzz app, it will be stored in the following folder on the device:

/data/data/com.hfad.starbuzz/databases

com.hfad.starbuzz is the app's unique identifier.

An app can store several databases in this folder. Each database consists of two files.

The first file is the **database file** and has the same name as your database—for example, "starbuzz". This is the main SQLite database file. All of your data is stored in this file.

The second file is the **journal file**. It has the same name as your database, with a suffix of "-journal"—for example, "starbuzz-journal". The journal file contains all of the changes made to your database. If there's a problem, Android will use the journal to undo your latest changes.

Android comes with SQLite classes

Android uses a set of classes that allows you to manage a SQLite database. There are three types of object that do the bulk of this work:

The SQLite Helper

A SQLite helper enables you to create and manage databases. You create one by extending the `SQLiteOpenHelper` class.

The SQLite Database

The `SQLiteDatabase` class gives you access to the database. It's like a `SQLConnection` in JDBC.

Cursors

A `Cursor` lets you read from and write to the database. It's like a `ResultSet` in JDBC.

We're going to use these objects to show you how to create a SQLite database your app can use to persist data by replacing the `Drink` class with a SQLite database.

there are no Dumb Questions

Q: If there's no username and password on the database, how is it kept secure?

A: The directory where an app's databases are stored is only readable by the app itself. The database is secured down at the operating system level.

Q: Can I write an Android app that talks to some other kind of external database, such as Oracle?

A: There's no reason why you can't access other databases over a network connection, but be careful to conserve the resources used by Android. For example, you might use less battery power if you access your database via a web service. That way, if you're not talking to the database, you're not using up any resources.

Q: Why doesn't Android use JDBC to access SQLite databases?

A: We know we're going to be using a SQLite database, so using JDBC would be overkill. Those layers of database drivers that make JDBC so flexible would just drain the battery on an Android device.

Q: Is the database directory inside the app's directory?

A: No. It's kept in a separate directory from the app's code. That way, the app can be overwritten with a newer version, but the data in the database will be kept safe.

The current Starbuzz app structure

Here's a reminder of the current structure of the Starbuzz app:

1. **TopLevelActivity displays a list of options: Drinks, Food, and Stores.**
2. **When the user clicks on the Drinks option, it launches DrinkCategoryActivity.**
   This activity displays a list of drinks that it gets from the Java `Drink` class.
3. **When the user clicks on a drink, its details get displayed in DrinkActivity.**
   `DrinkActivity` gets details of the drink from the Java `Drink` class.

How does the app structure need to change if we're to use a SQLite database?

Do this!

We're going to update the Starbuzz app in this chapter, so open your original Starbuzz project in Android Studio.

Let's change the app to use a database

We'll use a SQLite helper to create a SQLite database we can use with our Starbuzz app. We're going to replace our `Drink` Java class with a database, so we need our SQLite helper to do the following:

1. **Create the database.**
   Before we can do anything else, we need to get the SQLite helper to create version 1 (the first version) of our Starbuzz database.

2. **Create the Drink table and populate it with drinks.**
   Once we have a database, we can create a table in it. The table's structure needs to reflect the attributes in the current `Drink` class, so it needs to be able to store the name, description, and image resource ID of each drink. We'll then add three drinks to it.

The app has the same structure as before except that we're replacing the file *Drink.java* with a SQLite helper and a SQLite Starbuzz database. The SQLite helper will maintain the Starbuzz database, and provide access to it for the other activities. We'll change the activities to use the database in the next chapter.

Let's start by looking at the SQLite helper.

The SQLite helper manages your database

The **`SQLiteOpenHelper`** class is there to help you create and maintain your SQLite databases. Think of it as a personal assistant who takes care of the general database housekeeping.

Let's look at some typical tasks that the SQLite helper can assist you with:

Creating the database

When you first install an app, the database file won't exist. The SQLite helper will make sure the database file is created with the correct name and with the correct table structures installed.

Getting access to the database

Our app shouldn't need to know all of the details about where the database file is, so the SQLite helper can serve us with an easy-to-use database object whenever we need it. At all hours, day or night.

The SQLite helper

Keeping the database shipshape

The structure of the database will probably change over time, and the SQLite helper can be relied upon to convert an old version of a database into a shiny, spiffy new version, with all the latest database structures it needs.

Create database
Create table

Create the SQLite helper

You create a SQLite helper by writing a class that extends the **`SQLiteOpenHelper`** class. When you do this, you ***must*** override the `onCreate()` and `onUpgrade()` methods. These methods are mandatory.

The `onCreate()` method gets called when the database first gets created on the device. The method should include all the code needed to create the tables you need for your app.

The `onUpgrade()` method gets called when the database needs to be upgraded. As an example, if you need to modify the structure of the database after it's been released, this is the method to do it in.

In our app, we're going to use a SQLite helper called `StarbuzzDatabaseHelper`. Create this class in your Starbuzz project by switching to the Project view of Android Studio's explorer, selecting the `com.hfad.starbuzz` package in the *app/src/main/java* folder, and navigating to File→New...→Java Class. Name the class "StarbuzzDatabaseHelper", make sure the package name is `com.hfad.starbuzz` and then replace its contents with the code below:

```
package com.hfad.starbuzz;

import android.database.sqlite.SQLiteOpenHelper;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper {

    StarbuzzDatabaseHelper(Context context) {
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

This is the full path of the SQLiteOpenHelper class.

SQLite helpers must extend the SQLiteOpenHelper class.

We'll write the code for the constructor on the next page.

The onCreate() and onUpgrade() methods are mandatory. We've left them empty for now, and we'll look at them in more detail throughout the chapter.

To get the SQLite helper to do something, we need to add code to its methods. The first thing to do is tell the SQLite helper what database it needs to create.

Create database
Create table

Specify the database

There are two pieces of information the SQLite helper needs in order to create the database.

First, we need to give the database a name. By giving the database a name, we make sure that the database remains on the device when it's closed. If we don't, the database will only be created in memory, so once the database is closed, it will disappear.

Creating databases that are only held in memory can be useful when you're testing your app.

The second piece of information we need to provide is the version of the database. The database version needs to be an integer value, starting at 1. The SQLite helper uses this version number to determine whether the database needs to be upgraded.

You specify the database name and version by passing them to the constructor of the `SQLiteOpenHelper` superclass. We're going to give our database a name of "starbuzz", and as it's the first version of the database, we'll give it a version number of 1. Here's the code we need (update your version of *StarbuzzDatabaseHelper.java* to match the code below):

Name: "starbuzz"
Version: 1

SQLite database

```
...
class StarbuzzDatabaseHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "starbuzz"; // the name of our database
    private static final int DB_VERSION = 1; // the version of the database

    StarbuzzDatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }
...
}
```

We're calling the constructor of the SQLiteOpenHelper superclass, and passing it the database name and version.

This parameter is an advanced feature relating to cursors. We'll cover cursors in the next chapter.

The constructor specifies details of the database, but the database doesn't get created at that point. The SQLite helper waits until the app needs to access the database, and then creates the database.

We've now done everything we need to tell the SQLite helper what database to create. The next step is to tell it what tables to create.

Create database
Create table

Inside a SQLite database

The data inside a SQLite database is stored in tables. A table contains several rows, and each row is split into columns. A column contains a single piece of data, like a number or a piece of text.

You need to create a table for each distinct piece of data that you want to record. In the Starbuzz app, for example, we'll need to create a table for the drink data. It will look something like this:

The columns in the table are _id, NAME, DESCRIPTION, and IMAGE_RESOURCE_ID. The Drink class contained similarly named attributes.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	"Latte"	"Espresso and steamed milk"	54543543
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453
3	"Filter"	"Our best drip coffee"	44324234

Some columns can be specified as **primary keys**. A primary key uniquely identifies a single row. If you say that a column is a primary key, then the database won't allow you to store rows with duplicate keys.

We recommend that your tables have a single column called _id to hold the primary key that contains integer values. This is because Android code is hardwired to expect a numeric _id column, so not having one can cause you problems later on.

Storage classes and data types

Each column in a table is designed to store a particular type of data. For example, in our DRINK table, the DESCRIPTION column will only ever store text data. Here are the main data types you can use in SQLite, and what they can store:

INTEGER	Any integer type
TEXT	Any character type
REAL	Any floating-point number
NUMERIC	Booleans, dates, and date-times
BLOB	Binary Large Object

Unlike most database systems, you don't need to specify the column size in SQLite. Under the hood, the data type is translated into a much broader storage class. This means you can say very generally what kind of data you're going to store, but you're not forced to be specific about the size of data.

It's an Android convention to call your primary key columns _id. Android code expects there to be an _id column on your data. Ignoring this convention will make it harder to get the data out of your database and into your user interface.

You create tables using Structured Query Language (SQL)

Every application that talks to SQLite needs to use a standard database language called Structured Query Language (SQL). SQL is used by almost every type of database. If you want to create the DRINK table, you will need to do it in SQL.

This is the SQL command to create the table:

The _id column is the primary key.

```
CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    NAME TEXT,
                    DESCRIPTION TEXT,
                    IMAGE_RESOURCE_ID INTEGER)
```

The table name

These are table columns.

The `CREATE TABLE` command says what columns you want in the table, and what the data type is of each column. The `_id` column is the primary key of the table, and the special keyword `AUTOINCREMENT` means that when we store a new row in the table, SQLite will automatically generate a unique integer for it.

The onCreate() method is called when the database is created

The SQLite helper is in charge of creating the SQLite database. An empty database is created on the device the first time it needs to be used, and then the SQLite helper's `onCreate()` method is called. The `onCreate()` method has one parameter, a `SQLiteDatabase` object that represents the database that's been created.

You can use the `SQLiteDatabase execSQL()` method to execute SQL on the database. This method has one parameter, the SQL you want to execute.

```
execSQL(String sql);
```

Execute the SQL in the String on the database.

The SQLiteDatabase class gives you access to the database.

We'll use the `onCreate()` method to create the DRINK table. Here's the code (we'll add the code in a few pages):

```
public void onCreate(SQLiteDatabase db){
    db.execSQL("CREATE TABLE DRINK ("
               + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
               + "NAME TEXT, "
               + "DESCRIPTION TEXT, "
               + "IMAGE_RESOURCE_ID INTEGER);");
}
```

This gives us an empty DRINK table. We want to prepopulate it with drink data, so let's look at how you do that.

Create database
Create table

Insert data using the insert() method

To insert data into a table in a SQLite database, you start by specifying what values you want to insert into the table. To do this, you first create a **`ContentValues`** object:

```
ContentValues drinkValues = new ContentValues();
```

A `ContentValues` object describes a set of data. You usually create a new `ContentValues` object for each row of data you want to create.

You add data to the `ContentValues` object using its **`put()`** method. This method adds name/value pairs of data: `NAME` is the column you want to add data to, and `value` is the data:

```
contentValues.put("NAME", "value");
```

NAME is the column you want to add data to. value is the value you want it to have.

As an example, here's how you'd use the `put()` method to add the name, description, and image resource ID of a latte to a `ContentValues` object called `drinkValues`:

This is a single row of data.

```
drinkValues.put("NAME", "Latte");
drinkValues.put("DESCRIPTION", "Espresso and steamed milk");
drinkValues.put("IMAGE_RESOURCE_ID", R.drawable.latte);
```

Put the value "Latte" in the NAME column.

Put "Espresso and steamed milk" in the DESCRIPTION column.

You need a separate call to the put() method for each value you want to enter.

Once you've added a row of data to the `ContentValues` object, you insert it into the table using the `SQLiteDatabase` `insert()` method. This method inserts data into a table, and returns the ID of the record once it's been inserted. If the method is unable to insert the record, it returns a value of `-1`. As an example, here's how you'd insert the data from `drinkValues` into the DRINK table:

```
db.insert("DRINK", null, drinkValues);
```

This inserts a single row into the table.

The middle parameter is usually set to `null`, as in the above code. It's there in case the `ContentValues` object is empty, and you want to insert an empty row into your table. It's unlikely you'd want to do this, but if you did you'd replace the `null` value with the name of one of the columns in your table.

Running the lines of code above will insert a Latte row into the DRINK table:

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	"Latte"	"Espresso and steamed milk"	54543543

A shiny new record gets inserted into the table.

The `insert()` methods inserts one row of data at a time. But what if you want to insert multiple records?

Insert multiple records

To insert multiple rows into a table, you need to make repeat calls to the `insert()` method. Each call to the method inserts a separate row.

To insert multiple rows, you usually create a new method that inserts a single row of data, and call it each time you want to add a new row. As an example, here's an `insertDrink()` method we wrote to insert drinks into the DRINK table:

```
private static void insertDrink(SQLiteDatabase db,
                                String name,
                                String description,
                                int resourceId) {
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("NAME", name);
    drinkValues.put("DESCRIPTION", description);
    drinkValues.put("IMAGE_RESOURCE_ID", resourceId);
    db.insert("DRINK", null, drinkValues);
}
```

This is the database we want to add records to.

We're passing the data to the method as parameters.

Construct a ContentValues object with the data.

Then insert the data.

To add three drinks to the DRINK table, each one a separate row of data, you'd call the method three times:

```
insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
                R.drawable.cappuccino);
insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
```

That's everything you need to know to insert data into tables. On the next page we'll show you the revised code for *StarbuzzDatabaseHelper.java*.

The StarbuzzDatabaseHelper code

Create database
Create table

Here's the complete code for *StarbuzzDatabaseHelper.java* (update your code to reflect our changes):

```
package com.hfad.starbuzz;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper{

    private static final String DB_NAME = "starbuzz"; // the name of our database
    private static final int DB_VERSION = 1; // the version of the database

    StarbuzzDatabaseHelper(Context context){
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        db.execSQL("CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                   + "NAME TEXT, "
                   + "DESCRIPTION TEXT, "
                   + "IMAGE_RESOURCE_ID INTEGER);");
        insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
        insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
                    R.drawable.cappuccino);
        insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }

    private static void insertDrink(SQLiteDatabase db, String name,
                                    String description, int resourceId) {
        ContentValues drinkValues = new ContentValues();
        drinkValues.put("NAME", name);
        drinkValues.put("DESCRIPTION", description);
        drinkValues.put("IMAGE_RESOURCE_ID", resourceId);
        db.insert("DRINK", null, drinkValues);
    }
}
```

You need to add this extra import statement.

Say what the database name and version is. It's the first version of the database, so the version should be 1.

onCreate() gets called when the database first gets created, so we're using it to create the table and insert data.

Create the DRINK table.

Insert each drink in a separate row.

onUpgrade() gets called when the database needs to be upgraded. We'll look at this next.

We need to insert several drinks, so we created a separate method to do this.

What the SQLite helper code does

1. **The user installs the app and launches it.**
When the app needs to access the database, the SQLite helper checks to see if the database already exists.

2. **If the database doesn't exist, it gets created.**
It's given the name and version number specified in the SQLite helper.

3. **When the database is created, the onCreate() method in the SQLite helper is called.**
It adds a DRINK table to the database, and populates it with records.

What if you need to make changes to the database?

So far, you've seen how to create a SQLite database that your app will be able to use to persist data. But what if you need to make changes to the database at some future stage?

As an example, suppose lots of users have already installed your Starbuzz app on their devices, and you want to a add a new FAVORITE column to the DRINK table. How would you distribute this change to new and existing users?

When you need to change an app's database, there are two key scenarios you have to deal with.

The first scenario is that the user has never installed your app before, and doesn't have the database installed on their device. In this case, the SQLite helper creates the database the first time the database needs to be accessed, and runs its `onCreate()` method.

The second scenario is where the user installs a new version of your app that includes a different version of the database. If the SQLite helper spots that the database that's installed is out of date, it will call either the `onUpgrade()` or `onDowngrade()` method.

So how can the SQLite helper tell if the database is out of date?

SQLite databases have a version number

The SQLite helper can tell whether the SQLite database needs updating by looking at its version number. You specify the version of the database in the SQLite helper by passing it to the `SQLiteOpenHelper` superclass in its constructor.

Earlier on, we specified the version number of the database like this:

```
...
    private static final String DB_NAME = "starbuzz";
    private static final int DB_VERSION = 1;

    StarbuzzDatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }
...
```

When the database gets created, its version number gets set to the version number in the SQLite helper, and the SQLite helper `onCreate()` method gets called.

When you want to update the database, you change the version number in the SQLite helper code. To *upgrade* the database, specify a number that's larger than you had before, and to *downgrade* your database, specify a number that's lower:

```
...
    private static final int DB_VERSION = 2;
...
```

Here we're increasing the version number, so the database will get upgraded.

Most of the time, you'll want to upgrade the database, so specify a number that's larger. You usually only downgrade your database when you want to undo changes you made in a previous upgrade.

When the user installs the latest version of the app on their device, the first time the app needs to use the database, the SQLite helper checks its version number against that of the database on the device.

If the version number in the SQLite helper code is ***higher*** than that of the database, it calls the SQLite helper **`onUpgrade()`** method. If the version number in the SQLite helper code is ***lower*** than that of the database, it calls the **`onDowngrade()`** method instead.

Once it's called either of these methods, it changes the version number of the database to match the version number in the SQLite helper.

Geek Bits

SQLite databases support a version number that's used by the SQLite helper, and an internal schema version. Whenever a change is made to the database schema, such as the table structure, the database increments the schema version by 1. You have no control over this value, it's just used internally by SQLite.

What happens when you change the version number

Let's look at what happens when you release a new version of the app where you've changed the SQLite helper version number from 1 to 2. We'll consider two scenarios: where a first-time user installs the app, and when an existing user installs it.

Scenario 1: A first-time user installs the app

1. **The first time the user runs the app, the database doesn't exist, so the SQLite helper creates it.**
   The SQLite helper gives the database the name and version number specified in the SQLite helper code.

2. **When the database is created, the onCreate() method in the SQLite helper is called.**
   The `onCreate()` method includes code to populate the database.

That's what happens when a first-time user installs the app. What about when an existing user installs the new version?

Scenario 2: an existing user installs the new version

1. **When the user runs the new version of the app, the database helper checks whether the database exists.**
   If the database already exists, the SQLite helper doesn't recreate it.

2. **The SQLite helper checks the version number of the existing database against the version number in the SQLite helper code.**
   If the SQLite helper version number is higher than the database version, it calls the `onUpgrade()` method. If the SQLite helper version number is lower than the database version, it calls the `onDowngrade()` method. It then changes the database version number to reflect the version number in the SQLite helper code.

Now that you've seen under what circumstances the `onUpgrade()` and `onDowngrade()` methods get called, let's find out more about how you use them.

Upgrade your database with onUpgrade()

The `onUpgrade()` method has three parameters—the SQLite database, the user's version number of the database, and the new version of the database that's passed to the `SQLiteOpenHelper` superclass:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    //Your code goes here
}
```

The user's version of the database, which is out of date

The new version described in the SQlite helper code

Remember, to upgrade the database, the new version must be higher than the user's existing version.

The version numbers are important, as you can use them to say what database changes should be made depending on which version of the database the user already has. As an example, suppose you needed to run code if the user has version 1 of the database, and the SQLite helper version number is higher. Your code would look like this:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion == 1) {
        //Code to run if the database version is 1
    }
}
```

This code will only run if the user has version 1 of the database, and the SQLite helper version number is higher.

You can also use the version numbers to apply successive updates like this:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion == 1) {
        //Code to run if the database version is 1
    }
    if (oldVersion < 3) {
        //Code to run if the database version is 1 or 2
    }
}
```

This code will only run if the user's database is at version 1.

This code will run if the user's database is at version 1 or 2.

Using this approach means that you can make sure that the user gets all the database changes applied that they need, irrespective of which version they have installed.

The `onDowngrade()` method works in a similar way to the `onUpgrade()` method. Let's take a look on the next page.

Downgrade your database with onDowngrade()

The `onDowngrade()` method isn't used as often as the `onUpgrade()` method, as it's used to revert your database to a previous version. This can be useful if you release a version of your app that includes database changes, but you then discover that there are bugs. The `onDowngrade()` method allows you to undo the changes and set the database back to its previous version.

Just like the `onUpgrade()` method, the `onDowngrade()` method has three parameters—the SQLite database you want to downgrade, the version number of the database itself, and the new version of the database that's passed to the `SQLiteOpenHelper` superclass:

```
@Override
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    //Your code goes here
}
```

To downgrade the database, the new version must be lower than the user's exisiting version.

Just as with the `onUpgrade()` method, you can use the version numbers to revert changes specific to a particular version. As an example, if you needed to make changes to the database when the user's database version number is 3, you'd use code like following:

```
@Override
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion == 3) {
        //Code to run if the database version is 3
    }
}
```

This code will run if the user has version 3 of the database, but you want to downgrade it to a lower version.

Now that you know how to upgrade and downgrade a database, let's walk through the more common scenario: upgrading.

Upgrade database

Let's upgrade the database

Suppose we need to upgrade our database to add a new column to the DRINK table. As we want all new and existing users to get this change, we need to make sure that it's included in both the `onCreate()` and `onUpgrade()` methods. The `onCreate()` method will make sure that all new users get the new column, and the `onUpgrade()` method will make sure that all existing users get it too.

Rather than put similar code in both the `onCreate()` and `onUpgrade()` methods, we're going to create a separate `updateMyDatabase()` method, called by both the `onCreate()` and `onUpgrade()` methods. We'll move the code that's currently in the `onCreate()` method to this new `updateMyDatabase()` method, and we'll add extra code to create the extra column. Using this approach means that you can put all of your database code in one place, and more easily keep track of what changes you've made each time you've updated the database.

Here's the full code for *StarbuzzDatabaseHelper.java* (update your code to reflect our changes):

```
package com.hfad.starbuzz;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper{

    private static final String DB_NAME = "starbuzz"; // the name of our database
    private static final int DB_VERSION = 12; // the version of the database

    StarbuzzDatabaseHelper(Context context){
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        updateMyDatabase(db, 0, DB_VERSION);
    }
```

Changing the version number to a larger integer means the SQLite helper will know that you want to upgrade the database.

Replace the existing onCreate() code with this call to updateMyDatabase().

The code continues on the next page.

The SQLite helper code (continued)

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        updateMyDatabase(db, oldVersion, newVersion);
    }

    private static void insertDrink(SQLiteDatabase db, String name,
                                    String description, int resourceId) {
        ContentValues drinkValues = new ContentValues();
        drinkValues.put("NAME", name);
        drinkValues.put("DESCRIPTION", description);
        drinkValues.put("IMAGE_RESOURCE_ID", resourceId);
        db.insert("DRINK", null, drinkValues);
    }

    private void updateMyDatabase(SQLiteDatabase db, int oldVersion, int newVersion) {
        if (oldVersion < 1) {
            db.execSQL("CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                       + "NAME TEXT, "
                       + "DESCRIPTION TEXT, "
                       + "IMAGE_RESOURCE_ID INTEGER);");
            insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
            insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
                        R.drawable.cappuccino);
            insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
        }
        if (oldVersion < 2) {
            //Code to add the extra column
        }
    }
}
```

Call the updateMyDatabase() method from onUpgrade(), passing along the parameters.

This is the code we previously had in our onCreate() method.

This code will run if the user already has version 1 of the database. We need to write this code next.

The next thing we need to do is write the code to upgrade the database. Before we do that, try the exercise on the next page.

BE the SQLite Helper

On the right, you'll see some SQLite helper code. Your job is to play like you're the SQLite helper and say which code will run for each of the users below. We've labeled the code we want you to consider. We've done the first one to start you off.

```
...

class MyHelper extends SQLiteOpenHelper{

    StarbuzzDatabaseHelper(Context context){
        super(context, "fred", null, 4);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
     A  //Run code A
        ...
    }

    @Override
    public void onUpgrade(SQLiteDatabase db,
                          int oldVersion,
                          int newVersion){
        if (oldVersion < 2) {
         B  //Run code B
            ...
        }
        if (oldVersion == 3) {
         C  //Run code C
            ...
        }
     D  //Run code D
        ...
    }

    @Override
    public void onDowngrade(SQLiteDatabase db,
                            int oldVersion,
                            int newVersion){
        if (oldVersion == 3) {
         E  //Run code E
            ...
        }
        if (oldVersion < 6) {
         F  //Run code F
            ...
        }
    }
}
```

User 1 runs the app for the first time.

Code segment A. The user doesn't have the database, so the onCreate() method runs.

User 2 has database version 1.

User 3 has database version 2.

User 4 has database version 3.

User 5 has database version 4.

User 6 has database version 5.

Answers on page 654.

Upgrade an existing database

When you need to upgrade your database, there are two types of actions you might want to perform:

- **Change the database records.**
  Earlier in the chapter, you saw how to insert records in your database using the `SQLiteDatabase insert()` method. You may want to add more records when you upgrade the database, or update or delete the records that are already there.

- **Change the database structure.**
  You've already seen how you can create tables in the database. You may also want to add columns to existing tables, rename tables, or remove tables completely.

We'll start by looking at how you change database records.

How to update records

You update records in a table in a similar way to how you insert them.

You start by creating a new `ContentValues` object that specifies what you want to update values to. As an example, suppose you wanted to update the Latte data in the DRINK table so that the value of the DESCRIPTION field is "Tasty":

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	"Latte"	~~"Espresso and steamed milk"~~ "Tasty"	54543543

To do this, you'd create a new `ContentValues` object that describes the data that needs to be updated:

```
ContentValues drinkValues = new ContentValues();
drinkValues.put("DESCRIPTION", "Tasty");
```

We want to change the value of the DESCRIPTION column to Tasty, so we give the name "DESCRIPTION" a value of "Tasty".

Notice that when you're updating records, you only need to specify the data you want to change in the `ContentValues` object, not the entire row of data.

Once you've added the data you want to change to the `ContentValues` object, you use the `SQLiteDatabase update()` method to update the data. We'll look at this on the next page.

Update records with the update() method

The `update()` method lets you update records in the database, and returns the number of records it's updated. To use the `update()` method, you specify the table you want to update records in, the `ContentValues` object that contains the values you want to update, and the conditions for updating them.

As an example, here's how you'd change the value of the DESCRIPTION column to "Tasty" where the name of the drink is "Latte":

```
ContentValues drinkValues = new ContentValues();
drinkValues.put("DESCRIPTION", "Tasty");
db.update("DRINK",
          drinkValues,
          "NAME = ?",
          new String[] {"Latte"});
```

This is the name of the table whose records you want to update.

This is the ContentValues object that contains the new values.

The conditions for updating the data, in this case where NAME = "Latte".

The value "Latte" is substituted for the ? in the "NAME = ?" statement above.

The first parameter of the `update()` method is the name of the table you want to update (in this case, the DRINK table).

The second parameter is the `ContentValues` object that describes the values you want to update. In the example above, we've added `"DESCRIPTION"` and `"Tasty"` to the `ContentValues` object, so it updates the value of the DESCRIPTION column to "Tasty".

The last two parameters specify which records you want to update by describing conditions for the update. Together, they form the `WHERE` clause of a SQL statement.

The third parameter specifies the name of the column in the condition. In the above example, we want to update records where the value of the NAME column is "Latte", so we use `"NAME = ?"`; it means that we want the value in the NAME column to be equal to some value. The `?` symbol is a placeholder for this value.

The last parameter is an array of Strings that says what the value of the condition should be. In our particular case we want to update records where the value of the NAME column is "Latte", so we use:

```
new String[] {"Latte"});
```

We'll look at more complex conditions on the next page.

If you set the last two parameters of the `update()` method to `null`, ALL records in the table will be updated.

As an example, the code:

```
db.update("DRINK",
          drinkValues,
          null, null);
```

will update all records in the DRINK table.

Apply conditions to multiple columns

You can also specify conditions that apply to multiple columns. As an example, here's how you'd update all records where the name of the drink is "Latte", or the drink description is "Our best drip coffee":

```
db.update("DRINK",
          drinkValues,
          "NAME = ? OR DESCRIPTION = ?",
          new String[] {"Latte", "Our best drip coffee"});
```

This means: Where NAME = "Latte" or DESCRIPTION = "Our best drip coffee".

Each ? placeholder is replaced with a value from this array. The number of values in the array must match the number of ?'s.

If you want to apply conditions that cover multiple columns, you specify the column names in the `update()` method's third parameter. As before, you add a placeholder `?` for each value you want to add as part of each condition. You then specify the condition values in the `update()` method's fourth parameter.

The condition values must be Strings, even if the column you're applying the condition to contains some other type of data. As an example, here's how you'd update DRINK records where the `_id` (numeric) is `1`:

```
db.update("DRINK",
          drinkValues,
          "_id = ?",
          new String[] {Integer.toString(1)});
```

Convert the int 1 to a String value.

Delete records with the delete() method

You delete records using the `SQLiteDatabase` `delete()` method. It works in a similar way to the `update()` method you've just seen. You specify the table you want to delete records from, and conditions for deleting the data. As an example, here's how you'd delete all records from the DRINK table where the name of the drink is "Latte":

```
db.delete("DRINK",
          "NAME = ?",
          new String[] {"Latte"});
```

See how similar this is to the update() method.

The entire row is deleted.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
~~1~~	~~"Latte"~~	~~"Espresso and steamed milk"~~	~~54543543~~

The first parameter is the name of the table you want to delete records from (in this case, DRINK). The second and third arguments allow you to describe conditions to specify exactly which records you wish to delete (in this case, where NAME = "Latte").

Sharpen your pencil

Here's the `onCreate()` method of a `SQLiteOpenHelper` class. Your job is to say what values have been inserted into the NAME and DESCRIPTION columns of the DRINK table when the `onCreate()` method has finished running.

```
@Override
public void onCreate(SQLiteDatabase db) {
    ContentValues espresso = new ContentValues();
    espresso.put("NAME", "Espresso");
    ContentValues americano = new ContentValues();
    americano.put("NAME", "Americano");
    ContentValues latte = new ContentValues();
    latte.put("NAME", "Latte");
    ContentValues filter = new ContentValues();
    filter.put("DESCRIPTION", "Filter");
    ContentValues mochachino = new ContentValues();
    mochachino.put("NAME", "Mochachino");

    db.execSQL("CREATE TABLE DRINK ("
               + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
               + "NAME TEXT, "
               + "DESCRIPTION TEXT);");
    db.insert("DRINK", null, espresso);
    db.insert("DRINK", null, americano);
    db.delete("DRINK", null, null);
    db.insert("DRINK", null, latte);
    db.update("DRINK", mochachino, "NAME = ?", new String[] {"Espresso"});
    db.insert("DRINK", null, filter);
}
```

You don't need to enter the value of the _id column.

_id	NAME	DESCRIPTION

Answers on page 655.

Change the database structure

In addition to creating, updating, or deleting database records, you may want to make changes to the database structure. As an example, in our particular case we want to add a new FAVORITE column to the DRINK table.

Add new columns to tables using SQL

Earlier in the chapter, you saw how you could create tables using the SQL `CREATE TABLE` command like this:

```
CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    NAME TEXT,
                    DESCRIPTION TEXT,
                    IMAGE_RESOURCE_ID INTEGER)
```

The _id column is the primary key.

The table name

The table columns

You can also use SQL to change an existing table using the `ALTER TABLE` command. As an example, here's what the command looks like to add a column to a table:

```
ALTER TABLE DRINK
ADD COLUMN FAVORITE NUMERIC
```

The table name

The column you want to add

In the example above, we're adding a column to the DRINK table called FAVORITE that holds numeric values.

Renaming tables

You can also use the `ALTER TABLE` command to rename a table. As an example, here's how you'd rename the DRINK table to FOO:

```
ALTER TABLE DRINK
RENAME TO FOO
```

The current table name

The new name of the table

On the next page, we'll show you how to remove a table from the database.

Delete tables by dropping them

If you want to delete a table from the database, you use the `DROP TABLE` command. As an example, here's how you'd delete the DRINK table:

```
DROP TABLE DRINK
```

← The name of the table you want to remove

This command is useful if you have a table in your database schema that you know you don't need anymore, and want to remove it in order to save space. Make sure you only use the `DROP TABLE` command if you're absolutely sure you want to get rid of the table and all of its data.

Execute the SQL using execSQL()

As you saw earlier in the chapter, you execute SQL commands using the `SQLiteDatabase` `execSQL()` method:

```
SQLiteDatabase.execSQL(String sql);
```

You use the `execSQL()` method any time you need to execute SQL on the database. As an example, here's how you'd execute SQL to add a new FAVORITE column to the DRINK table:

```
db.execSQL("ALTER TABLE DRINK ADD COLUMN FAVORITE NUMERIC;");
```

Now that you've seen the sorts of actions you might want to perform when upgrading your database, let's apply this to *StarbuzzDatabaseHelper.java*.

The full SQLite helper code

Here's the full code for *StarbuzzDatabaseHelper.java* that will add a new FAVORITE column to the DRINK table. Update your code to match ours (the changes are in bold):

```
package com.hfad.starbuzz;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper{

    private static final String DB_NAME = "starbuzz"; // the name of our database
    private static final int DB_VERSION = 2; // the version of the database

    StarbuzzDatabaseHelper(Context context){
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        updateMyDatabase(db, 0, DB_VERSION);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        updateMyDatabase(db, oldVersion, newVersion);
    }
```

Changing the version number to a larger integer enables the SQLite helper to spot that you want to upgrade the database.

The code to create any database tables is in the updateMyDatabase() method.

The code to upgrade the database is in our updateMyDatabase() method.

The code continues over the page.

Upgrade database

The SQLite helper code (continued)

```
    private static void insertDrink(SQLiteDatabase db, String name,
                                    String description, int resourceId) {
        ContentValues drinkValues = new ContentValues();
        drinkValues.put("NAME", name);
        drinkValues.put("DESCRIPTION", description);
        drinkValues.put("IMAGE_RESOURCE_ID", resourceId);
        db.insert("DRINK", null, drinkValues);
    }

    private void updateMyDatabase(SQLiteDatabase db, int oldVersion, int newVersion) {
        if (oldVersion < 1) {
            db.execSQL("CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                       + "NAME TEXT, "
                       + "DESCRIPTION TEXT, "
                       + "IMAGE_RESOURCE_ID INTEGER);");
            insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
            insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
                        R.drawable.cappuccino);
            insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
        }
        if (oldVersion < 2) {
            db.execSQL("ALTER TABLE DRINK ADD COLUMN FAVORITE NUMERIC;");
        }
    }
}
```

Add a numeric FAVORITE column to the DRINK table.

The new code in the SQLite helper means that existing users will get the new FAVORITE column added to the DRINK table the next time they access the database. It also means that any new users will get the complete database created for them, including the new column.

We'll go through what happens when the code runs on the next page. Then in the next chapter, you'll see how to use the database data in your activities.

What happens when the code runs

1. **When the database first needs to be accessed, the SQLite helper checks whether the database already exists.**

2a. **If the database doesn't exist, the SQLite helper creates it and runs its onCreate() method.**
Our `onCreate()` method code calls the `updateMyDatabase()` method. This creates the DRINK table (including the new FAVORITE column) and populates the table with records.

2b. **If the database already exists, the SQLite helper checks the version number of the database against the version number in the SQLite helper code.**
If the SQLite helper version number is higher than the database version, it calls the `onUpgrade()` method. If the SQLite helper version number is lower than the database version, it calls the `onDowngrade()` method. Our SQLite helper version number is higher than that of the existing database, so the `onUpgrade()` method is called. It calls the `updateMyDatabase()` method, and this adds the new FAVORITE column to the DRINK table.

BE the SQLite Helper Solution

On the right you'll see some SQLite helper code. Your job is to play like you're the SQLite helper and say which code will run for each of the users below. We've labeled the code we want you to consider. We've done the first one to start you off.

```
...

class MyHelper extends SQLiteOpenHelper{

    StarbuzzDatabaseHelper(Context context){
        super(context, "fred", null, 4);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
      A //Run code A
        ...
    }

    @Override
    public void onUpgrade(SQLiteDatabase db,
                          int oldVersion,
                          int newVersion){
        if (oldVersion < 2) {
          B //Run code B
            ...
        }
        if (oldVersion == 3) {
          C //Run code C
            ...
        }
      D //Run code D
        ...
    }

    @Override
    public void onDowngrade(SQLiteDatabase db,
                            int oldVersion,
                            int newVersion){
        if (oldVersion == 3) {
          E //Run code E
            ...
        }
        if (oldVersion < 6) {
          F //Run code F
            ...
        }
    }
}
```

The new version of the database is 4.

The onCreate() method will only run if the user doesn't have the database.

This will run if the user has version 1.

This will run if the user has version 3.

This will run if the user has version 1, 2, or 3.

This will never run. If the user has version 3, their database needs to be upgraded, not downgraded.

This will run if the user has version 5. For onDowngrade() to run, the user must have a version greater than 4, as that's the current version number of the helper.

User 1 runs the app for the first time.

Code segment A. The user doesn't have the database, so the onCreate() method runs.

User 2 has database version 1.

Code segment B, then D. The database needs to be upgraded with oldVersion == 1.

User 3 has database version 2.

Code segment D. The database needs to be upgraded with oldVersion == 2.

User 4 has database version 3.

Code segment C then D. The database needs to be upgraded with oldVersion == 3.

User 5 has database version 4.

None. The user has the correct version of the database.

User 6 has database version 5.

Code segment F. The database needs to be downgraded with oldVersion == 5.

Sharpen your pencil Solution

Here's the `onCreate()` method of a `SQLiteOpenHelper` class. Your job is to say what values have been inserted into the NAME and DESCRIPTION columns of the DRINK table when the `onCreate()` method has finished running.

```
@Override
public void onCreate(SQLiteDatabase db) {
    ContentValues espresso = new ContentValues();
    espresso.put("NAME", "Espresso");
    ContentValues americano = new ContentValues();
    americano.put("NAME", "Americano");
    ContentValues latte = new ContentValues();
    latte.put("NAME", "Latte");
    ContentValues filter = new ContentValues();
    filter.put("DESCRIPTION", "Filter");
    ContentValues mochachino = new ContentValues();
    mochachino.put("NAME", "Mochachino");

    db.execSQL("CREATE TABLE DRINK (" ← Create the table, adding _id, NAME, and DESCRIPTION columns.
               + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
               + "NAME TEXT, "
               + "DESCRIPTION TEXT);");
    db.insert("DRINK", null, espresso); ← Insert Espresso in the NAME column.
    db.insert("DRINK", null, americano); ← Insert Americano in the NAME column.
    db.delete("DRINK", null, null); ← Delete all the drinks.
    db.insert("DRINK", null, latte); ← Insert Latte in the NAME column.
    db.update("DRINK", mochachino, "NAME = ?", new String[] {"Espresso"});
    db.insert("DRINK", null, filter);
}
```

Set NAME to Mochachino where NAME is Espresso. No records get updated.

Insert Filter in the DESCRIPTION column.

Here's the final result of running the above code.

_id	NAME	DESCRIPTION
	Latte	
		Filter

Your Android Toolbox

You've got Chapter 15 under your belt and now you've added creating, updating, and upgrading databases to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- Android uses SQLite as its backend database to persist data.
- The `SQLiteDatabase` class gives you access to the SQLite database.
- A SQLite helper lets you create and manage SQLite databases. You create a SQLite helper by extending the `SQLiteOpenHelper` class.
- You must implement the `SQLiteOpenHelper` `onCreate()` and `onUpgrade()` methods.
- The database gets created the first time it needs to be accessed. You need to give the database a name and version number, starting at 1. If you don't give the database a name, it will just get created in memory.
- The `onCreate()` method gets called when the database first gets created.
- The `onUpgrade()` method gets called when the database needs to be upgraded.
- Execute SQL using the `SQLiteDatabase` `execSQL(String)` method.
- Use the SQL `ALTER TABLE` command to change an existing table. Use `RENAME TO` to rename the table, and `ADD COLUMN` to add a column.
- Use the SQL `DROP TABLE` command to delete a table.
- Add records to tables using the `insert()` method.
- Update records using the `update()` method.
- Remove records from tables using the `delete()` method.

16 basic cursors

Getting Data Out

Charles gave me a cursor that returned everything from the EXPENSIVE_GIFT table.

So how do you connect your app to a SQLite database?

So far you've seen how to create a SQLite database using a SQLite helper. The next step is to get your activities to access it. In this chapter, we'll focus on how you read data from a database. You'll find out **how to use cursors to get data from the database**. You'll see **how to navigate cursors,** and **how to get access to their data**. Finally, you'll discover how to use **cursor adapters** to bind cursors to list views.

The story so far...

In Chapter 15, you created a SQLite helper for Starbuzz Coffee. The SQLite helper creates a Starbuzz database, adds a DRINK table to it, and populates the table with drinks.

The activities in the Starbuzz app currently get their data from the Java `Drink` class. We're going to change the app so the activities get data from the SQLite database instead.

Here's a reminder of how the app currently works:

1. **TopLevelActivity displays a list of options for Drinks, Food, and Stores.**
2. **When the user clicks on the Drinks option, it launches DrinkCategoryActivity.**
   This activity displays a list of drinks that it gets from the Java `Drink` class.
3. **When the user clicks on a drink, its details get displayed in DrinkActivity.**
   `DrinkActivity` gets details of the drink from the Java `Drink` class.

The new Starbuzz app structure

There are two activities that use the `Drink` class: `DrinkActivity` and `DrinkCategoryActivity`. We need to change these activities so that they read data from the SQLite database with assistance from the SQLite helper.

Here's what the new structure of the Starbuzz app will look like:

We'll start by updating `DrinkActivity`, and we'll change `DrinkCategoryActivity` later on in the chapter.

Do this!

We're going to update the Starbuzz app in this chapter, so open your Starbuzz project in Android Studio.

What we'll do to change DrinkActivity to use the Starbuzz database

There are a number of steps we need to go through to change `DrinkActivity` so that it uses the Starbuzz database:

1. **Get a reference to the Starbuzz database.**
   We'll do this using the Starbuzz SQLite helper we created in Chapter 15.

2. **Create a cursor to read drink data from the database.**
   We need to read the data held in the Starbuzz database for the drink the user selects in `DrinkCategoryActivity`. The cursor will give us access to this data. (We'll explain cursors soon.)

3. **Navigate to the drink record.**
   Before we can use the data retrieved by the cursor, we need to explicitly navigate to it.

4. **Display details of the drink in DrinkActivity.**
   Once we've navigated to the drink record in the cursor, we need to read the data and display it in `DrinkActivity`.

Before we begin, let's remind ourselves of the *DrinkActivity.java* code we created in Chapter 7.

Database reference
Create cursor
Navigate to record
Display drink

The current DrinkActivity code

Below is a reminder of the current *DrinkActivity.java* code. The `onCreate()` method gets the drink ID selected by the user, gets the details of that drink from the `Drink` class, and then populates the activity's views using the drink attributes. We need to change the code in the `onCreate()` method to get the data from the Starbuzz database.

```
package com.hfad.starbuzz;

...
```

We're not showing you the import statements.

```
public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKID = "drinkId";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

        //Get the drink from the intent
        int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
        Drink drink = Drink.drinks[drinkId];

        //Populate the drink name
        TextView name = (TextView)findViewById(R.id.name);
        name.setText(drink.getName());

        //Populate the drink description
        TextView description = (TextView)findViewById(R.id.description);
        description.setText(drink.getDescription());

        //Populate the drink image
        ImageView photo = (ImageView)findViewById(R.id.photo);
        photo.setImageResource(drink.getImageResourceId());
        photo.setContentDescription(drink.getName());
    }
}
```

This is the drink the user selected.

Use the drink ID from the intent to get the drink details from the Drink class. We'll need to change this so the drink comes from the database.

We need to populate the views in the layout with values from the database, not from the Drink class.

Get a reference to the database

The first thing we need is to get a reference to the Starbuzz database using the SQLite helper we created in the last chapter. To do that, you start by getting a reference to the SQLite helper:

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
```

This is a Context, in this case the current activity.

You then call the SQLite helper's `getReadableDatabase()` or `getWritableDatabase()` to get a reference to the database. You use the `getReadableDatabase()` method if you need read-only access to the database, and the `getWritableDatabase()` method if you need to perform any updates. Both methods return a `SQLiteDatabase` object that you can use to access the database:

```
SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
```

or:

```
SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
```

If Android fails to get a reference to the database, a `SQLiteException` is thrown. This can happen if, for example, you call the `getWritableDatabase()` to get read/write access to the database, but you can't write to the database because the disk is full.

In our particular case, we only need to read data from the database, so we'll use the `getReadableDatabase()` method. If Android can't get a reference to the database and a `SQLiteException` is thrown, we'll use a `Toast` (a pop-up message) to tell the user that the database is unavailable:

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
try {
    SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
    //Code to read data from the database
} catch(SQLiteException e) {
    Toast toast = Toast.makeText(this,
                                 "Database unavailable",
                                 Toast.LENGTH_SHORT);
    toast.show();
}
```

This line displays the toast.

These lines create a Toast that will display the message "Database unavailable" for a few seconds.

Once you have a reference to the database, you can get data from it using a cursor. We'll look at cursors next.

- Database reference
- **Create cursor**
- Navigate to record
- Display drink

Get data from the database with a cursor

As we said in Chapter 15, a **cursor** lets you read from and write to the database. You specify what data you want access to, and the cursor brings back the relevant records from the database. You can then navigate through the records supplied by the cursor.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	“Latte”	"Espresso and steamed milk"	54543543
2	“Cappuccino”	"Espresso, hot milk and steamed-milk foam"	654334453
3	“Filter”	"Our best drip coffee"	44324234

You create a cursor using a **database query**. A database query lets you specify which records you want access to from the database. As an example, you can say you want to access all the records from the DRINK table, or just one particular record. These records are then returned in the cursor.

You create a cursor using the `SQLiteDatabase` `query()` method:

The query() method returns a Cursor object.

```
Cursor cursor = db.query(...);
```

The query() method parameters go here. We'll look at them over the next few pages.

There are many overloaded versions of this method with different parameters, so rather than go into each variation, we're only going to show you the most common ways of using it.

Return all the records from a table

Database reference
Create cursor
Navigate to record
Display drink

The simplest type of database query is one that returns all the records from a particular table in the database. This is useful if, for instance, you want to display all the records in a list in an activity. As an example, here's how you'd return the values in the _id, NAME, and DESCRIPTION columns for every record in the DRINK table:

This is the name of the table.

We want to return the values in these columns.

```
Cursor cursor = db.query("DRINK",
                         new String[] {"_id","NAME", "DESCRIPTION"},
                         null, null, null, null, null);
```

Set these parameters to null when you want to return all records from a table.

The query returns all the data from the _id, NAME, and DESCRIPTION columns in the DRINK table.

_id	NAME	DESCRIPTION
1	"Latte"	"Espresso and steamed milk"
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"
3	"Filter"	"Our best drip coffee"

To return all the records from a particular table, you pass the name of the table as the `query()` method's first parameter, and a String array of the column names as the second. You set all of the other parameters to `null`, as you don't need them for this type of query.

You specify each column whose value you want to return in an array of Strings.

```
Cursor cursor = db.query("TABLE_NAME",
                         new String[] {"COLUMN1","COLUMN2"},
                         null, null, null, null, null);
```

There are five extra parameters you need to set to null.

Next we'll look at how you can return the records in a particular order.

Behind the scenes, Android uses the query() method to construct an SQL SELECT statement.

Database reference
Create cursor
Navigate to record
Display drink

Return records in a particular order

If you want to display data in your app in a particular order, you use the query to sort the data by a particular column. This can be useful if, for example, you want to display drink names in alphabetical order.

By default, the data in the table appears in _id order, as this was the order in which data was entered:

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID	FAVORITE
1	"Latte"	"Espresso and steamed milk"	54543543	0
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453	0
3	"Filter"	"Our best drip coffee"	44324234	1

If you wanted to retrieve data from the _id, NAME, and FAVORITE column in ascending NAME order, you would use the following:

```
Cursor cursor = db.query("DRINK",
          new String[] {"_id", "NAME", "FAVORITE"},
          null, null, null, null,
          "NAME ASC");
```

Order by NAME in ascending order.

_id	NAME	FAVORITE
2	"Cappuccino"	0
3	"Filter"	1
1	"Latte"	0

The ASC keyword means that you want to order that column in ascending order. Columns are ordered in ascending order by default, so if you want you can omit the ASC. To order the data in descending order instead, you'd use DESC.

You can sort by multiple columns too. As an example, here's how you'd order by FAVORITE in descending order, followed by NAME in ascending order:

```
Cursor cursor = db.query("DRINK",
          new String[] {"_id", "NAME", "FAVORITE"},
          null, null, null, null,
          "FAVORITE DESC, NAME");
```

Order by FAVORITE in descending order, then NAME in ascending order.

_id	NAME	FAVORITE
3	"Filter"	1
2	"Cappuccino"	0
1	"Latte"	0

Next we'll look at how you return selected records from the database.

Database reference
Create cursor
Navigate to record
Display drink

Return selected records

You can filter your data by declaring conditions the data must meet, just as you did in Chapter 15. As an example, here's how you'd return records from the DRINK table where the name of the drink is "Latte":

```
Cursor cursor = db.query("DRINK",
                         new String[] {"_id", "NAME", "DESCRIPTION"},
                         "NAME = ?",
                         new String[] {"Latte"},
                         null, null, null);
```

These are the columns we want to return.

We want to return records where the value of the NAME column is "Latte".

The third and fourth parameters in the query describe the conditions the data must meet.

The third parameter specifies the column in the condition. In the above example we want to return records where the value of the NAME column is "Latte", so we use `"NAME = ?"`. We want the value in the NAME column to be equal to some value, and the `?` symbol is a placeholder for this value.

The fourth parameter is an array of Strings that specifies what the value of the condition should be. In the above example we want to update records where the value of the NAME column is "Latte", so we use:

```
new String[] {"Latte"};
```

_id	NAME	DESCRIPTION
1	"Latte"	"Espresso and steamed milk"

The query returns all the data from the NAME and DESCRIPTION columns in the DRINK table where the value of the NAME column is "Latte".

The value of the condition must be an array of Strings, even if the column you're applying the condition to contains some other type of data. As an example, here's how you'd return records from the DRINK table where the drink _id is `1`:

```
Cursor cursor = db.query("DRINK",
                         new String[] {"_id", "NAME", "DESCRIPTION"},
                         "_id = ?",
                         new String[] {Integer.toString(1)},
                         null, null, null);
```

Convert the int 1 to a String value.

You've now seen the most common ways of using the `query()` method to create a cursor, so try the following exercise to construct the cursor we need for *DrinkActivity.java*.

To find out more ways of using the query() method, visit the SQLiteDatabase documentation:

https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

Code Magnets

In our code for `DrinkActivity`, we want to get the name, description, and image resource ID for the drink ID passed to it in an intent. Can you create a cursor that will do that?

```
...
int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);

//Create a cursor
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
try {
    SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();

    Cursor cursor = db.query(..............,

            new String[] {..........................., ........................., ....................................},

            "                          ",
             ............................",

            new String[] {.......................................................................},

         null, null, null);
} catch(SQLiteException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}
...
```

You won't need to use all of the magnets.

Code Magnets Solution

In our code for `DrinkActivity` we want to get the name, description, and image resource ID for the drink passed to it in an intent. Can you create a cursor that will do that?

```
...
int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);

//Create a cursor
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
try {
    SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();

    Cursor cursor = db.query("DRINK",
            new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
            "_id = ?",
            new String[] {Integer.toString(drinkId)},
        null, null, null);
} catch(SQLiteException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}
...
```

We want to access the DRINK table.

Get the NAME, DESCRIPTION, and IMAGE_RESOURCE_ID.

Where _id matches the drinkId

drinkId is an int, so needed to be converted to a String.

You didn't need to use this magnet.

`id`

- Database reference
- **Create cursor**
- Navigate to record
- Display drink

The DrinkActivity code so far

We want to change *DrinkActivity.java*'s `onCreate()` method so that `DrinkActivity` gets its drink data from the Starbuzz database instead of from the `Drink` Java class. Here's the code so far (we suggest you wait until we show you the *full* code—a few pages ahead—before you update your version of *DrinkActivity.java*):

```
package com.hfad.starbuzz;
...
public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKID = "drinkId";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

        //Get the drink from the intent
        int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
        Drink drink = Drink.drinks[drinkId];

        //Create a cursor
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
            Cursor cursor = db.query ("DRINK",
                    new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
                    "_id = ?",
                    new String[] {Integer.toString(drinkId)},
                    null, null, null);
        } catch(SQLiteException e) {
            Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
        ...
    }
}
```

Our Starbuzz code uses the Activity class, but we could have changed the code to use AppCompatActivity if we'd wanted to.

We'll add the code to the onCreate() method.

We're no longer getting the drinks from Drink.java.

Get a reference to the database.

Create a cursor to get the name, description, and image resource ID of the drink the user selected.

Display a pop-up message if a SQLiteException is thrown.

There's more code we haven't changed yet, but you don't need to see it right now.

Now that we've created our cursor, the next thing we need to do is get the drink name, description, and image resource ID from the cursor so that we can update `DrinkActivity`'s views.

To read a record from a cursor, you first need to navigate to it

- Database reference
- Create cursor
- **Navigate to record**
- Display drink

You've now seen how to create a cursor; you get a reference to a `SQLiteDatabase`, and then use its `query()` method to say what data you want the cursor to return. But that's not the end of the story—once you have a cursor, you need to read values from it.

Whenever you need to retrieve values from a particular record in a cursor, you first need to navigate to that record.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	"Latte"	"Espresso and steamed milk"	54543543
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453
3	"Filter"	"Our best drip coffee"	44324234

In our particular case, we have a cursor that's composed of a single record that contains details of the drink the user selected. We need to navigate to that record in order to read details of the drink.

Database reference
Create cursor
Navigate to record
Display drink

Navigate cursors

There are four main methods you use to navigate through the records in a cursor: `moveToFirst()`, `moveToLast()`, `moveToPrevious()`, and `moveToNext()`.

To go to the first record in a cursor, you use its `moveToFirst()` method. This method returns a value of `true` if it finds a record, and `false` if the cursor hasn't returned any records):

```
if (cursor.moveToFirst()) {
    //Do something
};
```

Move to the first row.

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

To go to the last record, you use the `moveToLast()` method. Just like the `moveToFirst()` method, it returns a value of `true` if it finds a record, and `false` if it doesn't:

```
if (cursor.moveToLast()) {
    //Do something
};
```

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

Move to the last row.

To move through the records in the cursor, you use the `moveToPrevious()` and `moveToNext()` methods.

The `moveToPrevious()` method moves you to the previous record in the cursor. It returns `true` if it succeeds in moving to the previous record, and `false` if it fails (for example, if it's already at the first record):

```
if (cursor.moveToPrevious()) {
    //Do something
};
```

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

Move to the previous row.

The `moveToNext()` method works in a similar way to the `moveToPrevious()` method, except that it moves you to the next record in the cursor instead:

```
if (cursor.moveToNext()) {
    //Do something
};
```

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

Move to the next row.

In our case, we want to read values from the first (and only) record in the cursor, so we'll use the `moveToFirst()` method to navigate to this record.

Once you've navigated to a record in your cursor, you can access its values. We'll look at how to do that next.

Database reference
Create cursor
Navigate to record
Display drink

Get cursor values

You retrieve values from a cursor's current record using the cursor's `get*()` methods: `getString()`, `getInt()`, and so on. The exact method you use depends on the type of value you want to retrieve. To get a String value, for example, you'd use the `getString()` method, and to get an int value you'd use `getInt()`. Each of the methods takes a single parameter: the index of the column whose value you want to retrieve, starting at 0.

As an example, we're using the following query to create our cursor:

```
Cursor cursor = db.query ("DRINK",
                          new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
                          "_id = ?",
                          new String[] {Integer.toString(1)},
                          null, null, null);
```

The cursor has three columns: NAME, DESCRIPTION, and IMAGE_RESOURCE_ID. The first two columns, NAME and DESCRIPTION, contain Strings, and the third column, IMAGE_RESOURCE_ID, contains int values.

These are the cursor's columns.

Column 0 Column 1 Column 2

NAME	DESCRIPTION	IMAGE_ RESOURCE_ID
"Latte"	"Espresso and steamed milk"	54543543

Suppose you wanted to get the value of the NAME column for the current record. NAME is the first column in the cursor, and contains String values. You'd therefore use the `getString()` method, passing it a parameter of `0` for the column index:

```
String name = cursor.getString(0);
```

← NAME is column 0 and contains Strings.

Similarly, suppose you wanted to get the contents of the IMAGE_RESOURCE_ID column. This has a column index of 2 and contains int values, so you'd use the code:

```
int imageResource = cursor.getInt(2);
```

← IMAGE_RESOURCE_ID is column 2 and contains ints.

Finally, close the cursor and the database

Once you've finished retrieving values from the cursor, you need to close the cursor and the database in order to release their resources. You do this by calling the cursor and database **`close()`** methods:

```
cursor.close();
db.close();
```

These lines close the cursor and the database.

You can find details of all the cursor get methods in http://developer.android.com/reference/android/database/Cursor.html.

We've now covered all the code you need to replace the code in `DrinkActivity` so that it gets its data from the Starbuzz database. Let's look at the revised code in full.

- Database reference
- Create cursor
- Navigate to record
- **Display drink**

The DrinkActivity code

Here's the full code for *DrinkActivity.java* (apply the changes in bold to your code, then save your work):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;

public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKID = "drinkId";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

        //Get the drink from the intent
        int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
        Drink drink = Drink.drinks[drinkId];

        //Create a cursor
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
            Cursor cursor = db.query ("DRINK",
                    new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
                    "_id = ?",
                    new String[] {Integer.toString(drinkId)},
                    null, null, null);
```

We're using these extra classes in the code.

This is the ID of the drink the user chose.

We're no longer getting our data from the drinks array, so we need to delete this line.

Create a cursor that gets the NAME, DESCRIPTION, and IMAGE_RESOURCE_ID data from the DRINK table where _id matches drinkId.

The code continues on the next page.

The DrinkActivity code (continued)

- Database reference
- Create cursor
- Navigate to record
- **Display drink**

```
        //Move to the first record in the Cursor
        if (cursor.moveToFirst()) {

            //Get the drink details from the cursor
            String nameText = cursor.getString(0);
            String descriptionText = cursor.getString(1);
            int photoId = cursor.getInt(2);

            //Populate the drink name
            TextView name = (TextView)findViewById(R.id.name);
            name.setText(drink.getName());
            name.setText(nameText);

            //Populate the drink description
            TextView description = (TextView)findViewById(R.id.description);
            description.setText(drink.getDescription());
            description.setText(descriptionText);

            //Populate the drink image
            ImageView photo = (ImageView)findViewById(R.id.photo);
            photo.setImageResource(drink.getImageResourceId());
            photo.setContentDescription(drink.getName());
            photo.setImageResource(photoId);
            photo.setContentDescription(nameText);
        }
        cursor.close();
        db.close();
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText(this,
                                     "Database unavailable",
                                     Toast.LENGTH_SHORT);
        toast.show();
    }
}
}
```

There's only one record in the cursor, but we still need to move to it.

The name of the drink is the first item in the cursor, the description is the second column, and the image resource ID is the third. That's because we told the cursor to use the NAME, DESCRIPTION, and IMAGE_RESOURCE_ID columns from the database in that order.

Set the drink name to the value from the database.

Use the drink description from the database.

Set the image resource ID and description to the values from the database.

Close the cursor and database.

If a SQLiteException is thrown, this means there's a problem with the database. In this case, we'll use a toast to display a message to the user.

So that's the complete `DrinkActivity` code. Let's review where we've got to, and what we need to do next.

Connecting your activities to a database takes more code than using a Java class.

But if you take your time working through the code in this chapter, you'll be fine.

What we've done so far

Now that we've finished updating the *DrinkActivity.java* code, let's look at the app structure diagram to see what we've done, and what we need to do next.

`DrinkActivity` now gets all of its drink data from the Starbuzz database. Next, we need to update the code in `DrinkCategoryActivity` so that it uses data from the database rather than from the Java `Drink` class. We'll look at the steps to do this on the next page.

there are no Dumb Questions

Q: How much SQL do I need to know to create cursors?

A: It's useful to have an understanding of SQL `SELECT` statements, as behind the scenes the `query()` method translates to one. In general, your queries probably won't be too complex, but SQL knowledge is a useful skill.

If you want to learn more about SQL, we suggest getting a copy of *Head First SQL* by Lynn Beighley.

Q: You said that if the database can't be accessed, a `SQLiteException` is thrown. How should I deal with it?

A: First, check the exception details. The exception might be caused by an error in SQL syntax, which you can then rectify.

How you handle the exception depends on the impact it has on your app. As an example, if you can get read access to the database but can't write to it, you can still give the user read-only access to the database, but you might want to tell the user that you can't save their changes. Ultimately, it all depends on your app.

What we'll do to change DrinkCategoryActivity to use the Starbuzz database

When we updated `DrinkActivity` to get it to read data from the Starbuzz database, we created a cursor to read data for the drink the user selected, and then we used the values from the cursor to update `DrinkActivity`'s views.

The steps we need to go through to update `DrinkCategoryActivity` are slightly different. This is because `DrinkCategoryActivity` displays a list view that uses the drink data as its source. We need to switch the source of this data to be the Starbuzz database.

Here are the steps we need to go through to change `DrinkCategoryActivity` so that it uses the Starbuzz database:

1. **Create a cursor to read drink data from the database.**
As before, we need to get a reference to the Starbuzz database. Then we'll create a cursor to retrieve the drink names from the DRINK table.

2. **Replace the list view's array adapter with a cursor adapter.**
The list view currently uses an array adapter to get its drink names. This is because the data's held in an array in the `Drink` class. Because we're now accessing the data using a cursor, we'll use a cursor adapter instead.

Before we get started on these tasks, let's remind ourselves of the *DrinkCategoryActivity.java* code we created in Chapter 7.

The current DrinkCategoryActivity code

Here's a reminder of what the current *DrinkCategoryActivity.java* code looks like. The `onCreate()` method populates a list view with drinks using an array adapter. The `onListItemClick()` method adds the drink the user selects to an intent, and then starts `DrinkActivity`:

```
package com.hfad.starbuzz;

...

public class DrinkCategoryActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink_category);
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<>(
                this,
                android.R.layout.simple_list_item_1,
                Drink.drinks);
        ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
        listDrinks.setAdapter(listAdapter);

        //Create a listener to listen for clicks in the list view
        AdapterView.OnItemClickListener itemClickListener =
                new AdapterView.OnItemClickListener(){
                    public void onItemClick(AdapterView<?> listDrinks,
                                            View itemView,
                                            int position,
                                            long id) {
                        //Pass the drink the user clicks on to DrinkActivity
                        Intent intent = new Intent(DrinkCategoryActivity.this,
                                DrinkActivity.class);
                        intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int) id);
                        startActivity(intent);
                    }
                };

        //Assign the listener to the list view
        listDrinks.setOnItemClickListener(itemClickListener);
    }
}
```

At the moment, we're using an ArrayAdapter to bind an array to the ListView. We need to replace this code so that the data comes from a database instead.

Create cursor
Cursor adapter

Get a reference to the Starbuzz database...

We need to change `DrinkCategoryActivity` so that it gets its data from the Starbuzz database. Just as before, this means that we need to create a cursor to return the data we need.

We start by getting a reference to the database. We only need to read the drink data and not update it, so we'll use the `getReadableDatabase()` method as we did before:

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
```

We get a reference to the database in exactly the same way that we did earlier in this chapter.

...then create a cursor that returns the drinks

To create the cursor, we need to specify what data we want it to contain. We want to use the cursor to display a list of drink names, so the cursor needs to include the NAME column. We'll also include the _id column to get the ID of the drink: we need to pass the ID of the drink the user chooses to `DrinkActivity` so that `DrinkActivity` can display its details. Here's the cursor:

```
cursor = db.query("DRINK",
                  new String[]{"_id", "NAME"},
                  null, null, null, null, null);
```

This cursor returns the _id and NAME of every record in the DRINK table.

Putting this together, here's the code to get a reference to the database and create the cursor (you'll add this code to *DrinkCategoryActivity.java* later when we show you the full code listing):

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
try {
    SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
    cursor = db.query("DRINK",
                      new String[]{"_id", "NAME"},
                      null, null, null, null, null);
    //Code to use data from the database
} catch(SQLiteException e) {
    Toast toast = Toast.makeText(this,
                                 "Database unavailable",
                                 Toast.LENGTH_SHORT);
    toast.show();
}
```

If the database is unavailable, a SQLiteException gets thrown. If this happens, we'll use a toast to display a pop-up message as before.

Next we'll use the cursor's data to populate `DrinkCategoryActivity`'s list view.

Create cursor
Cursor adapter

How do we replace the array data in the list view?

When we wanted `DrinkCategoryActivity`'s list view to display data from the `Drink.drinks` array, we used an array adapter. As you saw back in Chapter 7, an array adapter is a type of adapter that works with arrays. It acts as a bridge between the data in an array and a list view:

Now that we're getting our data from a cursor, we'll use a **cursor adapter** to bind the data to our list view. A cursor adapter is just like an array adapter, except that instead of getting its data from an array, it reads the data from a cursor:

We'll look at this in more detail on the next page.

ListViews and Spinners can use any subclass of the Adapter class for their data. This includes ArrayAdapter, CursorAdapter, and SimpleCursorAdapter (a subclass of CursorAdapter).

Create cursor
Cursor adapter

A simple cursor adapter maps cursor data to views

We're going to create a simple cursor adapter, a type of cursor adapter that can be used in most cases where you need to display cursor data in a list view. It takes columns from a cursor, and maps them to text views or image views, for example in a list view.

In our case, we want to display a list of drink names. We'll use a simple cursor adapter to map the name of each drink returned by our cursor to `DrinkCategoryActivity`'s list view.

Here's how it will work:

1. **The list view asks the adapter for data.**

2. **The adapter asks the cursor for data from the database.**

3. **The adapter returns the data to the list view.**
The name of each drink is displayed in the list view as a separate text view.

Let's construct the simple cursor adapter.

Create cursor
Cursor adapter

How to use a simple cursor adapter

You use a simple cursor adapter in a similar way to how you use an array adapter: you initialize the adapter, then attach it to the list view.

We're going to create a simple cursor adapter to display a list of drink names from the DRINK table. To do this, we'll create a new instance of the `SimpleCursorAdapter` class, passing in parameters to tell the adapter what data to use and how it should be displayed. Finally, we'll assign the adapter to the list view.

Here's the code (you'll add it to *DrinkCategoryActivity.java* later in the chapter):

```
SimpleCursorAdapter listAdapter = new SimpleCursorAdapter(this,
                    android.R.layout.simple_list_item_1,
                    cursor,
                    new String[]{"NAME"},
                    new int[]{android.R.id.text1},
                    0);
listDrinks.setAdapter(listAdapter);
```

"this" is the current activity.

This is the same layout we used with the array adapter. It displays a single value for each row in the list view.

This is the cursor.

Display the contents of the NAME column in the ListView text views.

Use setAdapter() to connect the adapter to the list view.

The general form of the `SimpleCursorAdapter` constructor looks like this:

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(Context context,
                                                      int layout,
                                                      Cursor cursor,
                                                      String[] fromColumns,
                                                      int[] toViews,
                                                      int flags)
```

This is usually the current activity.

How to display the data

The cursor you create. The cursor should include the _id column, and the data you want to appear.

Which columns in the cursor to match to which views

Used to determine the behavior of the cursor

The `context` and `layout` parameters are exactly the same ones you used when you created an array adapter: `context` is the current context, and `layout` says how you want to display the data. Instead of saying which array we need to get our data from, we need to specify which cursor contains the data. You then use `fromColumns` to specify which columns in the cursor you want to use, and `toViews` to say which views you want to display them in.

The `flags` parameter is usually set to `0`, which is the default. The alternative is to set it to `FLAG_REGISTER_CONTENT_OBSERVER` to register a content observer that will be notified when the content changes. We're not covering this alternative here, as it can lead to memory leaks (you'll see how to deal with changing content in the next chapter).

Any cursor you use with a cursor adapter MUST include the _id column or it won't work.

Create cursor
Cursor adapter

Close the cursor and database

When we introduced you to cursors earlier in the chapter, we said that you needed to close the cursor and database after you'd finished with it in order to release their resources. In our `DrinkActivity` code, we used a cursor to retrieve drink details from the database, and once we'd used these values with our views, we immediately closed the cursor and database.

When you use a cursor adapter, (including a simple cursor adapter) it works slightly differently; the cursor adapter needs the cursor to stay open in case it needs to retrieve more data from it. Let's look in more detail at how cursor adapters work to see why this might happen.

1. **The list view gets displayed on the screen.**
When the list is first displayed, it will be sized to fit the screen. Let's say it has space to show five items.

2. **The list view asks its adapter for the first five items.**

3. **The cursor adapter asks its cursor to read five rows from the database.**
No matter how many rows the database table contains, the cursor only needs to read the first five rows.

The story continues

4 The user scrolls the list.
As the user scrolls the list, the adapter asks the cursor to read more rows from the database. This works fine if the cursor's still open. But if the cursor's already been closed, the cursor adapter can't get any more data from the database.

This means that you can't immediately close the cursor and database once you've used the `setAdapter()` method to connect the cursor adapter to your list view. Instead, we'll close the cursor and database in the activity's `onDestroy()` method, which gets called just before the activity is destroyed. Because the activity's being destroyed, there's no further need for the cursor or database connection to stay open, so they can be closed:

```
public void onDestroy(){
    super.onDestroy();
    cursor.close();
    db.close();
}
```

Close the cursor and database when the activity is destroyed.

That's everything you need to know in order to update the code for `DrinkCategoryActivity`, so have a go at the exercise on the next page.

Pool Puzzle

Your **job** is to take code segments from the pool and place them into the blank lines in *DrinkCategoryActivity.java*. You may **not** use the same code segment more than once, and you won't need to use all the code segments. Your **goal** is to populate the list view with a list of drinks from the database.

```
public class DrinkCategoryActivity extends Activity {

    private SQLiteDatabase db;
    private Cursor cursor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink_category);
        ListView listDrinks = (ListView) findViewById(R.id.list_drinks);

        ..................................... starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {
            db = starbuzzDatabaseHelper. ..................................... ;
            cursor = db.query("DRINK",
                    new String[]{ ..................................................... },
                    null, null, null, null, null);
```

The code continues → on the next page

Note: each segment in the pool can only be used once!

```
        SimpleCursorAdapter listAdapter = new ......................................(this,
                android.R.layout.simple_list_item_1,
                .............,
                new String[]{..............},
                new int[]{android.R.id.text1},
                0);
        listDrinks.setAdapter(listAdapter);
    } catch( ...............................  e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }

    //Create the listener
    AdapterView.OnItemClickListener itemClickListener =
            new AdapterView.OnItemClickListener(){
                public void onItemClick(AdapterView<?> listDrinks,
                                        View itemView,
                                        int position,
                                        long id) {
                    //Pass the drink the user clicks on to DrinkActivity
                    Intent intent = new Intent(DrinkCategoryActivity.this,
                            DrinkActivity.class);
                    intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int) id);
                    startActivity(intent);
                }
            };

    //Assign the listener to the list view
    listDrinks.setOnItemClickListener(itemClickListener);
  }

  @Override
  public void onDestroy(){
    super.onDestroy();
    ..............close();
    ..............close();
  }
}
```

Pool Puzzle Solution

Your **job** is to take code segments from the pool and place them into the blank lines in *DrinkCategoryActivity.java*. You may **not** use the same code segment more than once, and you won't need to use all the code segments. Your **goal** is to populate the list view with a list of drinks from the database.

```
public class DrinkCategoryActivity extends Activity {

    private SQLiteDatabase db;
    private Cursor cursor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink_category);
        ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
```

You get a reference to the database using a SQLiteOpenHelper.

```
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {
            db = starbuzzDatabaseHelper. getReadableDatabase() ;
            cursor = db.query("DRINK",
                    new String[]{ "_id", "NAME" },
                    null, null, null, null, null);
```

We're reading from the database, so we just need read-only access.

The cursor must include the _id and NAME columns. We want to pass the drink ID to DrinkActivity, and we need to display the drink names.

These code snippets were not needed.

We're using a SimpleCursorAdapter.

```
        SimpleCursorAdapter listAdapter = new SimpleCursorAdapter (this,
                android.R.layout.simple_list_item_1,
                cursor ,
                new String[]{"NAME" },
                new int[]{android.R.id.text1},
                0);
        listDrinks.setAdapter(listAdapter);
    } catch( SQLiteException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }

    //Create the listener
    AdapterView.OnItemClickListener itemClickListener =
            new AdapterView.OnItemClickListener(){
                public void onItemClick(AdapterView<?> listDrinks,
                                        View itemView,
                                        int position,
                                        long id) {
                    //Pass the drink the user clicks on to DrinkActivity
                    Intent intent = new Intent(DrinkCategoryActivity.this,
                            DrinkActivity.class);
                    intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int) id);
                    startActivity(intent);
                }
            };

    //Assign the listener to the list view
    listDrinks.setOnItemClickListener(itemClickListener);
  }

  @Override
  public void onDestroy(){
    super.onDestroy();
    cursor .close();
    db .close();
  }
}
```

Use the cursor we just created.

Display the contents of the NAME column.

If the database is unavailable, we'll catch the SQLiteException.

Close the cursor before you close the database.

Create cursor
Cursor adapter

The revised code for DrinkCategoryActivity

Here's the full code for *DrinkCategoryActivity.java*, with the array adapter replaced by a cursor adapter (the changes are in bold); update your code to match ours:

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;
import android.view.View;
import android.content.Intent;
import android.widget.AdapterView;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.widget.SimpleCursorAdapter;
import android.widget.Toast;

public class DrinkCategoryActivity extends Activity {

    private SQLiteDatabase db;
    private Cursor cursor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink_category);
        ~~ArrayAdapter<Drink> listAdapter = new ArrayAdapter<>(~~
            ~~this,~~
            ~~android.R.layout.simple_list_item_1,~~
            ~~Drink.drinks);~~
        ListView listDrinks = (ListView) findViewById(R.id.list_drinks);
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {
            db = starbuzzDatabaseHelper.getReadableDatabase();
            cursor = db.query("DRINK",
                    new String[]{"_id", "NAME"},
                    null, null, null, null, null);
```

We're using these extra classes, so you need to import them.

We're adding these as private variables so we can close the database and cursor in our onDestroy() method.

We're no longer using an array adapter, so delete these lines of code.

Get a reference to the database.

Create the cursor.

The code continues on the next page.

Create cursor
Cursor adapter

The DrinkCategoryActivity code (continued)

```
        SimpleCursorAdapter listAdapter = new SimpleCursorAdapter(this,
                android.R.layout.simple_list_item_1,
                cursor,
                new String[]{"NAME"},
                new int[]{android.R.id.text1},
                0);
        listDrinks.setAdapter(listAdapter);
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }

    //Create a listener to listen for clicks in the list view
    AdapterView.OnItemClickListener itemClickListener =
            new AdapterView.OnItemClickListener(){
                public void onItemClick(AdapterView<?> listDrinks,
                                        View itemView,
                                        int position,
                                        long id) {
                    //Pass the drink the user clicks on to DrinkActivity
                    Intent intent = new Intent(DrinkCategoryActivity.this,
                            DrinkActivity.class);
                    intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int) id);
                    startActivity(intent);
                }
            };

    //Assign the listener to the list view
    listDrinks.setOnItemClickListener(itemClickListener);
  }

  @Override
  public void onDestroy(){
      super.onDestroy();
      cursor.close();
      db.close();
  }
}
```

Map the contents of the NAME column to the text in the ListView.

Create the cursor adapter.

Set the adapter to the ListView.

Display a message to the user if a SQLiteException gets thrown.

We didn't need to change any of the listener code.

We're closing the database and cursor in the activity's onDestroy() method. The cursor will stay open until the cursor adapter no longer needs it.

Let's try running our freshly updated app.

Test drive the app

When you run the app, `TopLevelActivity` gets displayed.

When you click on the Drinks item, `DrinkCategoryActivity` is launched. It displays all the drinks from the Starbuzz database.

When you click on one of the drinks, `DrinkActivity` is launched and details of the selected drink are displayed.

The app looks exactly the same as before, but now the data is being read from the database. In fact, you can now delete the *Drink.java* code, because we no longer need the array of drinks. Every piece of data we need is now coming from the database.

Your Android Toolbox

You've got Chapter 16 under your belt and now you've added basic cursors to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- A **cursor** lets you read from and write to the database.
- You create a cursor by calling the `SQLiteDatabase query()` method. Behind the scenes, this builds a SQL `SELECT` statement.
- The `getWritableDatabase()` method returns a `SQLiteDatabase` object that allows you to read from and write to the database.
- The `getReadableDatabase()` returns a `SQLiteDatabase` object. This gives you read-only access to the database. It *may* also allow you to write to the database, but this isn't guaranteed.
- Navigate through a cursor using the `moveTo*()` methods.
- Get values from a cursor using the `get*()` methods. Close cursors and database connections after you've finished with them.
- A **cursor adapter** is an adapter that works with cursors. Use `SimpleCursorAdapter` to populate a list view with the values returned by a cursor.

17 cursors and asynctasks

Staying in the Background

In most apps, you'll need your app to update its data.

So far you've seen how to create apps that read data from a SQLite database. But what if you want to update the app's data? In this chapter you'll see how to get your app to **respond to user input** and **update values in the database**. You'll also find out how to **refresh the data that's displayed** once it's been updated. Finally, you'll see how writing efficient **multithreaded code** with **AsyncTask**s will keep your app speedy.

We want our Starbuzz app to update database data

In Chapter 16, you learned how to change your app to read its data from a SQLite database. You saw how to read an individual record (a drink from the Starbuzz data) and display that record's data in an activity. You also learned how to populate a list view with database data (in this case drink names) using a cursor adapter.

In both of these scenarios, you only needed to *read* data from the database. But what if you want users to be able to update the data?

We're going to change the Starbuzz app so that users can record which drinks are their favorites. We'll do this by adding a checkbox to `DrinkActivity`; if it's checked, it means the current drink is one of the user's favorites:

Users can say which drinks are their favorites by checking a checkbox. We need to add this checkbox to DrinkActivity and get it to update the database.

We'll also add a new list view to `TopLevelActivity`, which contains the user's favorite drinks:

In the real world, you'd probably want to use tab navigation for these items. We're deliberately keeping the app pretty basic because we want you to focus on databases.

We'll add a ListView to TopLevelActivity, which contains the user's favorite drinks.

When the user clicks on a drink, it takes them to that drink's details in DrinkActivity.

We'll update DrinkActivity first

In Chapter 15, we added a FAVORITE column to the DRINK table in the Starbuzz database. We'll use this column to let users indicate whether a particular drink is one of their favorites. We'll display its value in the new checkbox we're going to add to `DrinkActivity`, and when the user clicks on the checkbox, we'll update the FAVORITE column with the new value.

Here are the steps we'll go through to update `DrinkActivity`:

1. **Update DrinkActivity's layout to add a checkbox and text label.**

We'll add this checkbox and label to activity_drink.xml.

2. **Display the value of the FAVORITE column in the checkbox.**
To do this, we'll need to retrieve the value of the FAVORITE column from the Starbuzz database.

3. **Update the FAVORITE column when the checkbox is clicked.**
We'll update the FAVORITE column with the value of the checkbox so that the data in the database stays up to date.

Let's get started.

Do this!

We're going to update the Starbuzz app in this chapter, so open your Starbuzz project in Android Studio.

Update layout
Show favorite
Update favorite

Add a checkbox to DrinkActivity's layout

We'll start by adding a new checkbox to `DrinkActivity`'s layout to indicate whether the current drink is one of the user's favorites. We're using a checkbox, as it's an easy way to display true/false values.

First, add a String resource called `"favorite"` to *strings.xml* (we'll use this as a label for the checkbox):

```
<string name="favorite">Favorite</string>
```

Then add the checkbox to *activity_drink.xml*. We'll give the checkbox an ID of `favorite` so that we can refer to it in our activity code. We'll also set its `android:onClick` attribute to `"onFavoriteClicked"` so that it calls the `onFavoriteClicked()` method in `DrinkActivity` when the user clicks on the checkbox. Here's the layout code; update your code to reflect our changes (they're in bold):

```
<LinearLayout ... >
    <ImageView
        android:id="@+id/photo"
        android:layout_width="190dp"
        android:layout_height="190dp" />

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/description"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <CheckBox android:id="@+id/favorite"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/favorite"
        android:onClick="onFavoriteClicked" />
</LinearLayout>
```

These are the photo, name, and description views we added when we first created the activity.

The checkbox has an ID of favorite.

We're giving the checkbox a label.

When the checkbox is clicked, the onFavoriteClicked() method in DrinkActivity will get called. We need to write this method.

Next we'll change the `DrinkActivity` code to get the checkbox to reflect the value of the FAVORITE column from the database.

Display the value of the FAVORITE column

In order to update the checkbox, we first need to retrieve the value of the FAVORITE column from the database. We can do this by updating the cursor we're using in `DrinkActivity`'s `onCreate()` method to read drink values from the database.

Here's the cursor we're currently using to return data for the drink the user has selected:

```
Cursor cursor = db.query("DRINK",
        new String[]{"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
        "_id = ?",
        new String[]{Integer.toString(drinkId)},
        null, null, null);
```

To include the FAVORITE column in the data that's returned, we simply add it to the array of column names returned by the cursor:

Add the FAVORITE column to the cursor.

```
Cursor cursor = db.query("DRINK",
        new String[]{"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID", "FAVORITE"},
        "_id = ?",
        new String[]{Integer.toString(drinkId)},
        null, null, null);
```

You don't need to update your version of the code yet. We'll show you the full set of changes for DrinkActivity.java soon.

Once we have the value of the FAVORITE column, we can update the `favorite` checkbox accordingly. To get this value, we first navigate to the first (and only) record in the cursor using:

```
cursor.moveToFirst();
```

We can then get the value of the column for the current drink. The FAVORITE column contains numeric values, where `0` is false and `1` is true. We want the `favorite` checkbox to be ticked if the value is `1` (true), and unticked if the value is `0` (false), so we'll use the following code to update the checkbox:

Get the value of the FAVORITE column. It's stored in the database as 1 for true, 0 for false.

```
boolean isFavorite = (cursor.getInt(3) == 1);
CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
favorite.setChecked(isFavorite);
```

Set the value of the favorite checkbox.

That's all the code we need to reflect the value of the FAVORITE column in the `favorite` checkbox. Next, we need to get the checkbox to respond to clicks so that it updates the database when its value changes.

Respond to clicks to update the database

Update layout
Show favorite
Update favorite

When we added the `favorite` checkbox to *activity_drink.xml*, we set its `android:onClick` attribute to `onFavoriteClicked()`. This means that whenever the user clicks on the checkbox, the `onFavoriteClicked()` method in the activity will get called. We'll use this method to update the database with the current value of the checkbox. If the user checks or unchecks the checkbox, the `onFavoriteClicked()` method will save the user's change to the database.

In Chapter 15, you saw how to use SQLiteDatabase methods to change the data held in a SQLite database: the `insert()` method to insert data, the `delete()` method to delete data, and the `update()` method to update existing records.

You can use these methods to change data from within your activity. As an example, you could use the `insert()` method to add new drink records to the DRINK table, or the `delete()` method to delete them. In our case, we want to update the DRINK table's FAVORITE column with the value of the checkbox using the `update()` method.

As a reminder, the `update()` method looks like this:

```
db.update(String table,
          ContentValues values,
          String conditionClause,
          String[] conditionArguments);
```

The table whose data you want to update

The new values

The criteria for updating the data

where `table` is the name of the table you want to update, and `values` is a `ContentValues` object containing name/value pairs of the columns you want to update and the values you want to set them to. The `conditionClause` and `conditionArguments` parameters lets you specify which records you want to update.

You already know everything you need to get `DrinkActivity` to update the FAVORITE column for the current drink when the checkbox is clicked, so have a go at the following exercise.

Code Magnets

In our code for `DrinkActivity` we want to update the FAVORITE column in the database with the value of the `favorite` checkbox. Can you construct the `onFavoriteClicked()` method so that it will do that?

```
public class DrinkActivity extends Activity {
...
    //Update the database when the checkbox is clicked

    public void onFavoriteClicked(                    ){

        int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);

                              drinkValues = new                              ;

        drinkValues.put(                      , favorite.isChecked());

        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {

            SQLiteDatabase db = starbuzzDatabaseHelper.                                      ;

            db.update(                ,                        ,

                                                    , new String[] {Integer.toString(drinkId)});
            db.close();
        } catch(SQLiteException e) {
            Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

You won't need to use all the magnets.

`drinkValues`

`"FAVORITE"`

`View view`

`"_id = ?"`

`ContentValues`

`"DRINK"`

`ContentValues()`

`getReadableDatabase()`

`favorite`

`getWritableDatabase()`

Code Magnets Solution

In our code for `DrinkActivity` we want to update the FAVORITE column in the database with the value of the `favorite` checkbox. Can you construct the `onFavoriteClicked()` method so that it will do that?

```
public class DrinkActivity extends Activity {
...
    //Update the database when the checkbox is clicked

    public void onFavoriteClicked( View view ){

        int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);

        ContentValues drinkValues = new ContentValues();

        drinkValues.put( "FAVORITE" , favorite.isChecked());

        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {

            SQLiteDatabase db = starbuzzDatabaseHelper. getWritableDatabase();

            db.update( "DRINK" , drinkValues ,
                                "_id = ?" , new String[] {Integer.toString(drinkId)});
            db.close();
        } catch(SQLiteException e) {
            Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

We need read/write access to the database to update it.

You didn't need to use these magnets.

`getReadableDatabase()` `favorite`

- [x] Update layout
- [x] Show favorite
- [ ] Update favorite

The full DrinkActivity.java code

We've now done everything we need to change `DrinkActivity` so that it reflects the contents of the FAVORITE column in the `favorite` checkbox. It then updates the value of the column in the database if the user changes the value of the checkbox.

Here's the full code for *DrinkActivity.java*, so update your version of the code so that it reflects ours (our changes are in bold):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.view.View;
import android.widget.CheckBox;
import android.content.ContentValues;

public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKID = "drinkId";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

        //Get the drink from the intent
        int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);
```

We're using these extra classes, so you need to import them.

The code continues on the next page.

DrinkActivity.java (continued)

Update layout
Show favorite
Update favorite

```
//Create a cursor
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
try {
    SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
    Cursor cursor = db.query("DRINK",
            new String[]{"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID", "FAVORITE"},
            "_id = ?",
            new String[]{Integer.toString(drinkId)},
            null, null, null);

    //Move to the first record in the Cursor
    if (cursor.moveToFirst()) {
        //Get the drink details from the cursor
        String nameText = cursor.getString(0);
        String descriptionText = cursor.getString(1);
        int photoId = cursor.getInt(2);
        boolean isFavorite = (cursor.getInt(3) == 1);

        //Populate the drink name
        TextView name = (TextView) findViewById(R.id.name);
        name.setText(nameText);

        //Populate the drink description
        TextView description = (TextView) findViewById(R.id.description);
        description.setText(descriptionText);

        //Populate the drink image
        ImageView photo = (ImageView) findViewById(R.id.photo);
        photo.setImageResource(photoId);
        photo.setContentDescription(nameText);

        //Populate the favorite checkbox
        CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
        favorite.setChecked(isFavorite);
    }
```

Add the FAVORITE column to the cursor.

If the FAVORITE column has a value of 1, this indicates a true value.

If the drink is a favorite, put a checkmark in the favorite checkbox.

The code continues on the next page.

- ☑ Update layout
- ☑ Show favorite
- ➔ ☐ Update favorite

DrinkActivity.java (continued)

```
            cursor.close();
            db.close();
        } catch (SQLiteException e) {
            Toast toast = Toast.makeText(this,
                    "Database unavailable",
                    Toast.LENGTH_SHORT);
            toast.show();
        }
    }

    //Update the database when the checkbox is clicked
    public void onFavoriteClicked(View view){
        int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);

        //Get the value of the checkbox
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
        ContentValues drinkValues = new ContentValues();
        drinkValues.put("FAVORITE", favorite.isChecked());

        //Get a reference to the database and update the FAVORITE column
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            db.update("DRINK",
                      drinkValues,
                     "_id = ?",
                     new String[] {Integer.toString(drinkId)});
            db.close();
        } catch(SQLiteException e) {
            Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

Add the value of the favorite checkbox to the drinkValues ContentValues object.

Update the drink's FAVORITE column in the database to the value of the checkbox.

Display a message if there's a problem with the database.

Let's check what happens when we run the app.

Test drive the app

When we run the app and navigate to a drink, the new `favorite` checkbox is displayed (unchecked):

Here's the new checkbox we added with its label.

When we click on the checkbox, a checkmark appears to indicate that the drink is one of our favorites:

When we click on the checkbox, a checkmark appears, and the value gets written to the database.

When we close the app and navigate back to the drink, the checkmark remains. The value of the checkbox has been written to the database.

That's everything we need to display the value of the FAVORITE column from the database, and update the database with any changes to it.

Display favorites in TopLevelActivity

The next thing we need to do is display the user's favorite drinks in `TopLevelActivity`. Here are the steps we'll go through to do this:

1. **Add a list view and text view to TopLevelActivity's layout.**

2. **Populate the list view and get it to respond to clicks.**
   We'll create a new cursor that retrieves the user's favorite drinks from the database, and attach it to the list view using a cursor adapter. We'll then create an `onItemClickListener` so that we can get `TopLevelActivity` to start `DrinkActivity` when the user clicks on one of the drinks.

3. **Refresh the list view data when we choose a new favorite drink.**
   If we choose a new favorite drink in `DrinkActivity`, we want it to be displayed in `TopLevelActivity`'s list view when we navigate back to it.

Applying all of these changes will enable us to display the user's favorite drinks in `TopLevelActivity`.

We'll go through these steps over the next few pages.

Display the favorite drinks in activity_top_level.xml

As we said on the previous page, we're going to add a list view to *activity_top_level.xml*, which we'll use to display a list of the user's favorite drinks. We'll also add a text view to display a heading for the list.

First, add the following String resource to *strings.xml* (we'll use this for the text view's text):

```
<string name="favorites">Your favorite drinks:</string>
```

Next, we'll add the new text view and list view to the layout. Here's our code for *activity_top_level.xml*; update your version of the code to match our changes:

```
<LinearLayout ... >
    <ImageView
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:src="@drawable/starbuzz_logo"
        android:contentDescription="@string/starbuzz_logo" />

    <ListView
        android:id="@+id/list_options"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/options" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/favorites" />

    <ListView
        android:id="@+id/list_favorites"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The layout already contains the Starbuzz logo and list view.

We'll add a text view to display the text "Your favorite drinks." We'll put this in a String called favorites.

The list_favorites ListView will display the user's favorite drinks.

Those are all the changes we need to make to *activity_top_level.xml*. Next, we'll update *TopLevelActivity.java*.

- Update layout
- **Populate list view**
- Refresh data

Refactor TopLevelActivity.java

Before we write any code for our new list view, we're going to refactor our existing `TopLevelActivity` code. This will make the code a lot easier to read later on. We'll move the code relating to the options list view into a new method called `setupOptionsListView()`. We'll then call this method from the `onCreate()` method.

Here's our code for *TopLevelActivity.java* (update your version of the code to reflect our changes).

```
package com.hfad.starbuzz;
...
public class TopLevelActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_top_level);
        setupOptionsListView();
    }

    private void setupOptionsListView() {
        //Create an OnItemClickListener
        AdapterView.OnItemClickListener itemClickListener =
                                        new AdapterView.OnItemClickListener(){
            public void onItemClick(AdapterView<?> listView,
                                    View itemView,
                                    int position,
                                    long id) {
                if (position == 0) {
                    Intent intent = new Intent(TopLevelActivity.this,
                                               DrinkCategoryActivity.class);
                    startActivity(intent);
                }
            }
        };
        //Add the listener to the list view
        ListView listView = (ListView) findViewById(R.id.list_options);
        listView.setOnItemClickListener(itemClickListener);
    }
}
```

Call the new setupOptionsListView() method.

All of this code was in the onCreate() method. We're putting it in a new method to make the code tidier.

If the Drink option in the list_options list view is clicked, start DrinkCategoryActivity.

What changes are needed for TopLevelActivity.java

We need to display the user's favorite drinks in the `list_favorites` list view we added to the layout, and get it to respond to clicks. To do this, we need to do the following:

1. **Populate the list_favorites list view using a cursor.**
   The cursor will return all drinks where the FAVORITE column has been set to `1`—all drinks that the user has flagged as being a favorite. Just as we did in our code for `DrinkCategoryActivity`, we can connect the cursor to the list view using a cursor adapter.

2. **Create an onItemClickListener so that the list_favorites list view can respond to clicks.**
   If the user clicks on one of their favorite drinks, we can create an intent that starts `DrinkActivity`, passing it the ID of the drink that was clicked. This will show the user details of the drink they've just chosen.

You've already seen all the code that's needed to do this. In fact, it's almost identical to the code we wrote in earlier chapters to control the list of drinks in `DrinkCategoryActivity`. The only difference is that this time, we only want to display drinks with a value of 1 in the FAVORITE column.

We've decided to put the code that controls a list view in a new method called `setupFavoritesListView()`. We'll show you this method on the next page before adding it to *TopLevelActivity.java*.

Ready Bake Code

The **setupFavoritesListView()** method populates the **list_favorites** list view with the names of the user's favorite drinks. Make sure you understand the code below before turning the page.

- Update layout
- **Populate list view**
- Refresh data

```
private void setupFavoritesListView() {
    //Populate the list_favorites ListView from a cursor
    ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
    try{
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        db = starbuzzDatabaseHelper.getReadableDatabase();
        favoritesCursor = db.query("DRINK",
                new String[] { "_id", "NAME"},
                "FAVORITE = 1",
                null, null, null, null);
        CursorAdapter favoriteAdapter =
                new SimpleCursorAdapter(TopLevelActivity.this,
                        android.R.layout.simple_list_item_1,
                        favoritesCursor,
                        new String[]{"NAME"},
                        new int[]{android.R.id.text1}, 0);
        listFavorites.setAdapter(favoriteAdapter);
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }

    //Navigate to DrinkActivity if a drink is clicked
    listFavorites.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> listView, View v, int position, long id) {
            Intent intent = new Intent(TopLevelActivity.this, DrinkActivity.class);
            intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int)id);
            startActivity(intent);
        }
    });
}
```

Get the list_favorites list view.

Create a cursor that gets the values of the _id and NAME columns where FAVORITE=1.

Get the names of the user's favorite drinks.

Create a new cursor adapter.

Use the cursor in the cursor adapter.

Display the names of the drinks in the list view.

Display a message if there's a problem with the database.

This will get called if an item in the list view is clicked.

If the user clicks on one of the items in the list_favorites list view, create an intent to start DrinkActivity, including the ID of the drink as extra information.

The new TopLevelActivity.java code

Update layout
Populate list view
Refresh data

We've updated `TopLevelActivity` to populate the `list_favorites` list view and make it respond to clicks. Update your version of *TopLevelActivity.java* to match ours (there's a lot of new code, so go through it carefully and take your time):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.widget.AdapterView;
import android.widget.ListView;
import android.view.View;
import android.database.Cursor;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteDatabase;
import android.widget.SimpleCursorAdapter;
import android.widget.CursorAdapter;
import android.widget.Toast;

public class TopLevelActivity extends Activity {

    private SQLiteDatabase db;
    private Cursor favoritesCursor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_top_level);
        setupOptionsListView();
        setupFavoritesListView();
    }
```

We're using all these extra classes, so we need to import them.

We're adding the database and cursor as private variables so that we can access them in the setUpFavoritesListView() and onDestroy() methods.

Call the setupFavoritesListView() method from the onCreate() method.

The code continues on the next page.

Update layout
Populate list view
Refresh data

The TopLevelActivity.java code (continued)

We don't need to change this method.

```
private void setupOptionsListView() {
    //Create an OnItemClickListener
    AdapterView.OnItemClickListener itemClickListener =
                                        new AdapterView.OnItemClickListener(){
        public void onItemClick(AdapterView<?> listView,
                                View itemView,
                                int position,
                                long id) {
            if (position == 0) {
                Intent intent = new Intent(TopLevelActivity.this,
                                           DrinkCategoryActivity.class);
                startActivity(intent);
            }
        }
    };

    //Add the listener to the list view
    ListView listView = (ListView) findViewById(R.id.list_options);
    listView.setOnItemClickListener(itemClickListener);
}
```

This is the method we created to populate the list_favorites list view and make it respond to clicks.

```
private void setupFavoritesListView() {
    //Populate the list_favorites ListView from a cursor
    ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
    try{
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        db = starbuzzDatabaseHelper.getReadableDatabase();
        favoritesCursor = db.query("DRINK",
                new String[] { "_id", "NAME"},
                "FAVORITE = 1",
                null, null, null, null);
```

Get a reference to the database.

The list_favorites list view will use this cursor for its data.

The code continues on the next page.

The TopLevelActivity.java code (continued)

Update layout
Populate list view
Refresh data

```
        CursorAdapter favoriteAdapter =
                new SimpleCursorAdapter(TopLevelActivity.this,
                        android.R.layout.simple_list_item_1,
                        favoritesCursor,
                        new String[]{"NAME"},
                        new int[]{android.R.id.text1}, 0);
        listFavorites.setAdapter(favoriteAdapter);
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }

    //Navigate to DrinkActivity if a drink is clicked
    listFavorites.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> listView, View v, int position, long id) {
            Intent intent = new Intent(TopLevelActivity.this, DrinkActivity.class);
            intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int)id);
            startActivity(intent);
        }
    });
}

//Close the cursor and database in the onDestroy() method
@Override
public void onDestroy(){
    super.onDestroy();
    favoritesCursor.close();
    db.close();
}
}
```

Use the cursor in a cursor adapter.

Set the curser adapter to the list view.

Get the list_favorites list view to respond to clicks.

Start DrinkActivity, passing it the ID of the drink that was clicked on.

The onDestroy() method gets called just before the activity is destroyed. We'll close the cursor and database in this method, as we no longer need them if the activity's being destroyed.

The above code populates the `list_favorites` list view with the user's favorite drinks. When the user clicks on one of these drinks, an intent starts `DrinkActivity` and passes it the ID of the drink. Let's take the app for a test drive and see what happens.

- Update layout
- **Populate list view**
- Refresh data

Test drive the app

When you run the app, the new text view and `list_favorites` list view are displayed in `TopLevelActivity`. If you've marked a drink as being a favorite, it appears in the list view.

If you click on that drink, `DrinkActivity` starts and details of the drink are displayed.

Here's the new list_favorites list view we created. It displays a latte, as we marked this as a favorite drink earlier in the chapter.

When we click on the drink, its details are displayed.

But there's a problem. If you select a new drink as being a favorite, when you go back to `TopLevelActivity` the `list_favorites` list view doesn't include the new drink. The new drink is only included in the list view if you rotate the device.

We've marked a cappuccino as a favorite, but it's not appearing in the list view.

When we rotate the device, the cappuccino appears in the list view. Why's that?

Why do you think the new drink we chose as a favorite doesn't appear in the list view until we rotate the device? Give this some thought before turning the page.

Cursors don't automatically refresh

- ☑ Update layout
- ☑ Populate list view
- ☐ **Refresh data**

If the user chooses a new favorite drink by navigating through the app to `DrinkActivity`, the new favorite drink isn't automatically displayed in the `list_favorites` list view in `TopLevelActivity`. This is because ***cursors retrieve data when the cursor gets created***.

In our case, the cursor is created in the activity `onCreate()` method, so it gets its data when the activity is created. When the user navigates through the other activities, `TopLevelActivity` is stopped. It's not destroyed and recreated, so neither is the cursor.

When you start a second activity, the second activity is stacked on top of the first. The first activity isn't destroyed. Instead, it's paused and then stopped, as it loses the focus and stops being visible to the user.

Cursors don't automatically keep track of whether the underlying data in the database has changed. So if the underlying data changes after the cursor's been created, the cursor doesn't get updated: it still contains the original records, and none of the changes. That means that if the user marks a new drink as being a favorite after the cursor is created, the cursor will be out of date.

If you update the data in the database...

...the cursor won't see the new data if the cursor's already been created.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID	FAVORITE
1	"Latte"	"Espresso and steamed milk"	54543543	**1**
2	"Cappucc			
3	"Filter"			

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID	FAVORITE
1	"Latte"	"Espresso and steamed milk"	54543543	**0**
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453	0
3	"Filter"	"Our best drip coffee"	44324234	0

So how do we get around this?

- Update layout
- Populate list view
- **Refresh data**

Change the cursor with changeCursor()

The solution is to change the underlying cursor used by the `list_favorites` list view to an updated version. To do this, you define a new version of the cursor, get a reference to the list view's cursor adapter, and then call the cursor adapter's `changeCursor()` method to change the cursor. Here are the details:

1. Define the cursor

You define the cursor in exactly the same way as you did before. In our case, we want the query to return the user's favorite drinks, so we use:

```
Cursor newCursor = db.query("DRINK",
                            new String[] { "_id", "NAME"},
                            "FAVORITE = 1",
                            null, null, null, null);
```

This is the same query that we had before.

2. Get a reference to the cursor adapter

You get a reference to the list view's cursor adapter by calling the list view's `getAdapter()` method. This method returns an object of type `Adapter`. As our list view is using a cursor adapter, we can cast the adapter to a `CursorAdapter`:

```
ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
CursorAdapter adapter = (CursorAdapter) listFavorites.getAdapter();
```

You get the ListView's adapter using the getAdapter() method.

3. Change the cursor using changeCursor()

You change the cursor used by the cursor adapter by calling its `changeCursor()` method. This method takes one parameter, the new cursor:

```
adapter.changeCursor(newCursor);
```

Change the cursor used by the cursor adapter to the new one.

The `changeCursor()` method replaces the cursor adapter's current cursor with the new one. It then closes the old cursor, so you don't need to do this yourself.

We're going to change the cursor used by the `list_favorites` list view in `TopLevelActivity`'s `onRestart()` method. This means that the data in the list view will get refreshed when the user returns to `TopLevelActivity`. Any new favorite drinks the user has chosen will be displayed, and any drinks that are no longer flagged as favorites will be removed from the list.

We'll show you the full code for *TopLevelActivity.java* over the next few pages.

The revised TopLevelActivity.java code

Update layout
Populate list view
Refresh data

Here's the full *TopLevelActivity.java* code; update your code to reflect our changes (in bold).

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.widget.AdapterView;
import android.widget.ListView;
import android.view.View;
import android.database.Cursor;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteDatabase;
import android.widget.SimpleCursorAdapter;
import android.widget.CursorAdapter;
import android.widget.Toast;

public class TopLevelActivity extends Activity {

    private SQLiteDatabase db;
    private Cursor favoritesCursor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_top_level);
        setupOptionsListView();
        setupFavoritesListView();
    }
```

You don't need to change any of the code on this page.

The code continues on the next page.

Update layout
Populate list view
Refresh data

The TopLevelActivity.java code (continued)

```
private void setupOptionsListView() {
    //Create an OnItemClickListener
    AdapterView.OnItemClickListener itemClickListener =
                                        new AdapterView.OnItemClickListener(){
        public void onItemClick(AdapterView<?> listView,
                                View itemView,
                                int position,
                                long id) {
            if (position == 0) {
                Intent intent = new Intent(TopLevelActivity.this,
                                           DrinkCategoryActivity.class);
                startActivity(intent);
            }
        }
    };

    //Add the listener to the list view
    ListView listView = (ListView) findViewById(R.id.list_options);
    listView.setOnItemClickListener(itemClickListener);
}

private void setupFavoritesListView() {
    //Populate the list_favorites ListView from a cursor
    ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
    try{
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        db = starbuzzDatabaseHelper.getReadableDatabase();
        favoritesCursor = db.query("DRINK",
                                   new String[] { "_id", "NAME"},
                                   "FAVORITE = 1",
                                   null, null, null, null);

        CursorAdapter favoriteAdapter =
                new SimpleCursorAdapter(TopLevelActivity.this,
                        android.R.layout.simple_list_item_1,
                        favoritesCursor,
                        new String[]{"NAME"},
                        new int[]{android.R.id.text1}, 0);
        listFavorites.setAdapter(favoriteAdapter);
```

You don't need to change any of the code on this page.

The code continues on the next page.

The TopLevelActivity.java code (continued)

Update layout
Populate list view
Refresh data

```
        } catch(SQLiteException e) {
            Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }

        //Navigate to DrinkActivity if a drink is clicked
        listFavorites.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> listView, View v, int position, long id) {
                Intent intent = new Intent(TopLevelActivity.this, DrinkActivity.class);
                intent.putExtra(DrinkActivity.EXTRA_DRINKID, (int)id);
                startActivity(intent);
            }
        });
    }

    @Override
    public void onRestart() {
        super.onRestart();
        Cursor newCursor = db.query("DRINK",
                                    new String[] { "_id", "NAME"},
                                    "FAVORITE = 1",
                                    null, null, null, null);
        ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
        CursorAdapter adapter = (CursorAdapter) listFavorites.getAdapter();
        adapter.changeCursor(newCursor);
        favoritesCursor = newCursor;
    }

    //Close the cursor and database in the onDestroy() method
    @Override
    public void onDestroy(){
        super.onDestroy();
        favoritesCursor.close();
        db.close();
    }
}
```

Add the onRestart() method. This will get called when the user navigates back to TopLevelActivity.

Create a new version of the cursor.

Switch the cursor being used by the list_favorites list view to the new cursor.

Change the value of favoritesCursor to the new cursor so we can close it in the activity's onDestroy() method.

Let's see what happens when we run the app now.

- Update layout
- Populate list view
- **Refresh data**

Test drive the app

When we run the app, our favorite drinks are displayed in `TopLevelActivity` as before. When we click on one of the drinks, its details are displayed in `DrinkActivity`. If we uncheck the `favorite` checkbox for that drink and return to `TopLevelActivity`, the data in the `list_favorites` list view is refreshed and the drink is no longer displayed.

I've been thinking... Using databases in my app clearly has a lot of advantages, but doesn't opening and reading from the database slow the app down?

Databases are powerful, but they can be slow.

That means that even though our app works, we need to keep an eye on performance...

Databases can make your app go in slooow-moooo....

Think about what your app has to do when it opens a database. It first needs to go searching for the database file. If the database file isn't there, it needs to create a blank database. Then it needs to run all of the SQL commands to create tables inside the database and any initial data it needs. Finally, it needs to fire off some queries to get the data out of there.

That all takes time. For a tiny database like the one used in the Starbuzz app, it's not a lot of time. But as a database gets bigger and bigger, that time will increase and increase. Before you know it, your app will lose its mojo and will be slower than YouTube on Thanksgiving.

There's not a lot you can do about the speed of creating and reading from a database, but you *can* prevent it from slowing down your interface.

Life is better when threads work together

The big problem with accessing a slow database is that can make your app feel unresponsive. To understand why, you need to think about how threads work in Android. Since Lollipop, there are three kinds of threads you need to think about:

- **The main event thread**
  This is the real workhorse in Android. It listens for intents, it receives touch messages from the screen, and it calls all of the methods inside your activities.
- **The render thread**
  You don't normally interact with this thread, but it reads a list of requests for screen updates and then calls the device's low-level graphics hardware to repaint the screen and make your app look pretty.
- **Any other threads that you create**

If you're not careful, your app will do almost all of its work on the main event thread because this thread runs your event methods. If you just drop your database code into the `onCreate()` method (as we did in the Starbuzz app), then the main event thread will be busy talking to the database, instead of rushing off to look for any events from the screen or other apps. If your database code takes a long time, users will feel like they're being ignored or wonder if the app has crashed.

So the trick is to **move your database code off the main event thread and run it in a custom thread in the background**. We'll go through how you do this using the `DrinkActivity` code we wrote earlier in the chapter. As a reminder, the code updates the FAVORITE column in the Starbuzz database when the user clicks on the `favorite` checkbox, and displays a message if the database is unavailable.

Sharpen your pencil

We're going to run the `DrinkActivity` code to update the database in a background thread, but before we rush off and start hacking code, let's think about what we need to do.

The code that we have at the moment does three different things. Choose the type of thread you think each should run on. We've completed the first one to start you off.

A Set up the interface.

```
int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);
CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
ContentValues drinkValues = new ContentValues();
drinkValues.put("FAVORITE", favorite.isChecked());
```

This code must run on the main event thread, as it needs to access the activity's views.

Main event thread	A background thread
✓	

B Talk to the database.

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getWriteableDatabase();
db.update("DRINK",...);
```

Main event thread	A background thread

C Update what's displayed on the screen.

```
Toast toast = Toast.makeText(...);
toast.show();
```

Main event thread	A background thread

Sharpen your pencil Solution

We're going to run the `DrinkActivity` code to update the database in a background thread, but before we rush off and start hacking code, let's think about what we need to do.

The code that we have at the moment does three different things. Choose the type of thread you think each should run on. We've completed the first one to start you off.

A Set up the interface.

```
int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);
CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
ContentValues drinkValues = new ContentValues();
drinkValues.put("FAVORITE", favorite.isChecked());
```

Main event thread	A background thread
✓	

B Talk to the database.

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getWriteableDatabase();
db.update("DRINK",...);
```

We want to run the database code in the background.

Main event thread	A background thread
	✓

C Update what's displayed on the screen.

```
Toast toast = Toast.makeText(...);
toast.show();
```

We must run the code to display a message on the screen on the main event thread; otherwise, we'll get an exception.

Main event thread	A background thread
✓	

What code goes on which thread?

When you use databases in your app, it's a good idea to run database code in a background thread, and update views with the database data in the main event thread. We're going to work through the `onFavoritesClicked()` method in the `DrinkActivity` code so that you can see how to approach this sort of problem.

Here's the code for the method (we've split it into sections, which we'll describe below):

```
//Update the database when the checkbox is clicked
public void onFavoriteClicked(View view){
```

1
```
    int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);
    CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("FAVORITE", favorite.isChecked());
```

2
```
    SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
    try {
        SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        db.update("DRINK", drinkValues,
                                "_id = ?", new String[] {Integer.toString(drinkId)});
        db.close();
    } catch(SQLiteException e) {
```

3
```
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }
}
```

1 **Code that needs to be run before the database code**
The first few lines of code get the value of the `favorite` checkbox, and put it in the `drinkValues` `ContentValues` object. This code must be run before the database code.

2 **Database code that needs to be run on a background thread**
This updates the DRINK table.

3 **Code that needs to be run after the database code**
If the database is unavailable, we want to display a message to the user. This must run on the main event thread.

We're going to implement the code using an **`AsyncTask`**. What's that, you ask?

AsyncTask performs asynchronous tasks

An AsyncTask lets you perform operations in the background. When they've finished running, it then allows you to update views in the main event thread. If the task is repetitive, you can even use it to publish the progress of the task while it's running.

You create an AsyncTask by extending the AsyncTask class, and implementing its doInBackground() method. The code in this method runs in a background thread, so it's the perfect place for you to put database code. The AsyncTask class also has an onPreExecute() method that runs before doInBackground(), and an onPostExecute() method that runs afterward. There's an onProgressUpdate() method if you need to publish task progress.

Here's what an AsyncTask looks like:

You add your AsyncTask class as an inner class to the activity that needs to use it.

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>

    protected void onPreExecute() {
        //Code to run before executing the task
    }

    protected Result doInBackground(Params... params) {
        //Code that you want to run in a background thread
    }

    protected void onProgressUpdate(Progress... values) {
        //Code that you want to run to publish the progress of your task
    }

    protected void onPostExecute(Result result) {
        //Code that you want to run when the task is complete
    }
}
```

This method is optional. It runs before the code you want to run in the background.

You must implement this method. It contains the code you want to run in the background.

This method is optional. It lets you publish progress of the code running in the background.

This method is also optional. It runs after the code has finished running in the background.

AsyncTask is defined by three generic parameters: Params, Progress, and Results. Params is the type of object used to pass any task parameters to the doInBackground() method, Progress is the type of object used to indicate task progress, and Result is the type of the task result. You can set any of these to Void if you're not going to use them.

We'll go through this over the next few pages by creating a new AsyncTask called UpdateDrinkTask we can use to update drinks in the background. Later on, we'll add this to our DrinkActivity code as an inner class.

The onPreExecute() method

onPreExecute()

We'll start with the `onPreExecute()` method. This gets called before the background task begins, and it's used to set up the task. It's called on the main event thread, so it has access to views in the user interface. The `onPreExecute()` method takes no parameters, and has a `void` return type.

In our case, we're going to use the `onPreExecute()` method to get the value of the `favorite` checkbox, and put it in the `drinkValues` `ContentValues` object. This is because we need access to the checkbox view in order to do this, and it must be done before any of our database code can be run. We're using a separate attribute outside the method for the `drinkValues` `ContentValues` object so that other methods in the class can access the `ContentValues` object (we'll look at these methods over the next few pages).

Here's the code:

```
private class UpdateDrinkTask extends AsyncTask<Params, Progress, Result> {

    private ContentValues drinkValues;

    protected void onPreExecute() {
        CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
        drinkValues = new ContentValues();
        drinkValues.put("FAVORITE", favorite.isChecked());
    }

    ...

}
```

Before we run the database code, we need to get the value of the favorite checkbox.

Next, we'll look at the `doInBackground()` method.

The doInBackground() method

The `doInBackground()` method runs in the background immediately after `onPreExecute()`. You define what type of parameters the task should receive, and what the return type should be.

We're going to use the `doInBackground()` method for our database code so that it runs in a background thread. We'll pass it the ID of the drink we need to update; because the drink ID is an `int` value, we need to specify that the `doInBackground()` method receives `Integer` objects. We'll use a `Boolean` return value so we can tell whether the code ran successfully:

```
private class UpdateDrinkTask extends AsyncTask<Integer, Progress, Boolean> {

    private ContentValues drinkValues;

    ...

    protected Boolean doInBackground(Integer... drinks) {
        int drinkId = drinks[0];
        SQLiteOpenHelper starbuzzDatabaseHelper =
                                 new StarbuzzDatabaseHelper(DrinkActivity.this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            db.update("DRINK", drinkValues,
                                 "_id = ?", new String[] {Integer.toString(drinkId)});
            db.close();
            return true;
        } catch(SQLiteException e) {
            return false;
        }
    }

    ...
}
```

You change this to Integer to match the parameter of the doInBackground() method.

You change this to Boolean to match the return type of the doInBackground() method.

This code runs in a background thread.

This is an array of Integers, but we'll just include one item, the drink ID.

The update() method uses the drinkValues object that the onPreExecute() method created.

Next, we'll look at the `onProgressUpdate()` method.

The onProgressUpdate() method

The `onProgressUpdate()` method is called on the main event thread, so it has access to views in the user interface. You can use this method to display progress to the user by updating views on the screen. You define what type of parameters the method should have.

The `onProgressUpdate()` method runs if a call to `publishProgress()` is made by the `doInBackground()` method like this:

```
protected Boolean doInBackground(Integer... count) {
    for (int i = 0; i < count; i++) {
        publishProgress(i);
    }
}

protected void onProgressUpdate(Integer... progress) {
    setProgress(progress[0]);
}
```

This calls the onProgressUpdate() method, passing in a value of i.

In our app, we're not publishing the progress of our task, so we don't need to implement this method. We'll indicate that we're not using any objects for task progress by changing the signature of `UpdateDrinkTask`:

We're not using the onProgressUpdate() method, so this is Void.

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {

    ...

}
```

Finally, we'll look at the `onPostExecute()` method.

The onPostExecute() method

The `onPostExecute()` method is called after the background task has finished. It's called on the main event thread, so it has access to views in the user interface. You can use this method to present the results of the task to the user. The `onPostExecute()` method gets passed the results of the `doInBackground()` method, so it must take parameters that match the `doInBackground()` return type.

We're going to use the `onPostExecute()` method to check whether the database code in the `doInBackground()` method ran successfully. If it didn't, we'll display a message to the user. We're doing this in the `onPostExecute()` method, as this method can update the user interface; the `doInBackground()` method runs in a background thread, so it can't update views.

Here's the code:

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {

    ...

    protected void onPostExecute(Boolean success) {
        if (!success) {
            Toast toast = Toast.makeText(DrinkActivity.this,
                                         "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

This is set to Boolean, as our doInBackground() method returns a Boolean.

Pass the toast the DrinkActivity context.

Now that we've written the code for our `AsyncTask` methods, let's revisit the `AsyncTask` class parameters.

The AsyncTask class parameters

When we first introduced the `AsyncTask` class, we said it was defined by three generic parameters: `Params`, `Progress`, and `Results`. You specify what these are by looking at the type of parameters used by your `doInBackground()`, `onProgressUpdate()`, and `onPostExecute()` methods. `Params` is the type of the `doInBackground()` parameters, `Progress` is the type of the `onProgressUpdate()` parameters, and `Result` is the type of the `onPostExecute()` parameters:

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>

    protected void onPreExecute() {
        //Code to run before executing the task
    }

    protected Result doInBackground(Params... params) {
        //Code that you want to run in a background thread
    }

    protected void onProgressUpdate(Progress... values) {
        //Code that you want to run to publish the progress of your task
    }

    protected void onPostExecute(Result result) {
        //Code that you want to run when the task is complete
    }
}
```

In our example, `doInBackground()` takes `Integer` parameters, `onPostExecute()` takes a `Boolean` parameter, and we're not using the `onProgressUpdate()` method. This means that in our example, `Params` is `Integer`, `Progress` is `Void`, and `Result` is `Boolean`:

This is Void because we didn't implement the onProgressUpdate() method.

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
    ...
    protected Boolean doInBackground(Integer... drinks) {
        ...
    }

    protected void onPostExecute(Boolean... success) {
        ...
    }
}
```

We'll show you the full `UpdateDrinkTask` class on the next page.

The full UpdateDrinkTask class

Here's the full code for the `UpdateDrinkTask` class. It needs to be added to `DrinkActivity` as an inner class, but we suggest you wait to do that until we show you how to execute it and show you the full *DrinkActivity.java* code listing.

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
    private ContentValues drinkValues;

    protected void onPreExecute() {
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
        drinkValues = new ContentValues();
        drinkValues.put("FAVORITE", favorite.isChecked());
    }

    protected Boolean doInBackground(Integer... drinks) {
        int drinkId = drinks[0];
        SQLiteOpenHelper starbuzzDatabaseHelper =
                new StarbuzzDatabaseHelper(DrinkActivity.this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            db.update("DRINK", drinkValues,
                      "_id = ?", new String[] {Integer.toString(drinkId)});
            db.close();
            return true;
        } catch(SQLiteException e) {
            return false;
        }
    }

    protected void onPostExecute(Boolean success) {
        if (!success) {
            Toast toast = Toast.makeText(DrinkActivity.this,
                    "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

We've defined drinkValues as a private variable, as it's used by the onExecute() and doInBackground() methods.

Before we run the database code, we need to put the value of the favorite checkbox in the drinkValues ContentValues object.

Our database code goes in the doInBackground() method.

After the database code has run in the background, check whether it ran successfully. If it didn't, display a message.

We have to put the code to display a message in the onPostExecute() method, as it needs to be run on the main event thread to update the screen.

Execute the AsyncTask...

You run the `AsyncTask` by calling the `AsyncTask execute()` method and passing it any parameters required by the `doInBackground()` method. As an example, we want to pass the drink the user chose to our `AsyncTask`'s `doInBackground()` method, so we call it using:

```
int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);
new UpdateDrinkTask().execute(drinkId);
```

Execute the AsyncTask and pass it the drink ID.

The type of parameter you pass with the `execute()` method must match the type of parameter expected by the `AsyncTask` `doInBackground()` method. We're passing an integer value (the drink ID), which matches the type of parameter expected by our `doInBackground()` method:

```
protected Boolean doInBackground(Integer... drinks) {
    ...
}
```

...in DrinkActivity's onFavoritesClicked() method

Our `UpdateDrinkTask` class (the `AsyncTask` we created) needs to update the FAVORITE column in the Starbuzz database whenever the `favorite` checkbox in `DrinkActivity` is clicked. We therefore need to execute it in `DrinkActivity`'s `onFavoritesClicked()` method. Here's what the new version of the method looks like:

```
//Update the database when the checkbox is clicked
public void onFavoriteClicked(View view){
    int drinkId = (Integer)getIntent().getExtras().get(EXTRA_DRINKID);
    new UpdateDrinkTask().execute(drinkId);
}
```

The new version of the onFavoritesClicked() method no longer contains code to update the FAVORITE column. Instead, it calls the AsyncTask, which performs the update in the background.

We'll show you the new *DrinkActivity.java* code over the next few pages.

The full DrinkActivity.java code

Here's the complete code for *DrinkActivity.java*; update your version of the code to reflect our changes:

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.view.View;
import android.widget.CheckBox;
import android.content.ContentValues;
import android.os.AsyncTask;
```

We're using the AsyncTask class, so we need to import it.

```
public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKID = "drinkId";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

        //Get the drink from the intent
        int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);

        //Create a cursor
        SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
            Cursor cursor = db.query("DRINK",
                    new String[]{"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID", "FAVORITE"},
                    "_id = ?",
                    new String[]{Integer.toString(drinkId)},
                    null, null, null);
```

We don't need to change the onCreate() method, we're just showing it for completeness.

The code continues on the next page.

The full DrinkActivity.java code (continued)

```
        //Move to the first record in the Cursor
        if (cursor.moveToFirst()) {
            //Get the drink details from the cursor
            String nameText = cursor.getString(0);
            String descriptionText = cursor.getString(1);
            int photoId = cursor.getInt(2);
            boolean isFavorite = (cursor.getInt(3) == 1);

            //Populate the drink name
            TextView name = (TextView) findViewById(R.id.name);
            name.setText(nameText);

            //Populate the drink description
            TextView description = (TextView) findViewById(R.id.description);
            description.setText(descriptionText);

            //Populate the drink image
            ImageView photo = (ImageView) findViewById(R.id.photo);
            photo.setImageResource(photoId);
            photo.setContentDescription(nameText);

            //Populate the favorite checkbox
            CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
            favorite.setChecked(isFavorite);
        }
        cursor.close();
        db.close();
    } catch (SQLiteException e) {
        Toast toast = Toast.makeText(this,
                "Database unavailable",
                Toast.LENGTH_SHORT);
        toast.show();
    }
}
```

None of the code on this page needs to change.

The code continues on the next page.

The full DrinkActivity.java code (continued)

```
//Update the database when the checkbox is clicked
public void onFavoriteClicked(View view){
    int drinkId = (Integer) getIntent().getExtras().get(EXTRA_DRINKID);

    //Get the value of the checkbox
    CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("FAVORITE", favorite.isChecked());

    //Get a reference to the database and update the FAVORITE column
    SQLiteOpenHelper starbuzzDatabaseHelper =
            new StarbuzzDatabaseHelper(this);
    try {
        SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        db.update("DRINK",
                drinkValues,
                "_id = ?",
                new String[] {Integer.toString(drinkId)});
        db.close();
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }

    new UpdateDrinkTask().execute(drinkId);
}
```

Delete all these lines of code, as we're now using an AsyncTask for these actions.

Execute the task.

The code continues on the next page.

The full DrinkActivity.java code (continued)

Add the AsyncTask to the activity as an inner class.

```
    //Inner class to update the drink.
    private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
        private ContentValues drinkValues;

        protected void onPreExecute() {
            CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
            drinkValues = new ContentValues();
            drinkValues.put("FAVORITE", favorite.isChecked());
        }

        protected Boolean doInBackground(Integer... drinks) {
            int drinkId = drinks[0];
            SQLiteOpenHelper starbuzzDatabaseHelper =
                    new StarbuzzDatabaseHelper(DrinkActivity.this);
            try {
                SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
                db.update("DRINK", drinkValues,
                        "_id = ?", new String[] {Integer.toString(drinkId)});
                db.close();
                return true;
            } catch(SQLiteException e) {
                return false;
            }
        }

        protected void onPostExecute(Boolean success) {
            if (!success) {
                Toast toast = Toast.makeText(DrinkActivity.this,
                        "Database unavailable", Toast.LENGTH_SHORT);
                toast.show();
            }
        }
    }
}
```

Before the database code runs, put the value of the checkbox in the drinkValues ContentValues object.

Run the database code in a background thread.

Update the value of the FAVORITE column.

If the database code didn't run correctly, display a message to the user.

That's everything you need in order to create an `AsyncTask`.
Let's check what happens when we run the app.

Test drive the app

When we run the app and navigate to a drink, we can indicate that the drink is a favorite drink by checking the "favorite" checkbox. Clicking on the checkbox still updates the FAVORITE column in the database with its value, but this time the code is running on a background thread.

In an ideal world, all of your database code should run in the background. We're not going to change our other Starbuzz activities to do this, but why not make this change yourself?

there are no Dumb Questions

Q: I've written code before that just ran the database code and it was fine. Do I really need to run it in the background?

A: For really small databases, like the one in the Starbuzz app, you probably won't notice the time it takes to access the database. But that's just because the database is small. If you use a larger database, or if you run an app on a slower device, the time it takes to access the database will be significant. So yes, you should *always* run database code in the background.

Q: Remind me—why is it bad to update a view from the background thread?

A: The short answer is that it will throw an exception if you try. The longer answer is that multithreaded user interfaces are hugely buggy. Android avoided the problem by simply banning them.

Q: Which part of the database code is slowest: opening the database, or reading data from it?

A: There's no general way of knowing. If your database has a complex data structure, opening the database for the first time will take a long time because it will need to create all the tables. If you're running a complex query, that might take a very long time. In general, play it safe and run everything in the background.

Q: If it take a few seconds to read data from the database, what will the user see?

A: The user will see blank views until the database code sets the values.

Q: Why have we put the database code for just one activity in an `AsyncTask`?

A: We wanted to show you how to use `AsyncTasks` in one activity as an example. In the real world, you should do this for the database code in all your activities.

Your Android Toolbox

You've got Chapter 17 under your belt and now you've added writing to SQLite databases to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- The `CursorAdapter` `changeCursor()` method replaces the cursor currently used by a cursor adapter with a new cursor that you provide. It then closes the old cursor.
- Run your database code in a background thread using `AsyncTask`.

A summary of the AsyncTask steps

1. **onPreExecute() is used to set up the task.**
It's called before the background task begins, and runs on the main event thread.

2. **doInBackground() runs in the background thread.**
It runs immediately after `onPreExecute()`. You can specify what type of parameters it has, and what its return type is.

3. **onProgressUpdate() is used to display progress.**
It runs in the main event thread when the `doInBackground()` method calls `publishProgress()`.

4. **onPostExecute() is used to display the task outcome to the user when doInBackground has finished.**
It runs in the main event thread and takes the return value of `doInBackground()` as a parameter.

18 started services

At Your Service

There are some operations you want to keep on running, irrespective of which app has the focus. If you start downloading a file, for instance, *you don't want the download to stop when you switch to another app*. In this chapter we'll introduce you to **started services**, components that *run operations in the background*. You'll see how to create a started service using the **`IntentService`** class, and find out how its lifecycle fits in with that of an activity. Along the way, you'll discover how to **log messages**, and *keep users informed* using Android's built-in **notification service**.

Services work in the background

An Android app is a collection of activities and other components. The bulk of your app's code is there to interact with the user, but sometimes you need to do things in the background, such as download a large file, stream a piece of music, or listen for a message from the server.

These kinds of tasks aren't what activities are designed to do. In simple cases, you can create a thread, but if you're not careful your activity code will start to get complex and unreadable.

That's why **services** were invented. A service is an application component like an activity but without a user interface. They have a simpler lifecycle than activities, and they come with a bunch of features that make it easy to write code that will run in the background while the user is doing something else.

In addition to writing your own services, you can use Android's built-in ones.

Built-in services include the notification service, location service, alarm service, and download service.

There are three types of service

Services come in three main flavors:

- **Started services**
  A started service can run in the background indefinitely, even when the activity that started it is destroyed. If you wanted to download a large file from the Internet, you would probably create it as a started service. Once the operation is done, the service stops.

- **Bound services**
  A bound service is bound to another application component such as an activity. The activity can interact with it, send requests, and get results. A bound service runs as long as components are bound to it. When the components are no longer bound, the service is destroyed. If you wanted to create an odometer to measure the distance traveled by a vehicle, for example, you'd probably use a bound service. This way, any activities bound to the service could keep asking the service for updates on the distance traveled.

- **Scheduled services**
  A scheduled service is one that's scheduled to run at a particular time. As an example, from API 21, you can schedule jobs to run at an appropriate time.

In this chapter, we're going to look at how you create a started service.

We'll create a STARTED service

We're going to create a new project that contains an activity called `MainActivity`, and a started service called `DelayedMessageService`. Whenever `MainActivity` calls `DelayedMessageService`, it will wait for 10 seconds and then display a piece of text.

We're going to do this in two stages:

1. **Display the message in Android's log.**
   We'll start by displaying the message in Android's log so that we can check that the service works OK. We can look at the log in Android Studio.

2. **Display the message in a notification.**
   We'll get `DelayedMessageService` to use Android's built-in notification service to display the message in a notification.

Create the project

We'll start by creating the project. Create a new Android project for an application named "Joke" with a company domain of "hfad.com", making the package name `com.hfad.joke`. The minimum SDK should be API 19 so that it will work with most devices. You'll need an empty activity named "MainActivity" and a layout named "activity_main" so that your code matches ours. **Make sure that you uncheck the Backwards Compatibility (AppCompat) option when you create the activity.**

The next thing we need to do is create the service.

Log
Display notification

Use the IntentService class to create a basic started service

The simplest way of creating a started service is to extend the **`IntentService`** class, as it provides you with most of the functionality you need. You start it with an intent, and it runs the code that you specify in a separate thread.

We're going to add a new intent service to our project. To do this, switch to the Project view of Android Studio's explorer, click on the `com.hfad.joke` package in the *app/src/main/java* folder, go to File→New..., and select the Service option. When prompted, choose the option to create a new Intent Service. Name the service "DelayedMessageService" and uncheck the option to include helper start methods to minimize the amount of code that Android Studio generates for you. Click on the Finish button, then replace the code in *DelayedMessageService.java* with the code here:

Some versions of Android Studio may ask you what the source language should be. If prompted, select the option for Java.

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;

public class DelayedMessageService extends IntentService {

    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        //Code to do something
    }
}
```

Extend the IntentService class.

Put the code you want the service to run in the onHandleIntent() method.

The above code is all you need to create a basic intent service. You extend the `IntentService` class, add a public constructor, and implement the `onHandleIntent()` method.

The `onHandleIntent()` method should contain the code you want to run each time the service is passed an intent. It runs in a separate thread. If it's passed multiple intents, it deals with them one at a time.

We want `DelayedMessageService` to display a message in Android's log, so let's look at how you log messages.

Log
Display notification

How to log messages

Adding messages to a log can be a useful way of checking that your code works the way you want. You tell Android what to log in your Java code, and when the app's running, you check the output in Android's log (a.k.a. logcat).

You log messages using one of the following methods in the `Android.util.Log` class:

Log.v(String tag, String message)	Logs a verbose message.
Log.d(String tag, String message)	Logs a debug message.
Log.i(String tag, String message)	Logs an information message.
Log.w(String tag, String message)	Logs a warning message.
Log.e(String tag, String message)	Logs an error message.

There's also a Log.wtf() method you can use to report exceptions that should never happen. According to the Android documentation, wtf means "What a Terrible Failure." We know it really means "Welcome to Fiskidagurinn," which refers to the Great Fish Day festival held annually in Dalvik, Iceland. Android developers can often be heard to say, "My AVD just took 8 minutes to boot up. WTF??" as a tribute to the small town that gave its name to the standard Android executable bytecode format.

Each message is composed of a String tag you use to identify the source of the message, and the message itself. As an example, to log a verbose message that's come from `DelayedMessageService`, you use the `Log.v()` method like this:

```
Log.v("DelayedMessageService", "This is a message");
```

You can view the logcat in Android Studio and filter by the different types of message. To see the logcat, select the Android Monitor option at the bottom of your project screen in Android Studio and then select the logcat tab:

Log
Display notification

The full DelayedMessageService code

We want our service to get a piece of text from an intent, wait for 10 seconds, then display the piece of text in the log. To do this, we'll add a `showText()` method to log the text, and then call it from the `onHandleIntent()` method after a delay of 10 seconds.

Here's the full code for *DelayedMessageService.java* (update your version of the code to reflect our changes):

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";

    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }

    private void showText(final String text) {
        Log.v("DelayedMessageService", "The message is: " + text);
    }
}
```

We're using the Log class, so we need to import it.

Use a constant to pass a message from the activity to the service.

Call the super constructor.

This method contains the code you want to run when the service receives an intent.

Wait 10 seconds.

Get the text from the intent

Call the showText() method.

This logs a piece of text so we can see it in the logcat through Android Studio.

Log
Display notification

You declare services in AndroidManifest.xml

Just like activities, each service needs to be declared in *AndroidManifest.xml* so that Android can call it; if a service isn't declared in this file, Android won't know it's there and won't be able to call it.

Android Studio should update *AndroidManifest.xml* for you automatically whenever you create a new service by adding a new `<service>` element. Here's what our *AndroidManifest.xml* code looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.joke">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".DelayedMessageService"
            android:exported="false">
        </service>
    </application>
</manifest>
```

Don't worry if this code looks different than yours.

You declare a service in AndroidManifest.xml like this. Android Studio should do this for you automatically.

The service name has a "." in front of it so that Android can combine it with the package name to derive the fully qualified class name.

The `<service>` element contains two attributes: `name` and `exported`. The `name` attribute tells Android what the name of the service is—in our case, `DelayedMessageService`. The `exported` attribute tells Android whether the service can be used by other apps. Setting it to `false` means that the service will only be used within the current app.

Now that we have a service, let's get `MainActivity` to start it.

Add a button to activity_main.xml

We're going to get `MainActivity` to start `DelayedMessageService` whenever a button is clicked, so we'll add the button to `MainActivity`'s layout.

First, add the following values to *strings.xml*:

```
<string name="question">What is the secret of comedy?</string>
<string name="response">Timing!</string>
```

Then, replace your *activity_main.xml* code with ours below so that `MainActivity` displays a button:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context="com.hfad.joke.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="@string/question"
        android:id="@+id/button"
        android:onClick="onClick"/>

</LinearLayout>
```

This creates a button. When it's clicked, the onClick() method in the activity will get called.

The button will call an `onClick()` method whenever the user clicks it, so we'll add this method to `MainActivity`.

Log
Display notification

You start a service using startService()

We'll use `MainActivity`'s `onClick()` method to start `DelayedMessageService` whenever the user clicks on the button. You start a service from an activity in a similar way to how you start an activity. You create an explicit intent that's directed at the service you want to start, then use the **`startService()`** method in your activity to start it:

```
Intent intent = new Intent(this, DelayedMessageService.class);
startService(intent);
```

Starting a service is just like starting an activity, except you use startService() instead of startActivity().

Here's our code for *MainActivity.java*; update your version to match ours:

```
package com.hfad.joke;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, DelayedMessageService.class);
        intent.putExtra(DelayedMessageService.EXTRA_MESSAGE,
                                getResources().getString(R.string.response));
        startService(intent);
    }
}
```

We're using these classes, so we need to import them.

We're using Activity here, but you could use AppCompatActivity instead.

This will run when the button gets clicked.

Create the intent.

Add text to the intent.

Start the service.

That's all the code we need to get our activity to start the service. Before we take it for a test drive, let's go through what happens when the code runs.

Log
Display notification

What happens when you run the app

Here's what the code does when we run the app:

1. **MainActivity starts DelayedMessageService by calling startService() and passing it an intent.**
   The intent contains the message `MainActivity` wants `DelayedMessageService` to display, in this case "Timing!".

2. **When DelayedMessageService receives the intent, its onHandleIntent() method runs.**
   `DelayedMessageService` waits for 10 seconds.

3. **DelayedMessageService logs the message.**

4. **When DelayedMessageService has finished running, it's destroyed.**

Let's take the app for a test drive so we can see it working.

Test drive the app

Log
Display notification

When you run the app, `MainActivity` is displayed. It contains a single button:

Here's the button.

Press the button, switch back to Android Studio, and watch the logcat output in the bottom part of the IDE. After 10 seconds, the message "Timing!" should appear in the logcat.

```
03-22 12:39:26.294 12381-18456/com.hfad.joke V/DelayedMessageService: The message is: Timing!
```

After a 10-second delay, the message is displayed in the log.

Now that you've seen `DelayedMessageService` running, let's look in more detail at how started services work.

The states of a started service

When an application component (such as an activity) starts a service, the service moves from being created to running to being destroyed.

A started service spends most of its life in a ***running*** state; it's been started by another component such as an activity, and it runs code in the background. It continues to run even if the component that started it is destroyed. When the service has finished running code, it's ***destroyed***.

Just like an activity, when a service moves from being created to being destroyed, it triggers key service lifecycle methods, which it inherits.

When the service is created, its `onCreate()` method gets called. You override this method if you want to perform any tasks needed to set up the service.

When the service is ready to start, its `onStartCommand()` method is called. If you're using an `IntentService` (which is usually the case for a started service), you don't generally override this method. Instead, you add any code you want the service to run to its `onHandleIntent()` method, which is called after `onStartCommand()`.

The `onDestroy()` method is called when the started service is no longer running and it's about to be destroyed. You override this method to perform any final cleanup tasks, such as freeing up resources.

We'll take a closer look at how these methods fit into the service states on the next page.

A started service runs after it's been started.

onCreate() gets called when the service is first created, and it's where you do any service setup.

onDestroy() gets called just before the service gets destroyed.

The started service lifecycle: from create to destroy

Here's an overview of the started service lifecycle from birth to death.

1. **The component calls startService() and the service gets created.**

2. **The onCreate() method runs immediately after the service is created.**
   The `onCreate()` method is where any service initialization code should go, as this method always gets called after the service has launched but before it starts running.

3. **The onStartCommand() method runs when the service is about to start.**
   If your started service extends the `IntentService` class (which is usually the case), the `onStartCommand()` method creates a separate thread and `onHandleIntent()` is called. You add any code you want the service to run in the background to `onHandleIntent()`.

4. **The service spends most of its life running.**

5. **The onDestroy() method runs when the service has finished running, immediately before it's destroyed.**
   The `onDestroy()` method enables you to perform any final cleanup tasks such as freeing up resources.

6. **After the onDestroy() method has run, the service is destroyed.**
   The service ceases to exist.

The `onCreate()`, `onStartCommand()`, and `onDestroy()` methods are three of the main service lifecycle methods. So where do these methods come from?

Your service inherits the lifecycle methods

As you saw earlier in the chapter, the started service you created extends the `android.app.IntentService` class. This class gives your service access to the Android lifecycle methods. Here's a diagram showing the class hierarchy:

Context abstract class
(android.content.Context)
An interface to global information about the application environment. Allows access to application resources, classes, and operations.

ContextWrapper class
(android.content.ContextWrapper)
A proxy implementation for the `Context`.

Service class
(android.app.Service)
The `Service` class implements default versions of the lifecycle methods. You'll find out more about it in the next chapter.

IntentService class
(android.app.IntentService)
The `IntentService` class gives you an easy way of creating started services. It includes a method, `onHandleIntent()`, which handles any intents it's given on a background thread.

YourStartedService class
(com.hfad.foo)
Most of the behavior of your started service is handled by superclass methods your service inherits. All you do is override the methods you need and add a public constructor.

Now that you understand more about how started services work behind the scenes, have a go at the following exercise. After that, we'll look at how we can make `DelayedMessageService` display its message in a notification.

Service Magnets

Below is most of the code needed to create a started service called `WombleService` that plays a *.mp3* file in the background, and an activity that uses it. See if you can finish off the code.

This is the service.

```
public class WombleService extends ..............................{

    public WombleService() {
        super("WombleService");
    }

    @Override
    protected void ..............................(Intent intent) {
        MediaPlayer mediaPlayer =
                  MediaPlayer.create(getApplicationContext(), R.raw.wombling_song);
        mediaPlayer.start();
    }
}
```

This uses the Android MediaPlayer class to play a file called wombling_song.mp3. The file is located in the res/raw folder.

This is the activity.

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, ......................................);

        ........................(intent);
    }
}
```

You won't need to use all of these magnets.

`IntentService` `onHandleIntent` `startActivity` `startService`

`WombleService.class` `Underground` `Overground` `WombleService`

Service Magnets Solution

Below is most of the code needed to create a started service called `WombleService` that plays a *.mp3* file in the background, and an activity that uses it. See if you can finish off the code.

```
public class WombleService extends [IntentService] {

    public WombleService() {
        super("WombleService");
    }

    @Override
    protected void [onHandleIntent](Intent intent) {
        MediaPlayer mediaPlayer =
                  MediaPlayer.create(getApplicationContext(), R.raw.wombling_song);
        mediaPlayer.start();
    }
}
```

This is the service. It extends the IntentService class.

The code needs to run in the onHandleIntent() method.

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, [WombleService.class]);
        [startService](intent);
    }
}
```

This is the activity.

Create an explicit intent directed at WombleService.class.

Start the service.

You didn't need to use these magnets.

startActivity

Underground

Overground

WombleService

Log
Display notification

Android has a built-in notification service

We're going to change our Joke app so that our message gets displayed in a **notification**. Notifications are messages that are displayed outside the app's user interface. When a notification gets issued, it's displayed as an icon in the notification area of the status bar. You can see details of the notification in the notification drawer, which you access by swiping down from the top of the screen:

Unlike a toast or snackbar, notifications are available outside the app that issues them, so the user can access them no matter what app they're currently using (if any). They're much more configurable than toasts and snackbars, too.

To display the notification, we're going to use one of Android's built-in services, the notification service. You'll see how to do this over the next few pages.

Log
Display notification

We'll use notifications from the AppCompat Support Library

We're going to create notifications using classes from the AppCompat Support Library so that our notifications will work consistently across a wide range of Android versions. While it's possible to create notifications using classes from the main release of Android, recent changes to these classes mean that the newest features won't be available on older versions.

Before we can use the notification classes from the Support Library, we need to add it to our project as a dependency. To do this, choose File→Project Structure, then click on the app module and choose Dependencies. Android Studio may have already added the AppCompat Support Library for you automatically. If so, you will see it listed as appcompat-v7. If it hasn't been added, you will need to add it yourself. Click on the "+" button at the bottom or right side of the screen, choose the Library Dependency option, select the appcompat-v7 library, and then click on the OK button. Click on OK again to save your changes and close the Project Structure window.

Use notifications from the AppCompat Support Library to allow apps running on older versions of Android to include the newest features.

To get `DelayedMessageService` to display a notification, there are three things we need to do: create a notification builder, tell the notification to start `MainActivity` when it's clicked, and issue the notification. We'll build up the code over the next few pages, then show you the full code at the end.

Log
Display notification

First create a notification builder

The first thing we need to do is create a notification builder. This enables you to build a notification with specific content and features.

Each notification you create must include a small icon, a title, and some text as a bare minimum. Here's the code to do that:

The NotificationCompat class comes from the AppCompat Support Library.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
    .setSmallIcon(android.R.drawable.sym_def_app_icon)
    .setContentTitle(getString(R.string.question))
    .setContentText(text);
```

This displays a small icon, in this case a built-in Android icon.

Set the title and text.

To add more features to the notification, you simply add the appropriate method call to the builder. As an example, here's how you additionally specify that the notification should have a high priority, vibrate the device when it appears, and disappear when the user clicks on it:

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
            .setSmallIcon(android.R.drawable.sym_def_app_icon)
            .setContentTitle(getString(R.string.question))
            .setContentText(text)
            .setPriority(NotificationCompat.PRIORITY_HIGH)
            .setVibrate(new long[] {0, 1000})
            .setAutoCancel(true);
```

Make it a high priority and vibrate the device.

Simply chain the method calls together to add more features to the notification.

This makes the notification disappear when the user clicks on it.

Wait for 0 milliseconds before vibrating the device for 1,000 milliseconds.

These are just some of the properties that you can set. You can also set properties such as the notification's visibility to control whether it appears on the device lock screen, a number to display in case you want to send many notifications from the same app, and whether it should play a sound. You can find out more about these properties (and many others) here:

https://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html

Next, we'll add an action to the notification to tell it which activity to start when it's clicked.

You create a heads-up notification (one that appears in a small floating window) by setting its priority to high, and making it vibrate the device or play a sound.

Log
Display notification

Add an action to tell the notification which activity to start when clicked

When you create a notification, it's a good idea to add an action to it, specifying which activity in your app should be displayed when the user clicks on the notification. As an example, an email app might issue a notification when the user receives a new email, and display the contents of that email when the user clicks on it. In our particular case, we're going to start `MainActivity`.

You add an action by creating a **pending intent** to start an activity, which you then add to the notification. A pending intent is an intent that your app can pass to other applications. The application can then submit the intent on your app's behalf at a later time.

To create a pending intent, you first create an explicit intent directed to the activity you want to start when the notification is clicked. In our case, we want to start `MainActivity`, so we use:

This is a normal intent that starts MainActivity.

```
Intent actionIintent = new Intent(this, MainActivity.class);
```

We then use that intent to create a pending intent using the `PendingIntent.getActivity()` method.

```
PendingIntent actionPendingIntent = PendingIntent.getActivity(
    this,
    0,
    actionIntent,
    PendingIntent.FLAG_UPDATE_CURRENT);
```

A context, in this case the current service.

This is a flag that's used if you ever need to retrieve the pending intent. We don't need to, so we're setting it to 0.

This is the intent we created above.

This means that if there's a matching pending intent, it will be updated.

The `getActivity()` method takes four parameters: a context (usually `this`), an `int` request code, the explicit intent we defined above, and a flag that specifies the pending intent's behavior. In the above code, we've used a flag of `FLAG_UPDATE_CURRENT`. This means that if a matching pending intent already exists, its extra data will be updated with the contents of the new intent. Other options are `FLAG_CANCEL_CURRENT` (cancel any existing matching pending intents before generating a new one), `FLAG_NO_CREATE` (don't create the pending intent if there's no matching existing one), and `FLAG_ONE_SHOT` (you can only use the pending intent once).

Once you've created the pending intent, you add it to the notification using the notification builder `setContentIntent()` method:

```
builder.setContentIntent(actionPendingIntent);
```

This adds the pending intent to the notification.

This tells the notification to start the activity specified in the intent when the user clicks on the notification.

Log
Display notification

Issue the notification using the built-in notification service

Finally, you issue the notification using Android's notification service.

To do this, you first need to get a **`NotificationManager`**. You do this by calling the `getSystemService()` method, passing it a parameter of `NOTIFICATION_SERVICE`:

This gives you access to Android's notification service.

```
NotificationManager notificationManager =
                (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

You then use the notification manager to issue the notification by calling its **`notify()`** method. This takes two parameters: a notification ID and a `Notification` object.

The notification ID is used to identify the notification. If you send another notification with the same ID, it will replace the current notification. This is useful if you want to update an existing notification with new information.

You create the `Notification` object by calling the notification builder's **`build()`** method. The notification it builds includes all the content and features you've specified via the notification builder.

Here's the code to issue the notification:

This is an ID we'll use for the notification. It's a random number we made up.

```
public static final int NOTIFICATION_ID = 5453;

...

NotificationManager notificationManager =
                (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFICATION_ID, builder.build());
```

Use the notification service to display the notification we created.

That's everything we need to create and issue notifications. We'll show you the full code for `DelayedMessageService` on the next page.

The full code for DelayedMessageService.java

Here's the full code for *DelayedMessageService.java*. It now uses a notification to display a message to the user. Update your code to match ours:

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;
import android.support.v4.app.NotificationCompat;
import android.app.PendingIntent;
import android.app.NotificationManager;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";
    public static final int NOTIFICATION_ID = 5453;

    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }
```

Delete this line.

We're using these extra classes, so we need to import them.

This is used to identify the notification. It could be any number; we just decided on 5453.

The code continues on the next page.

The DelayedMessageService.java code (continued)

```
    private void showText(final String text) {
        Log.v("DelayedMessageService", "The message is: " + text);

        //Create a notification builder
        NotificationCompat.Builder builder =
                new NotificationCompat.Builder(this)
                        .setSmallIcon(android.R.drawable.sym_def_app_icon)
                        .setContentTitle(getString(R.string.question))
                        .setContentText(text)
                        .setPriority(NotificationCompat.PRIORITY_HIGH)
                        .setVibrate(new long[] {0, 1000})
                        .setAutoCancel(true);

        //Create an action
        Intent actionIntent = new Intent(this, MainActivity.class);
        PendingIntent actionPendingIntent = PendingIntent.getActivity(
                                            this,
                                            0,
                                            actionIntent,
                                            PendingIntent.FLAG_UPDATE_CURRENT);
        builder.setContentIntent(actionPendingIntent);

        //Issue the notification
        NotificationManager notificationManager =
                (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(NOTIFICATION_ID, builder.build());
    }
}
```

Use a notification builder to specify the content and features of the notification.

Create an intent.

Use the intent to create a pending intent.

Add the pending intent to the notification.

Display the notification using a notification manager.

That's all the code we need for our started service. Let's go through what happens when the code runs.

Log
Display notification

What happens when you run the code

Before you try running the updated app, let's go through what happens when the code runs:

1. **MainActivity starts DelayedMessageService by calling startService() and passing it an intent.**
   The intent contains the message `MainActivity` wants `DelayedMessageService` to display.

2. **DelayedMessageService waits for 10 seconds.**

3. **DelayedMessageService creates a notification builder and sets details of how the notification should be configured.**

4. **DelayedMessageService creates an intent for MainActivity, which it uses to create a pending intent.**

Log
Display notification

The story continues

5 **DelayedMessageService adds the pending intent to the notification builder.**

6 **DelayedMessageService creates a NotificationManager object and calls its notify() method.**

The notification service displays the notification built by the notification builder.

7 **When the user clicks on the notification, the notification uses its pending intent to start MainActivity.**

Now that we've gone through what the code does, let's take the app for a test drive.

Test drive the app

Log
Display notification

When you click on the button in `MainActivity`, a notification is displayed after 10 seconds. You'll receive the notification irrespective of which app you're in.

When you open the notification drawer and click on the notification, Android returns you to `MainActivity`.

You now know how to create a started service that displays a notification using the Android notification service. In the next chapter, we'll look at how you create a bound service.

Your Android Toolbox

You've got Chapter 18 under your belt and now you've added started services to your toolbox.

BULLET POINTS

- A **service** is an application component that can perform tasks in the background. It doesn't have a user interface.
- A **started service** can run in the background indefinitely, even when the activity that started it is destroyed. Once the operation is done, it stops itself.
- A **bound service** is bound to another component such as an activity. The activity can interact with it and get results.
- A **scheduled service** is one that's scheduled to run at a particular time.
- You can create a simple started service by extending the **`IntentService`** class, overriding its `onHandleIntent()` method and adding a public constructor.
- You declare services in *AndroidManifest.xml* using the **`<service>`** element.
- You start a started service using the **`startService()`** method.
- When a started service is created, its `onCreate()` method gets called, followed by `onStartCommand()`. If the service is an `IntentService`, `onHandleIntent()` is then called in a separate thread. When the service has finished running, `onDestroy()` gets called before the service is destroyed.
- The `IntentService` class inherits lifecycle methods from the **`Service`** class.
- You log messages using the **`Android.util.Log`** class. You can view these messages in the logcat in Android Studio.
- You create a notification using a **notification builder**. Each notification must include a small icon, a title, and some text as a bare minimum.
- A **heads-up** notification has its priority set to high, and vibrates the device or plays a sound when it's issued.
- You tell the notification which activity to start when it's clicked by creating a **pending intent** and adding it to the notification as an action.
- You issue the notification using a **notification manager**. You create a notification manager using Android's notification service.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

19 bound services and permissions

Bound Together

Started services are great for background operations, but what if you need a service that's more interactive? In this chapter you'll discover how to create a **bound service**, *a type of service your activity can interact with*. You'll see how to ***bind*** to the service when you need it, and how to ***unbind*** from it when you're done to save resources. You'll find out how to use **Android's Location Services** to get *location updates from your device GPS*. Finally, you'll discover how to use **Android's permission model**, including *handling runtime permission requests*.

Bound services are bound to other components

As you saw in Chapter 18, a started service is one that starts when it's passed an intent. It runs code in the background, and stops when the operation is complete. It continues running even if the component that starts it gets destroyed.

A **bound service** is one that's bound to another application component, such as an activity. Unlike a started service, the component can interact with the bound service and call its methods.

To see this in action, we're going to create a new odometer app that uses a bound service. We'll use Android's location service to track the distance traveled:

On the next page we'll look at the steps we'll go through to create the app.

Here's what we're going to do

We're going to build the app in three main steps:

1. **Create a basic version of a bound service called OdometerService.**
   We'll add a method to it, `getDistance()`, which will return a random number.

2. **Get an activity, MainActivity, to bind to OdometerService and call its getDistance() method.**
   We'll call the method every second, and update a text view in `MainActivity` with the results.

3. **Update OdometerService to use Android's Location Services.**
   The service will get updates on the user's current location, and use these to calculate the distance traveled.

Create a new Odometer project

We'll start by creating the project. Create a new Android project for an application named "Odometer" with a company domain of "hfad.com", making the package name `com.hfad.odometer`. The minimum SDK should be API 19 so that it will work with most devices. You'll need an empty activity named "MainActivity" and a layout named "activity_main" so that your code matches ours. **Make sure that you uncheck the Backwards Compatibility (AppCompat) option when you create the activity.**

Create a new service

You create a bound service by extending the **`Service`** class. This class is more general than the `IntentService` class we used in Chapter 18, which is used for started services. Extending `Service` gives you more flexibility, but requires more code.

We're going to add a new bound service to our project, so switch to the Project view of Android Studio's explorer, click on the `com.hfad.odometer` package in the *app/src/main/java* folder, go to File→New..., and select the Service option. When prompted, choose the option to create a new Service (not an Intent Service), and name the service "OdometerService". Uncheck the "Exported" option, as this only needs to be true if you want services outside this app to access the service. Make sure that the "checkednabled" option is checked; if it isn't, the activity won't be able to run the app. Then replace the code in *OdometerService.java* with this (shown here in bold):

Some versions of Android Studio may ask you what the source language should be. If prompted, select the option for Java.

```
package com.hfad.odometer;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class OdometerService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        //Code to bind the service
    }
}
```

The class extends the Service class.

The onBind() method is called when a component wants to bind to the service.

The above code implements one method, `onBind()`, which gets called when a component, such as an activity, wants to bind to the service. It has one parameter, an `Intent`, and returns an `IBinder` object.

`IBinder` is an interface that's used to bind your service to the activity, and you need to provide an implementation of it in your service code. We'll look at how you do this next.

Implement a binder

You implement the `IBinder` by adding a new inner class to your service code that extends the `Binder` class (which implements the `IBinder` interface). This inner class needs to include a method that activities can use to get a reference to the bound service.

We're going to define a binder called `OdometerBinder` that `MainActivity` can use to get a reference to `OdometerService`. Here's the code we'll use to define it:

```
public class OdometerBinder extends Binder {
    OdometerService getOdometer() {
        return OdometerService.this;
    }
}
```

When you create a bound service, you need to provide a Binder implementation.

The activity will use this method to get a reference to the OdometerService.

We need to return an instance of the `OdometerBinder` in `OdometerService`'s `onBind()` method. To do this, we'll create a new private variable for the binder, instantiate it, and return it in the `onBind()` method. Update your *OdometerService.java* code to include our changes below:

```
...
import android.os.Binder;

public class OdometerService extends Service {

    private final IBinder binder = new OdometerBinder();

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```

We're using this extra class, so we need to import it.

We're using a private final variable for our IBinder.object

This is our IBinder implementation.

Return the IBinder.

We've now written all the service code we need to allow `MainActivity` to bind to `OdometerService`. Next, we'll add a new method to the service to make it return a random number.

OdometerService
MainActivity
Location Services

Add a getDistance() method to the service

We're going to add a method to `OdometerService` called `getDistance()`, which our activity will call. We'll get it to return a random number for now, and later on we'll update it to use Android's location services.

Here's the full code for *OdometerService.java* including this change; update your version to match ours:

```
package com.hfad.odometer;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.Binder;
import java.util.Random;

public class OdometerService extends Service {

    private final IBinder binder = new OdometerBinder();
    private final Random random = new Random();

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    public double getDistance() {
        return random.nextDouble();
    }
}
```

We're using this extra class, so we need to import it.

We'll use a Random() object to generate random numbers.

Add the getDistance() method.

Return a random double.

Next we'll update `MainActivity` so that it uses `OdometerService`.

Update MainActivity's layout

The next step in creating our app is to get `MainActivity` to bind to `OdometerService` and call its `getDistance()` method. We're going to start by adding a text view to `MainActivity`'s layout. This will display the number returned by `OdometerService`'s `getDistance()` method.

Update your version of *activity_main.xml* to reflect our changes:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.hfad.odometer.MainActivity"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/distance"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="48sp"
        android:layout_gravity="center_horizontal"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>
```

We'll use the TextView to display the number returned by the OdometerService getDistance() method.

Now that we've added a text view to `MainActivity`'s layout, we'll update its activity code. Let's go through the changes we need to make.

What MainActivity needs to do

To get an activity to connect to a bound service and call its methods, there are a few steps you need to perform:

1. **Create a ServiceConnection.**
   This uses the service's `IBinder` object to form a connection with the service.

2. **Bind the activity to the service.**
   Once you've bound it to the service, you can call the service's methods directly.

3. **Interact with the service.**
   In our case, we'll use the service's `getDistance()` method to update the activity's text view.

When your activity is bound to the service, you can use it to update your activity.

4. **Unbind from the service when you've finished with it.**
   When the service is no longer used, Android destroys the service to free up resources.

We'll go through these steps with `MainActivity`, starting with creating the `ServiceConnection`.

Create a ServiceConnection

A **ServiceConnection** is an interface that enables your activity to bind to a service. It has two methods that you need to define: `onServiceConnected()` and `onServiceDisconnected()`. The `onServiceConnected()` method is called when a connection to the service is established, and `onServiceDisconnected()` is called when it disconnects.

We need to add a `ServiceConnection` to `MainActivity`. Here's what the basic code looks like; update your version of *MainActivity.java* to matches ours:

```
package com.hfad.odometer;

import android.app.Activity;
import android.os.Bundle;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.content.ComponentName;

public class MainActivity extends Activity {

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            //Code that runs when the service is connected
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            //Code that runs when the service is disconnected
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

We're using these classes, so we need to import them.

We're using an Activity here, but you could use AppCompatActivity instead.

Create a Service Connection object.

You need to define these methods.

Add MainActivity's onCreate() method.

We'll update the `onServiceConnected()` and `onServiceDisconnected()` methods on the next page.

The onServiceConnected() method

OdometerService
MainActivity
Location Services

As we said on the previous page, the `onServiceConnected()` method is called when a connection is established between the activity and the service. It takes two parameters: a `ComponentName` object that describes the service that's been connected to, and an `IBinder` object that's defined by the service:

The ComponentName identifies the service. It includes the service package and class names.

```
@Override
public void onServiceConnected(ComponentName componentName, IBinder binder) {
    //Code that runs when the service is connected
}
```

This is an IBinder defined by the service. We added one to OdometerService earlier.

There are two things we need the `onServiceConnected()` method to do:

- Use its `IBinder` parameter to get a reference to the service we're connected to, in this case `OdometerService`. We can do this by casting the `IBinder` to an `OdometerService.OdometerBinder` (as this is the type of `IBinder` we defined in `OdometerService`) and calling its `getOdometer()` method.
- Record that the activity is bound to the service.

Here's the code to do these things (update your version of *MainActivity.java* to include our changes):

```
public class MainActivity extends Activity {

    private OdometerService odometer;
    private boolean bound = false;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            OdometerService.OdometerBinder odometerBinder =
                    (OdometerService.OdometerBinder) binder;
            odometer = odometerBinder.getOdometer();
            bound = true;
        }
        ...
    };
...
}
```

Add these variables to record a reference to the service, and whether the activity is bound to it.

Use the IBinder to get a reference to the service.

The activity is bound to the service, so set the bound variable to true.

The onServiceDisconnected() method

The `onServiceDisconnected()` method is called when the service and the activity are disconnected. It takes one parameter, a `ComponentName` object that describes the service:

```
@Override
public void onServiceDisconnected(ComponentName componentName) {
    //Code that runs when the service is disconnected
}
```

There's only one thing we need the `onServiceDisconnected()` method to do when it's called: record that the activity is no longer bound to the service. Here's the code to do that; update your version of *MainActivity.java* to match ours:

```
public class MainActivity extends Activity {

    private OdometerService odometer;
    private boolean bound = false;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            OdometerService.OdometerBinder odometerBinder =
                    (OdometerService.OdometerBinder) binder;
            odometer = odometerBinder.getOdometer();
            bound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            bound = false;
        }
    };
...
}
```

Set bound to false, as MainActivity is no longer bound to OdometerService.

Next we'll look at how you bind to and unbind from the service.

Use bindService() to bind the service

OdometerService
MainActivity
Location Services

When you bind your activity to a service, you usually do it in one of two places:

- In the activity's `onStart()` method when the activity becomes visible. This is appropriate if you only need to interact with the service when it's visible.
- In the activity's `onCreate()` method when the activity gets created. Do this if you need to receive updates from the service even when the activity's stopped.

You don't usually bind to a service in the activity's onResume() method in order to keep the processing done in this method to a minimum.

In our case, we only need to display updates from `OdometerService` when `MainActivity` is visible, so we'll bind to the service in its `onStart()` method.

To bind to the service, you first create an explicit intent that's directed at the service you want to bind to. You then use the activity's `bindService()` method to bind to the service, passing it the intent, the `ServiceConnection` object defined by the service, and a flag to describe how you want to bind.

To see how you do this, here's the code you'd use to bind `MainActivity` to `OdometerService` (we'll add this code to *MainActivity.java* a few pages ahead):

```
@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, OdometerService.class);
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
}
```

This is an intent directed to the OdometerService.

This is the ServiceConnection object.

The bindService() method uses the intent and service connection to bind the activity to the service.

In the above code, we've used the flag `Context.BIND_AUTO_CREATE` to tell Android to create the service if it doesn't already exist. There are other flags you can use instead; you can see all the available ones in the Android documentation here:

https://developer.android.com/reference/android/content/Context.html

Next, we'll look at how you unbind the activity from the service.

Use unbindService() to unbind from the service

When you unbind your activity from a service, you usually add the code to do so to your activity's `onStop()` or `onDestroy()` method. The method you use depends on where you put your `bindService()` code:

- If you bound to the service in your activity's `onStart()` method, unbind from it in the `onStop()` method.
- If you bound to the service in your activity's `onCreate()` method, unbind from it in the `onDestroy()` method.

In our case, we used `MainActivity`'s `onStart()` method to bind to `OdometerService`, so we'll unbind from it in the activity's `onStop()` method.

You unbind from a service using the `unbindService()` method. The method takes one parameter, the `ServiceConnection` object. Here's the code that we need to add to `MainActivity` (we'll add this code to *MainActivity.java* a few pages ahead):

```
@Override
protected void onStop() {
    super.onStop();
    if (bound) {
        unbindService(connection);
        bound = false;
    }
}
```

This uses the ServiceConnection object to unbind from the service.

When we unbind, we'll set bound to false.

In the above code we're using the value of the `bound` variable to test whether or not we need to unbind from the service. If `bound` is `true`, this means `MainActivity` is bound to `OdometerService`. We need to unbind the service, and set the value of `bound` to `false`.

So far we have an activity that binds to the service when the activity starts, and unbinds from it when the activity stops. The final thing we need to do is get `MainActivity` to call `OdometerService`'s `getDistance()` method, and display its value.

Call OdometerService's getDistance() method

OdometerService
MainActivity
Location Services

Once your activity is bound to the service, you can call its methods. We're going to call the `OdometerService`'s `getDistance()` method every second, and update `MainActivity`'s text view with its value.

To do this, we're going to write a new method called `displayDistance()`. This will work in a similar way to the `runTimer()` code we used in Chapters 4 and 11.

Here's our `displayDistance()` method. We'll add it to *MainActivity.java* a couple of pages ahead:

```
private void displayDistance() {
    final TextView distanceView = (TextView) findViewById(R.id.distance);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            double distance = 0.0;
            if (bound && odometer != null) {
                distance = odometer.getDistance();
            }
            String distanceStr = String.format(Locale.getDefault(),
                                               "%1$,.2f miles", distance);
            distanceView.setText(distanceStr);
            handler.postDelayed(this, 1000);
        }
    });
}
```

Get the TextView.

Create a new Handler.

Call the Handler's post() method, passing in a new Runnable.

If we've got a reference to the OdometerService and we're bound to it, call getDistance().

You could use a String resource for "miles", but we've hardcoded it here for simplicity.

Post the code in the Runnable to be run again after a delay of 1 second. As this line of code is included in the Runnable run() method, it will run every second (with a slight lag).

We'll call the `displayDistance()` method in `MainActivity`'s `onCreate()` method so that it starts running when the activity gets created (we'll add this code to *MainActivity.java* on the next page):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    displayDistance();
}
```

Call displayDistance() in MainActivity's onCreate() method to kick it off.

We'll show you the full code for `MainActivity` on the next page.

The full MainActivity.java code

Here's the complete code for *MainActivity.java*; make sure your version of the code matches ours:

```
package com.hfad.odometer;

import android.app.Activity;
import android.os.Bundle;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.widget.TextView;
import java.util.Locale;

public class MainActivity extends Activity {

    private OdometerService odometer;
    private boolean bound = false;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            OdometerService.OdometerBinder odometerBinder =
                    (OdometerService.OdometerBinder) binder;
            odometer = odometerBinder.getOdometer();
            bound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            bound = false;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        displayDistance();
    }
```

We're using these extra classes, so we need to import them.

Use this for the OdometerService.

Use this to store whether or not the activity's bound to the service.

We need to define a ServiceConnection.

Get a reference to the OdometerService when the service is connected.

Call the displayDistance() method when the activity is created.

The code continues on the next page.

The MainActivity.java code (continued)

OdometerService
MainActivity
Location Services

```
    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, OdometerService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }
```

Bind the service when the activity starts.

```
    @Override
    protected void onStop() {
        super.onStop();
        if (bound) {
            unbindService(connection);
            bound = false;
        }
    }
```

Unbind the service when the activity stops.

Display the value returned by the service's getDistance() method.

```
    private void displayDistance() {
        final TextView distanceView = (TextView)findViewById(R.id.distance);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                double distance = 0.0;
                if (bound && odometer != null) {
                    distance = odometer.getDistance();
                }
                String distanceStr = String.format(Locale.getDefault(),
                                                   "%1$,.2f miles", distance);
                distanceView.setText(distanceStr);
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

Call OdometerService's getDistance() method.

Update the TextView's value every second.

That's all the code you need to get `MainActivity` to use `OdometerService`. Let's go through what happens when you run the code.

What happens when you run the code

Before you see the app up and running, let's walk through what the code does.

1. **When MainActivity is created, it creates a ServiceConnection object and calls the displayDistance() method.**

2. **MainActivity calls bindService() in its onStart() method.**
The `bindService()` method includes an intent meant for `OdometerService`, and a reference to the `ServiceConnection`.

3. **Android creates an instance of the OdometerService, and passes it the intent by calling its onBind() method.**

The story continues

OdometerService
MainActivity
Location Services

4. **OdometerService's onBind() method returns a Binder.**
The `Binder` is passed to `MainActivity`'s `ServiceConnection`.

5. **The ServiceConnection uses the Binder to give MainActivity a reference to OdometerService.**

6. **MainActivity's displayDistance() method calls OdometerService's getDistance() method every second.**
`OdometerService` returns a random number to `MainActivity`, in this case 0.56.

The story continues

7. **When MainActivity stops, it disconnects from OdometerService by calling unbindService().**

8. **OdometerService is destroyed when MainActivity is no longer bound to it.**

Now that you understand what happens when the code runs, let's take the app for a test drive.

Test drive the app

When we run the app, a random number is displayed in `MainActivity`. This number changes every second.

This is a random number generated by OdometerService.

We now have a working service that `MainActivity` can bind to. We still need to change the service so that the `getDistance()` method returns the distance traveled instead of a random number. Before we do that, however, we're going to take a closer look at how bound services work behind the scenes.

there are no Dumb Questions

Q: What's the difference between a started service and a bound service again?

A: A started service is created when an activity (or some other component) calls `startService()`. It runs code in the background, and when it finishes running, the service is destroyed.

A bound service is created when the activity calls `bindService()`. The activity can interact with the service by calling its methods. The service is destroyed when no components are bound to it.

Q: Can a service be both started and bound?

A: Yes. In such cases, the service is created when `startService()` or `bindService()` is called. It's only destroyed when the code it was asked to run in the background has stopped running, and there are no components bound to it.

Creating this kind of started-and-bound service is more complicated than creating a service that's only started or bound. You can find out how to do it in the Android documentation: *https://developer.android.com/guide/components/services.html*.

Q: What's the difference between a `Binder` and an `IBinder`?

A: An `IBinder` is an *interface*. A `Binder` is a class that *implements* the `IBinder` interface.

Q: Can other apps use a service I create?

A: Yes, but only if you set its `exported` attribute to `true` in *AndroidManifest.xml*.

The states of a bound service

When an application component (such as an activity) binds to a service, the service moves between three states: being created, being bound, and being destroyed. A bound service spends most of its time in a bound state.

Just like a started service, when a bound service is created, its `onCreate()` method gets called. As before, you override this method if you want to perform any tasks needed to set up the service.

The `onBind()` method runs when a component binds to the service. You override this method to return an `IBinder` object to the component, which it uses to get a reference to the service.

When all components have unbound from the service, its `onUnbind()` method is called.

Finally, the `onDestroy()` method is called when no components are bound to the service and it's about to be destroyed. As before, you override this method to perform any final cleanup tasks and free up resources.

We'll take a closer look at how these methods fit into the service states on the next page.

A bound service is destroyed when no components are bound to it.

The bound service lifecycle: from create to destroy

Here's a more detailed overview of the bound service lifecycle from birth to death.

1. **The component calls bindService() and the service gets created.**

2. **The onCreate() method runs immediately after the service is created.**
   The `onCreate()` method is where any service initialization code should go, as this method always gets called after the service has launched but before any components have bound to it.

3. **The onBind() method runs when the component binds to the service.**
   You override this method to return an `IBinder` object, which the component can use to get a reference to the service and call its methods.

4. **The service is bound for most of its life.**

5. **The onUnbind() method runs when all components have unbound from the service.**

6. **The onDestroy() method is called when no components are bound to the service and it's about to be destroyed.**
   You override this method if you want to perform any final cleanup tasks such as freeing up resources.

7. **After the onDestroy() method has run, the service is destroyed.**
   The service ceases to exist.

Now that you have a better understanding of how bound services work, let's change our Odometer app so that it displays the actual distance traveled by the user.

We'll use Android's Location Services to return the distance traveled

We need to get our `OdometerService` to return the distance traveled in its `getDistance()` method. To do this, we'll use Android's Location Services. These allow you to get the user's current location, request periodic updates, and ask for an intent to be fired when the user comes within a certain radius of a particular location.

In our case, we're going to use the Location Services to get periodic updates on the user's current location. We'll use these to calculate the distance the user has traveled.

To do this, we'll perform the following steps:

1. **Declare we need permission to use the Location Services.**
   Our app can only use the Location Services if the user grants our app permission to do so.

2. **Set up a location listener when the service is created.**
   This will be used to listen for updates from the Location Services.

3. **Request location updates.**
   We'll create a location manager, and use it to request updates on the user's current location.

4. **Calculate the distance traveled.**
   We'll keep a running total of the distance traveled by the user, and return this distance in the `OdometerService`'s `getDistance()` method.

5. **Remove location updates just before the service is destroyed.**
   This will free up system resources.

Before we start, we'll add the AppCompat Support Library to our project, as we'll need to use it in our code.

Add the AppCompat Support Library

OdometerService
MainActivity
Location Services

To get our location code working properly, there are a couple of classes we need to use from the AppCompat Support Library, so we'll add it to our project as a dependency. You do this in the same way that you did in earlier chapters. Choose File→Project Structure, then click on the app module and choose Dependencies. You'll be presented with the following screen:

Android Studio may have already added the AppCompat Support Library automatically. If so, you will see it listed as appcompat-v7, as shown above.

If the AppCompat Library isn't listed, you will need to add it yourself. To do this, click on the "+" button at the bottom or right side of the screen, choose the Library Dependency option, select the appcompat-v7 library, then click on the OK button. Click on OK again to save your changes and close the Project Structure window.

Next, we'll look at how to declare we need permission to use Android's Location Services.

Declare the permissions you need

Android allows you to perform many actions by default, but there are some that the user needs to give permission for in order for them to work. This can be because they use the user's private information, or could affect stored data or the way in which other apps function. Location Services is one of those things that the user needs to grant your app permission to use.

You declare the permissions your app requires in *AndroidManifest.xml* using the `<uses-permission>` element, which you add to the root `<manifest>` element. In our case, we need to access the user's precise location in order to display the distance traveled, so we need to declare the `ACCESS_FINE_LOCATION` permission. To do this, you add the following declaration to *AndroidManifest.xml* (update your version of the file to reflect this change):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.odometer">

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        ...
    </application>
</manifest>
```

Declare we need a permission.

We need to know the user's precise location.

How the app uses the above declaration depends on your app's target SDK (usually the most recent version of Android) and the API level of the user's device:

- If your target SDK is API level 23 or above, *and* the user's device is running 23 or above, **the app requests permission at runtime**. The user can deny or revoke permission, so whenever your code wants to use the thing that requires permission, it needs to check that permission is still granted. You'll find out how to do this later in the chapter.

- If your target SDK is API level 22 or below, *or* the user's device is running 22 or below, **the app requests permission when it's installed**. If the user denies permission, the app isn't installed. Once granted, permission can't be revoked except by uninstalling the app.

Now that we've declared that our app needs to know the user's location, let's get to work on `OdometerService`.

Add a location listener to OdometerService

OdometerService
MainActivity
Location Services

You create a location listener by implementing the **`LocationListener`** interface. It has four methods that you need to define: `onLocationChanged()`, `onProviderEnabled()`, `onProviderDisabled()`, and `onStatusChanged()`. `onLocationChanged()` gets called when the user's location has changed. We'll use this method later in the chapter to track the distance the user has traveled. The `onProviderEnabled()`, `onProviderDisabled()`, and `onStatusChanged()` methods are called when the location provider is enabled, when it is disabled, or when its status has changed, respectively.

We'll look at location providers on the next page.

We need to set up a location listener when `OdometerService` is first created, so we'll implement the interface in `OdometerService`'s `onCreate()` method. Update your version of *OdometerService.java* to include our changes below:

```
...
import android.os.Bundle;
import android.location.LocationListener;
import android.location.Location;

public class OdometerService extends Service {
    ...
    private LocationListener listener;

    @Override
    public void onCreate() {
        super.onCreate();
        listener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                //Code to keep track of the distance
            }

            @Override
            public void onProviderDisabled(String arg0) {}

            @Override
            public void onProviderEnabled(String arg0) {}

            @Override
            public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
        };
    }
    ...
}
```

We're using these extra classes, so we need to import them.

We're using a private variable for the LocationListener so other methods can access it.

Set up the LocationListener.

The Location parameter describes the current location. We'll use this later.

We'll complete this code later in the chapter.

We won't use any of these methods in our OdometerService code, but we still need to declare them.

We need a location manager and location provider

To get location updates, we need to do three things: create a location manager to get access to Android's Location Services, specify a location provider, and request that the location provider sends regular updates on the user's current location to the location listener we added on the previous page. We'll start by getting a location manager.

Create a location manager

You create a location manager in a similar way to how you created a notification manager in Chapter 18: using the `getSystemService()` method. Here's the code to create a location manager that you can use to access Android's Location Services (we'll add the code to `OdometerService`'s `onCreate()` method later on):

This is how you access the Android location service.

```
LocationManager locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

We used the getSystemService() method in Chapter 18 to get access to Android's notification service.

Specify the location provider

Next, we need to specify the location provider, which is used to determine the user's location. There are two main options: GPS or network. The GPS option uses the device's GPS sensor to establish the user's location, whereas the network option looks at Wi-Fi, Bluetooth, or mobile networks.

Not all devices have both types of location provider, so you can use the location manager's `getBestProvider()` method to get the most accurate location provider on the device. This method takes two parameters: a `Criteria` object you can use to specify criteria such as power requirements, and a flag to indicate whether it should currently be enabled on the device.

We want to use the location provider on the device with the greatest accuracy, so we'll use the following (we'll add it to `OdometerService` later):

```
String provider = locManager.getBestProvider(new Criteria(), true);
```

This gets the most accurate location provider that's available on the device.

Next we'll get the location provider to send location updates to the location listener.

OdometerService
MainActivity
Location Services

Request location updates...

You get the location provider to send updates to the location listener using the location manager's `requestLocationUpdates()` method. It takes takes four parameters: the location provider, the minimum time interval between updates in milliseconds, the minimum distance between location updates in meters, and the location listener you want to receive the updates. As an example, here's how you'd request location updates from the location provider every second when the device has moved more than a meter:

The location provider

The distance in meters

```
locManager.requestLocationUpdates(provider, 1000, 1, listener);
```

The LocationListener you want to receive updates

The time in milliseconds

...but first check that your app has permission

You can check your app's target SDK version by choosing File -> Project Structure, clicking on the app option, and then choosing Flavors.

If your app's target SDK is API level 23 or above, you need to check at runtime whether the user has granted you permission to get their current location. (As we said earlier in the chapter, if your target SDK is API level 23 or above, and the user's device is running one of these versions, the user may have installed the app without granting Location Services permission. You therefore have to check whether permission's been granted before running any code that requires Location Services, or your code won't compile.)

You check whether permission's been granted using the `ContextCompat.checkSelfPermission()` method. `ContextCompat` is a class from the AppCompat Support Library that provides backward compatibility with older versions of Android. Its `checkSelfPermission()` method takes two parameters: the current `Context` (usually `this`) and the permission you want to check. It returns a value of `PackageManager.PERMISSION_GRANTED` if permission has been granted.

In our case, we want to check whether the app's been granted `ACCESS_FINE_LOCATION` permission. Here's the code to do that:

Check if the ACCESS_FINE_LOCATION permission has been granted...

```
if (ContextCompat.checkSelfPermission(this,
                    android.Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
    locManager.requestLocationUpdates(provider, 1000, 1, listener);
}
```

...before requesting location updates.

On the next page we'll show you all the code you need to add to *OdometerService.java* to request location updates.

Here's the updated OdometerService code

Here's the full code to request location updates (update your version of *OdometerService.java* to include our changes):

```
...
import android.content.Context;
import android.location.LocationManager;
import android.location.Criteria;
import android.support.v4.content.ContextCompat;
import android.content.pm.PackageManager;

public class OdometerService extends Service {
    ...
    private LocationManager locManager;
    public static final String PERMISSION_STRING
            = android.Manifest.permission.ACCESS_FINE_LOCATION;
    ...

    @Override
    public void onCreate() {
        super.onCreate();
        listener = new LocationListener() {
            ...
        };
        locManager = (LocationManager) getSystemService (Context.LOCATION_SERVICE);
        if (ContextCompat.checkSelfPermission(this, PERMISSION_STRING)
                        == PackageManager.PERMISSION_GRANTED) {
            String provider = locManager.getBestProvider(new Criteria(), true);
            if (provider != null) {
                locManager.requestLocationUpdates(provider, 1000, 1, listener);
            }
        }
    }
    ...
}
```

We're using these extra classes, so import them.

We're using a private variable for the LocationManager so we can access it from other methods.

We're adding the permission String as a constant.

Get the LocationManager.

Check whether we have permission.

Get the most accurate location provider.

Request updates from the location provider.

Next we'll get our location listener to deal with the location updates.

Calculate the distance traveled

OdometerService
MainActivity
Location Services

So far, we've requested that the location listener be notified when the user's current location changes. When this happens, the listener's `onLocationChanged()` method gets called.

This method has one parameter, a `Location` object representing the user's current location. We can use this object to calculate the distance traveled by keeping a running total of the distance between the user's current location and their last.

You find the distance in meters between two locations using the `Location` object's `distanceTo()` method. As an example, here's how you'd find the distance between two locations called `location` and `lastLocation`:

```
double distanceInMeters = location.distanceTo(lastLocation);
```

This gets the distance between location and lastLocation.

Here's the code we need to add to `OdometerService` to record the distance traveled by the user (update your version of *OdometerService.java* to match ours):

```
public class OdometerService extends Service {
    private static double distanceInMeters;
    private static Location lastLocation = null;
    ...

    @Override
    public void onCreate() {
        super.onCreate();
        listener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                if (lastLocation == null) {
                    lastLocation = location;
                }
                distanceInMeters += location.distanceTo(lastLocation);
                lastLocation = location;
            }
        ...
        }
        ...
    }
    ...
}
```

We're using static variables to store the distance traveled and the user's last location so that their values are retained when the service is destroyed.

Set the user's starting location.

Update the distance traveled and the user's last location.

We'll use this code to return the distance traveled to `MainActivity`.

Return the miles traveled

To tell `MainActivity` how far the user has traveled, we need to update `OdometerService`'s `getDistance()` method. It currently returns a random number, so we'll change it to convert the value of the `distanceInMeters` variable to miles and return that value. Here's the new version of the `getDistance()` method; update your version of it to match ours:

```
public double getDistance() {
    return random.nextDouble();
    return this.distanceInMeters / 1609.344;
}
```

Delete this line.

This converts the distance traveled in meters into miles. We could make this calculation more precise if we wanted to, but it's accurate enough for our purposes.

Finally, we'll stop the listener getting location updates when the service is about to be destroyed.

Stop the listener getting location updates

We're going to stop the location listener getting updates using the `OdometerService`'s `onDestroy()` method, as this gets called just before the service is destroyed.

You stop the updates by calling the location manager's `removeUpdates()` method. It takes one parameter, the listener you wish to stop receiving the updates:

```
locManager.removeUpdates(listener);
```

This stops the location listener getting updates.

If your app's target SDK is API level 23 or above, you need to check whether the user has granted `ACCESS_FINE_LOCATION` permission before calling the `removeUpdates()` method. This is because you can only use this method if the user's granted this permission, and Android Studio will complain if you don't check first. You check whether permission's been granted in the same way we did earlier, by checking the return value of the `ContextCompat.checkSelfPermission()` method:

We can only remove the updates if we have permission.

```
if (ContextCompat.checkSelfPermission(this, PERMISSION_STRING)
        == PackageManager.PERMISSION_GRANTED) {
    locManager.removeUpdates(listener);
}
```

Over the next few pages we'll show you the full code for `OdometerService`, including the new `onDestroy()` method.

The full OdometerService.java code

OdometerService
MainActivity
Location Services

We've now done everything we need to get `OdometerService` to return the distance traveled. Update your version of *OdometerService.java* to match ours.

```
package com.hfad.odometer;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.IBinder;
import android.os.Binder;
import java.util.Random;
import android.location.LocationListener;
import android.location.Location;
import android.location.LocationManager;
import android.location.Criteria;
import android.support.v4.content.ContextCompat;
import android.content.pm.PackageManager;


public class OdometerService extends Service {

    private final IBinder binder = new OdometerBinder();
    private final Random random = new Random();
    private LocationListener listener;
    private LocationManager locManager;
    private static double distanceInMeters;
    private static Location lastLocation = null;
    public static final String PERMISSION_STRING
            = android.Manifest.permission.ACCESS_FINE_LOCATION;

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }
```

We're no longer returning a random number, so you can delete this import.

Delete the Random() object, as we're no longer using it.

The code continues on the next page.

The OdometerService.java code (continued)

None of the code on this page has changed.

```
@Override
public void onCreate() {
    super.onCreate();
    listener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            if (lastLocation == null) {
                lastLocation = location;
            }
            distanceInMeters += location.distanceTo(lastLocation);
            lastLocation = location;
        }

        @Override
        public void onProviderDisabled(String arg0) {
        }

        @Override
        public void onProviderEnabled(String arg0) {
        }

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle bundle) {
        }
    };

    locManager = (LocationManager) getSystemService (Context.LOCATION_SERVICE);
    if (ContextCompat.checkSelfPermission(this, PERMISSION_STRING)
                    == PackageManager.PERMISSION_GRANTED) {
        String provider = locManager.getBestProvider(new Criteria(), true);
        if (provider != null) {
            locManager.requestLocationUpdates(provider, 1000, 1, listener);
        }
    }
}
```

The code continues on the next page.

The OdometerService.java code (continued)

OdometerService
MainActivity
Location Services

```
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (locManager != null && listener != null) {
            if (ContextCompat.checkSelfPermission(this, PERMISSION_STRING)
                    == PackageManager.PERMISSION_GRANTED) {
                locManager.removeUpdates(listener);
            }
            locManager = null;
            listener = null;
        }
    }

    public double getDistance() {
        return this.distanceInMeters / 1609.344;
    }
}
```

Add the onDestroy() method.

Stop getting locations updates (if we have permission to remove them).

Set the LocationManager and LocationListener variables to null.

Let's take the app for a test drive.

there are no Dumb Questions

Q: I noticed I can call `checkSelfPermission()` directly in my service without using `ContextCompat`. Why do I have to use the `ContextCompat` version?

A: Because it's simpler to use. A `checkSelfPermission()` method was added to the `Context` class in API level 23, but this means it's not available on devices running an older version of Android.

- OdometerService
- MainActivity
- **Location Services**

Test drive the app

When we launch the app, 0.0 miles is initially displayed. When we check the app permissions, permission hasn't been granted to use Location Services. You can check this on your device by opening the device settings, choosing Apps, selecting the Odometer app, and opening the Permissions section:

Location Services permission for the app may be granted on your device by default. If this is the case, see what happens when you switch it off.

The app hasn't been granted permission to use Location Services.

When we grant Location permission for the Odometer app and return to the app, the location icon is displayed in the status bar, and the number of miles traveled increases when we take the device for a walk. The location icon disappears when we leave the app:

The location icon indicates our device GPS is being used.

The icon disappears when we leave the app.

The odometer works when we grant the app Location permission, but doesn't when permission's been denied.

Get the app to request permission

So far, we've made `OdometerService` check whether it has permission to get the user's precise location. If permission's been granted, it uses the Android Location Services to track the distance the user travels. But what if permission *hasn't* been granted?

If the app doesn't have permission to get the user's precise location, `OdometerService` can't use the Location Services we require. Rather than just accepting this, the app would work better if it were to ask the user to grant the permission.

We're going to change `MainActivity` so that if the user hasn't granted the permission we need, we'll request it. To achieve this, we'll get `MainActivity` to do three things:

1. **Before MainActivity binds to the service, request ACCESS_FINE_LOCATION permission if it's not already been granted.**
   This will present the user with a permission request dialog.

This dialog appears when you request ACCESS_FINE_LOCATION permission at runtime.

2. **Check the response, and bind to the service if permission is granted.**

3. **If permission is denied, issue a notification.**

We'll issue this notification if the user doesn't grant the permission we need.

Let's start by looking at how you make an activity request permissions at runtime.

Check permissions at runtime

Earlier in the chapter, you saw how to check whether the user has granted a particular permission using the `ContextCompat.checkSelfPermission()` method:

This code checks whether the user has granted a permission.

```
if (ContextCompat.checkSelfPermission(this, PERMISSION_STRING)
        == PackageManager.PERMISSION_GRANTED) {
    //Run code that needs the user's permission
}
```

If the user has granted permission, the method returns a value of `PackageManager.PERMISSION_GRANTED`, and the code requiring the permission will run successfully. But what if permission has been denied?

Ask for permissions you don't have

If the user hasn't granted one or more permissions needed by your code, you can use the **`ActivityCompat.requestPermissions()`** method to request permission at runtime. `ActivityCompat` is a class from the AppCompat Support Library that provides backward compatibility with older versions of Android. Its `requestPermissions()` method takes three parameters: a `Context` (usually `this`), a String array of permissions you want to check, and an int request code for the permission request. As an example, here's how you'd use the method to request the `ACCESS_FINE_LOCATION` permission:

The requestPermissions() method can only be called by an activity. It can't be called from a service.

Use this method to request permissions at runtime.

This is the request code for the permissions request. It can be any int. You'll see where this gets used in a couple of pages.

```
ActivityCompat.requestPermissions(this,
    new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION}, 6854);
```

When the `requestPermissions()` method is called, it displays one or more dialogs asking the user for each permission. The dialog gives the user a choice between denying or allowing the permission, and there's also a checkbox they can check if they don't want to be asked about the permission again. If they check the checkbox and deny the permission, subsequent calls to the `requestPermissions()` method won't display the dialog.

Note that the `requestPermissions()` method can only be called from an activity. You can't request permissions from a service.

We're going to update `MainActivity` so that it requests permission to get the device's location if it hasn't already been granted.

This is the permissions request dialog.

The user can deny or allow permissions using these options.

Check for Location Services permissions in MainActivity's onStart() method

We're currently using `MainActivity`'s `onStart()` method to bind the activity to `OdometerService`. We're going to change the code so that `MainActivity` only binds to the service if the user has granted the permission specified by the `PERMISSION_STRING` constant we defined in `OdometerService`. If permission's not been granted, we'll ask for it.

Here's the updated code for *MainActivity.java*; update your version of the code with our changes:

```
...
import android.content.pm.PackageManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;

public class MainActivity extends Activity {
    ...
    private final int PERMISSION_REQUEST_CODE = 698;

    ...
    @Override
    protected void onStart() {
        super.onStart();
        if (ContextCompat.checkSelfPermission(this, OdometerService.PERMISSION_STRING)
                != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
                    new String[]{OdometerService.PERMISSION_STRING},
                    PERMISSION_REQUEST_CODE);
        } else {
            Intent intent = new Intent(this, OdometerService.class);
            bindService(intent, connection, Context.BIND_AUTO_CREATE);
        }
    }
}
```

We're using these extra classes, so we need to import them.

We'll use this int for the permission request code.

If permission hasn't already been granted...

...request it at runtime.

If permission has already been granted, bind to the service.

Once you've requested permission from the user, you need to check the user's response. We'll do that next.

Check the user's response to the permission request

When you ask the user to grant a permission using the `requestPermissions()` method, you can't determine whether permission was granted by checking its return value. That's because the permission request happens asynchronously so that the current thread isn't blocked while you wait for the user to respond.

Instead, you check the user's response by overriding the activity's **`onRequestPermissionsResult()`** method. This has three parameters: an int request code to identify the permissions request, a String array of permissions, and an int array for the results of the requests.

To use this method, you first check whether the int request code matches the one you used in the `requestPermissions()` method. If it does, you then check whether the permission has been granted.

The code below checks whether the user granted the permission we requested using the `requestPermissions()` method on the previous page. If permission's been granted, it binds to `OdometerService`. Add this method to your version of *MainActivity.java*:

The onRequestPermissionsResult() method returns the results of your permissions requests.

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case PERMISSION_REQUEST_CODE: {
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Intent intent = new Intent(this, OdometerService.class);
                bindService(intent, connection, Context.BIND_AUTO_CREATE);
            } else {
                //Code to run if permission was denied
            }
        }
    }
}
```

Check whether the code matches the one we used in our requestPermissions() method.

If the request was cancelled, no results will be returned.

If permission was granted, bind to the service.

We still need to write this code.

Finally, if the user doesn't give us permission to use their current location, we need to inform them that the odometer won't work.

Issue a notification if we're denied permission

If the user decides not to grant permission to use their current location, the `OdometerService` won't be able to tell how far they've traveled. If this happens, we'll issue a notification to inform the user. We're going to use a notification because it will remain in the notification area until the user has decided what to do. Another advantage of using a notification is that we can get it to start `MainActivity` when it's clicked. Doing this means that `MainActivity`'s `onStart()` method will run, which will again prompt the user to grant the permission (unless the user has previously selected the option not to be asked again).

See if you can build the notification we require by having a go at the exercise on the next page.

We'll issue this notification if the user denies permission.

there are no Dumb Questions

Q: I tried turning off Location permissions for the Odometer app when I was in the middle of using it, and the number of miles displayed went back to 0. Why?

A: When you switch off the Location permissions for the app, Android may kill the process the app is running in. This resets all of the variables.

Q: That sounds drastic. Are there any other times when Android might kill a process?

A: Yes, when it's low on memory, but it will always try to keep alive any processes that are actively being used.

Q: Why aren't we calling the `requestPermissions()` method from `OdometerService`?

A: Because the `requestPermissions()` method is only available for activities, not services.

Q: Can I change the text that's displayed in the `requestPermissions()` dialog?

A: No. The text and options it displays are fixed, so Android won't let you change them.

Q: But I want to give the user more information about why I need a particular permission. Can I do that?

A: One option is to call the `ActivityCompat.shouldShowRequestPermissionRationale()` method before calling `requestPermissions()`. This returns a `true` value if the user has previously denied the permission request, and hasn't checked the checkbox to say they don't want to be asked again. If this is the case, you can give the user more information outside the permission request before requesting the permission again.

Q: What other permissions do I have to declare and ask permission for?

A: Generally, you need the user's permission for any actions that use private data or may affect the way in which other apps function. The online documentation for each class should indicate whether a permission is needed, and Android Studio should highlight this too. You can find a full list of actions requiring permissions here:

https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous

Q: What if I design an app that performs these kinds of actions and don't ask for permission?

A: If your target SDK is API level 23 or above and you don't request permission, your code won't compile.

Pool Puzzle

Your **goal** is to build and issue a heads-up notification. The notification should start `MainActivity` when it's clicked, then disappear. Take code snippets from the pool, and place them in the blank lines in the code. You may **not** use the same snippet more than once, and you won't need to use all the snippets.

Write the code to create this notification.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
                .setSmallIcon(android.R.drawable.ic_menu_compass)
                .setContentTitle("Odometer")
                .setContentText("Location permission required")
                .setPriority(NotificationCompat.                    )
                .setVibrate(new long[] {0, 1000})
                .                           (true);

Intent actionIntent = new Intent(this, MainActivity.class);
PendingIntent actionPendingIntent = PendingIntent.              (this, 0,
                        actionIntent, PendingIntent.FLAG_UPDATE_CURRENT);
builder.setContentIntent(                          );

NotificationManager notificationManager =
                  (NotificationManager) getSystemService(                                  );
notificationManager.notify(43,                    );
```

A built-in drawable to display a compass icon.

Note: each thing from the pool can only be used once!

Pool Puzzle Solution

Your **goal** is to build and issue a heads-up notification. The notification should start `MainActivity` when it's clicked, then disappear. Take code snippets from the pool, and place them in the blank lines in the code. You may **not** use the same snippet more than once, and you won't need to use all the snippets.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
                .setSmallIcon(android.R.drawable.ic_menu_compass)
                .setContentTitle("Odometer")
                .setContentText("Location permission required")
                .setPriority(NotificationCompat.PRIORITY_HIGH)
                .setVibrate(new long[] {0, 1000})
                .setAutoCancel(true);

Intent actionIntent = new Intent(this, MainActivity.class);
PendingIntent actionPendingIntent = PendingIntent.getActivity(this, 0,
                        actionIntent, PendingIntent.FLAG_UPDATE_CURRENT);
builder.setContentIntent(actionPendingIntent);

NotificationManager notificationManager =
                (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
notificationManager.notify(43, builder.build());
```

Heads-up notifications need to have a high priority.

This makes the notification disappear when it's clicked.

Create a PendingIntent using the getActivity() method.

Add the PendingIntent to the notification so that it starts MainActivity when it's clicked.

Use the notification service.

Build the notification.

You didn't need to use these snippets.

Add notification code to onRequestPermissionsResults()

We'll update our `MainActivity` code so that if the user denies our permission request, it issues a heads-up notification.

First, add the following Strings to *Strings.xml*; we'll use these for the notification's title and text:

```
<string name="app_name">Odometer</string>
<string name="permission_denied">Location permission required</string>
```

Android Studio may have already added this String.

Then update your version of *MainActivity.java* with the following code:

```
...
import android.support.v4.app.NotificationCompat;
import android.app.NotificationManager;
import android.app.PendingIntent;

public class MainActivity extends Activity {
    ...
    private final int NOTIFICATION_ID = 423;

    ...
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[], int[] grantResults) {
        switch (requestCode) {
            case PERMISSION_REQUEST_CODE: {
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    ...
                } else {
                    //Create a notification builder
                    NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
                        .setSmallIcon(android.R.drawable.ic_menu_compass)
                        .setContentTitle(getResources().getString(R.string.app_name))
                        .setContentText(getResources().getString(R.string.permission_denied))
                        .setPriority(NotificationCompat.PRIORITY_HIGH)
                        .setVibrate(new long[] {1000, 1000})
                        .setAutoCancel(true);
```

We're using these extra classes, so we need to import them.

We'll use this constant for the notification ID.

These settings are needed for all notifications.

Add these to make it a heads-up notification.

This line makes the notification disappear when it's clicked.

The code continues on the next page.

The notification code (continued)

```
                //Create an action
                Intent actionIntent = new Intent(this, MainActivity.class);
                PendingIntent actionPendingIntent = PendingIntent.getActivity(
                        this,
                        0,
                        actionIntent,
                        PendingIntent.FLAG_UPDATE_CURRENT);
                builder.setContentIntent(actionPendingIntent);

                //Issue the notification
                NotificationManager notificationManager =
                        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
                notificationManager.notify(NOTIFICATION_ID, builder.build());
            }
        }
    }
}

...

}
```

Adding a PendingIntent to the notification means that it will start MainActivity when it's clicked.

Build the notification and issue it.

That's all the code we need to display a notification if the user decides to deny us `ACCESS_FINE_LOCATION` permission. We'll show you the full `MainActivity` code over the next few pages, then take the app for a final test drive.

The full code for MainActivity.java

Here's our complete code for *MainActivity.java*; make sure your version of the code matches ours:

```
package com.hfad.odometer;

import android.app.Activity;
import android.os.Bundle;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.widget.TextView;
import java.util.Locale;
import android.content.pm.PackageManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v4.app.NotificationCompat;
import android.app.NotificationManager;
import android.app.PendingIntent;

public class MainActivity extends Activity {

    private OdometerService odometer;
    private boolean bound = false;
    private final int PERMISSION_REQUEST_CODE = 698;
    private final int NOTIFICATION_ID = 423;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            OdometerService.OdometerBinder odometerBinder =
                    (OdometerService.OdometerBinder) binder;
            odometer = odometerBinder.getOdometer();
            bound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            bound = false;
        }
    };
```

These are all classes from the AppCompat Support Library.

We've been using the Activity class, but you can use AppCompatActivity if you prefer.

We need a ServiceConnection in order to bind MainActivity to OdometerService.

The code continues on the next page.

The MainActivity.java code (continued)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    displayDistance();
}

@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case PERMISSION_REQUEST_CODE: {
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Intent intent = new Intent(this, OdometerService.class);
                bindService(intent, connection, Context.BIND_AUTO_CREATE);
            } else {
                //Create a notification builder
                NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
                        .setSmallIcon(android.R.drawable.ic_menu_compass)
                        .setContentTitle(getResources().getString(R.string.app_name))
                        .setContentText(getResources().getString(R.string.permission_denied))
                        .setPriority(NotificationCompat.PRIORITY_HIGH)
                        .setVibrate(new long[] { 1000, 1000})
                        .setAutoCancel(true);
                //Create an action
                Intent actionIntent = new Intent(this, MainActivity.class);
                PendingIntent actionPendingIntent = PendingIntent.getActivity(this, 0,
                        actionIntent, PendingIntent.FLAG_UPDATE_CURRENT);
                builder.setContentIntent(actionPendingIntent);
                //Issue the notification
                NotificationManager notificationManager =
                        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
                notificationManager.notify(NOTIFICATION_ID, builder.build());
            }
        }
    }
}
```

If we've asked the user for permissions at runtime, check the result.

Bind to the service if the user has granted permission.

Issue a notification if permission's been denied.

The code continues on the next page.

Request
Granted
Denied

The MainActivity.java code (continued)

```
    @Override
    protected void onStart() {
        super.onStart();
        if (ContextCompat.checkSelfPermission(this,
                OdometerService.PERMISSION_STRING)
                != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
                    new String[]{OdometerService.PERMISSION_STRING},
                    PERMISSION_REQUEST_CODE);
        } else {
            Intent intent = new Intent(this, OdometerService.class);
            bindService(intent, connection, Context.BIND_AUTO_CREATE);
        }
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (bound) {
            unbindService(connection);
            bound = false;
        }
    }

    private void displayDistance() {
        final TextView distanceView = (TextView)findViewById(R.id.distance);
        final Handler handler = new Handler();
        handler.post(new Runnable() {
            @Override
            public void run() {
                double distance = 0.0;
                if (bound && odometer != null) {
                    distance = odometer.getDistance();
                }
                String distanceStr = String.format(Locale.getDefault(),
                                                   "%1$,.2f miles", distance);
                distanceView.setText(distanceStr);
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

Request ACCESS_FINE_LOCATION permission if we don't have it already.

Bind to OdometerService if we've been granted the permission it requires.

We're binding to OdometerService in two different places, so you could put this code in a separate method.

Unbind from OdometerService when MainActivity stops.

Display the distance traveled.

Test drive the app

Request
Granted
Denied

When we run the app with its Location permission switched off, a permission request dialog is displayed. If we click on the Deny option, a notification is issued:

When we click on the notification, the permission request dialog is displayed again. If we click on the option to allow permission, the Location icon appears in the status bar and the number of miles increases when we take the device on a road trip:

We know you're full of great ideas for improving the Odometer app, so why not try them out? As an example, try adding Start, Stop, and Reset buttons to start, stop, and reset the distance traveled.

Your Android Toolbox

You've got Chapter 19 under your belt and now you've added bound services to your toolbox.

You can download the full code for the chapter from https://tinyurl.com/HeadFirstAndroid.

BULLET POINTS

- You create a bound service by extending the **`Service`** class. You define your own **`Binder`** object, and override the **`onBind()`** method.
- Bind a component to a service using the **`bindService()`** method.
- Use a **`ServiceConnection`** so that your activity can get a reference to the service when it's bound.
- Unbind a component from a service using the **`unbindService()`** method.
- When a bound service is created, its `onCreate()` method is called. `onBind()` gets called when a component binds to the service.
- When all components have unbound from the service, its `onUnbind()` method is called.
- A bound service is destroyed when no components are bound to it. Its `onDestroy()` method is called just before the service is destroyed.
- Use the **Android Location Services** to determine the current location of the device.
- To get the current location of the device, you need to declare that the app requires **`ACCESS_FINE_LOCATION`** permission in *AndroidManifest.xml*.
- Get location updates using a **`LocationListener`**.
- A **`LocationManager`** gives you access to Android's Location Services. Get the best location provider available on the device using its **`getBestProvider()`** method. Request location updates from the provider using **`requestLocationUpdates()`**.
- Use **`removeUpdates()`** to stop getting location updates.
- If your target SDK is API level 23 or above, check at runtime whether your app has been granted a permission using the **`ContextCompat.checkSelfPermission()`** method.
- Request permissions at runtime using **`ActivityCompat.requestPermissions()`**.
- Check the user's response to a permission request by implementing the activity's **`onRequestPermissionsResult()`** method.

Leaving town...

It's been great having you here in Androidville

We're sad to see you leave, but there's nothing like taking what you've learned and putting it to use. There are still a few more gems for you in the back of the book and a handy index, and then it's time to take all these new ideas and put them into practice. Bon voyage!

appendix i: relative and grid layouts

Meet the Relatives

There are two more layouts you will often meet in Androidville.

In this book we've concentrated on using simple *linear and frame layouts*, and introduced you to *Android's new constraint layout*. But there are two more layouts we'd like you to know about: **relative layouts** and **grid layouts**. They've largely been superseded by the constraint layout, but we have a soft spot for them, and we think they'll stay around for a few more years.

A relative layout displays views in relative positions

A relative layout allows you to position views relative to their parent layout, or relative to other views in the layout.

You define a relative layout using the `<RelativeLayout>` element like this:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...>

    ...

</RelativeLayout>
```

The layout_width and layout_height specify what size you want the layout to be.

There may be other attributes too.

This is where you add any views.

Positioning views relative to the parent layout

If you want a view to always appear in a particular position on the screen, irrespective of the screen size or orientation, you need to position the view relative to its parent. As an example, here's how you'd make sure a button always appears in the top-center of the layout:

```
<RelativeLayout ... >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

The layout contains the button, so the layout is the button's parent.

The lines of code:

```
android:layout_alignParentTop="true"
android:layout_centerHorizontal="true"
```

mean that the top edge of the button is aligned to the top edge of the layout, and the button is centered horizontally in the parent layout. This will be the case no matter what the screen size, language, or orientation of your device:

Positioning views to the left or right

You can also position a view to the left or right of the parent layout. There are two ways of doing this.

The first way is to explicitly position the view on the left or right using:

```
android:layout_alignParentLeft="true"
```

or:

```
android:layout_alignParentRight="true"
```

These lines of code mean that the left (or right) edge of the view is aligned to the left (or right) edge of the parent layout, regardless of the screen size, orientation, or language being used on the device.

Use start and end to take language direction into account

For apps where the minimum SDK is *at least* API 17, you can position views on the left or right depending on the language setting on the device. As an example, you might want views to appear on the left for languages that are read from left to right such as English. For languages that are read from right to left, you might want them to appear on the right instead so that their position is mirrored.

To do this, you use:

```
android:layout_alignParentStart="true"
```

or:

```
android:layout_alignParentEnd="true"
```

`android:layout_alignParentStart="true"` aligns the start edge of the view with that of its parent. The start edge is on the left for languages that are read from left to right, and the right edge for languages that are read from right to left.

`android:layout_alignParentEnd="true"` aligns the end edge of the view with that of its parent. The end edge is on the right for languages that are read from left to right, and the left edge for languages that are read from right to left.

Attributes for positioning views relative to the parent layout

Here are some of the most common attributes for positioning views relative to their parent layout. Add the attribute you want to the view you're positioning, then set its value to `"true"`:

```
android:attribute="true"
```

Attribute	What it does
layout_alignParentBottom	Aligns the bottom edge of the view to the bottom edge of the parent.
layout_alignParentLeft	Aligns the left edge of the view to the left edge of the parent.
layout_alignParentRight	Aligns the right edge of the view to the right edge of the parent.
layout_alignParentTop	Aligns the top edge of the view to the top edge of the parent.
layout_alignParentStart	Aligns the start edge of the view to the start edge of the parent.
layout_alignParentEnd	Aligns the end edge of the view to the end edge of the parent.
layout_centerInParent	Centers the view horizontally and vertically in the parent.
layout_centerHorizontal	Centers the view horizontally in the parent.
layout_centerVertical	Centers the view vertically in the parent.

Positioning views relative to other views

In addition to positioning views relative to the parent layout, you can also position views *relative to other views*. You do this when you want views to stay aligned in some way, irrespective of the screen size or orientation.

In order to position a view relative to another view, the view you're using as an anchor must be given an ID using the `android:id` attribute:

```
android:id="@+id/button_click_me"
```

The syntax "`@+id`" tells Android to include the ID as a resource in its resource file *R.java*. You must include the "+" whenever you define a new view in the layout. If you don't, Android won't add the ID as a resource and you'll get errors in your code. You can omit the "+" when the ID has already been added as a resource.

Here's how you create a layout with two buttons, with one button centered in the middle of the layout, and the second button positioned underneath the first:

We're using this button as an anchor for the second one, so it needs an ID.

```
<RelativeLayout ... >
    <Button
        android:id="@+id/button_click_me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Click Me" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignStart="@id/button_click_me"
        android:layout_below="@id/button_click_me"
        android:text="New Button" />
</RelativeLayout>
```

We're putting a second button underneath the first so that the start edges of both buttons are aligned.

The lines:

```
android:layout_alignStart="@id/button_click_me"
android:layout_below="@id/button_click_me"
```

When you're referring to views that have already been defined in the layout, you can use @id instead of @+id.

ensure that the second button has its start edge aligned to the start edge of the first button, and is always positioned beneath it.

Attributes for positioning views relative to other views

Here are attributes you can use when positioning views relative to another view. Add the attribute to the view you're positioning, and set its value to the view you're positioning relative to:

```
android:attribute="@+id/view_id"
```

Remember, you can leave out the "+" if you've already defined the view ID in your layout.

Attribute	What it does
layout_above	Puts the view above the view you're anchoring it to.
layout_below	Puts the view below the view you're anchoring it to.
layout_alignTop	Aligns the top edge of the view to the top edge of the view you're anchoring it to.
layout_alignBottom	Aligns the bottom edge of the view to the bottom edge of the view you're anchoring it to.
layout_alignLeft, layout_alignStart	Aligns the left (or start) edge of the view to the left (or start) edge of the view you're anchoring it to.
layout_alignRight, layout_alignEnd	Aligns the right (or end) edge of the view to the right (or end) edge of the view you're anchoring it to.
layout_toLeftOf, layout_toStartOf	Puts the right (or end) edge of the view to the left (or start) of the view you're anchoring it to.
layout_toRightOf, layout_toEndOf	Puts the left (or start) edge of the view to the right (or end) of the view you're anchoring it to.

A grid layout displays views in a grid

A grid layout splits the screen up into a grid of rows and columns, and allocates views to cells:

How you define a grid layout

You define a grid layout in a similar way to how you define the other types of layout, this time using the **<GridLayout>** element:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    ... >

    ...

</GridLayout>
```

These are the same attributes we used for our other layouts.

How many columns you want your layout to have (in this case, 2)

This is where you add any views.

You specify how many columns you want the grid layout to have using:

```
android:columnCount="number"
```

where `number` is the number of columns. You can also specify a maximum number of rows using:

```
android:rowCount="number"
```

but in practice you can usually let Android figure this out based on the number of views in the layout. Android will include as many rows as is necessary to display the views.

Adding views to the grid layout

You add views to a grid layout in a similar way to how you add views to a linear layout:

```
<GridLayout ... >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/textview" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/click_me" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit" />

</GridLayout>
```

Just as with a linear layout, there's no need to give your views IDs unless you're explicitly going to refer to them in your activity code. The views don't need to refer to each other within the layout, so they don't need to have IDs for this purpose.

By default, the grid layout positions your views in the order in which they appear in the XML. So if you have a grid layout with two columns, the grid layout will put the first view in the first position, the second view in the second position, and so on.

The downside of this approach is that if you remove one of your views from the layout, it can drastically change the appearance of the layout. To get around this, you can specify where you want each view to appear, and how many columns you want it to span.

Let's create a new grid layout

To see this in action, we'll create a grid layout that specifies which cells we want views to appear in, and how many columns they should span. The layout is composed of a text view containing the text "To", an editable text field that contains hint text of "Enter email address", an editable text field that contains hint text of "Message", and a button labeled "Send":

Here's what we're going to do

1. **Sketch the user interface, and split it into rows and columns.** This will make it easier for us to see how we should construct our layout.

2. **Build up the layout row by row.**

We'll start with a sketch

The first thing we'll do to create our new layout is sketch it out. That way we can see how many rows and columns we need, where each view should be positioned, and how many columns each view should span.

The grid layout needs two columns

We can position our views how we want if we use a grid layout with two columns:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:columnCount="2"
    tools:context="com.hfad.gridlayout.MainActivity" >
</GridLayout>
```

Now that we have the basic grid layout defined, we can start adding views.

Row 0: add views to specific rows and columns

The first row of the grid layout is composed of a text view in the first column, and an editable text field in the second column. You start by adding the views to the layout:

```
<GridLayout...>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/to" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill_horizontal"
        android:hint="@string/to_hint" />
</GridLayout>
```

You can use layout_gravity in grid layouts too. We're using fill_horizontal because we want the editable text field to fill the remaining horizontal space.

You can use android:gravity and android:layout_gravity attributes with grid layouts.

Then you use the `android:layout_row` and `android:layout_column` attributes to say which row and column you want each view to appear in. The row and column indices start from 0, so if you want a view to appear in the first column and first row, you use:

```
android:layout_row="0"
android:layout_column="0"
```

Columns and rows start at 0, so this refers to the first row and first column.

Row and column indices start at 0. layout_column="n" refers to column n+1 in the display.

Let's apply this to our layout code by putting the text view in column 0, and the editable text field in column 1.

```
<GridLayout...>
    <TextView
        ...
        android:layout_row="0"
        android:layout_column="0"
        android:text="@string/to" />

    <EditText
        ...
        android:layout_row="0"
        android:layout_column="1"
        android:hint="@string/to_hint" />
</GridLayout>
```

Row 1: make a view span multiple columns

The second row of the grid layout is composed of an editable text field that starts in the first column and spans across the second. The view takes up all the available space.

To get a view to span multiple columns, you start by specifying which row and column you want the view to start in. We want the view to start in the first column of the second row, so we need to use:

```
android:layout_row="1"
android:layout_column="0"
```

We want our view to go across two columns, and we can do this using the `android:layout_columnSpan` attribute like this:

```
android:layout_columnSpan="number"
```

where `number` is the number of columns we want the view to span across. In our case, this is:

```
android:layout_columnSpan="2"
```

Putting it all together, here's the code for the message view:

```
<GridLayout...>
    <TextView... />
    <EditText.../>
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:gravity="top"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:hint="@string/message" />
</GridLayout>
```

These are the views we added on the last page for row 0.

We want the view to fill the available space, and for the text to appear at the top.

The view starts in column 0, and spans two columns.

Now that we've added the views for the first two rows, all we need to do is add the button.

Row 2: make a view span multiple columns

We need the button to be centered horizontally across the two columns like this:

Layout Magnets

We wrote some code to center the Send button in the third row of the grid layout, but a sudden breeze blew some of it away. See if you can reconstruct the code using the magnets below.

```
<GridLayout...>
    <TextView... />
    <EditText.../>
    <EditText.../>

   <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:layout_row= ...............

        android:layout_column= ...............

        android:layout_gravity= ..................................

        android:layout_columnSpan= ...............

        android:text="@string/send" />

</GridLayout>
```

These are the views we've already added.

You won't need to use all of these magnets.

- fill_horizontal
- "0"
- "2"
- "0"
- "1"
- "1"
- "2"
- center_horizontal

Layout Magnets Solution

We wrote some code to center the Send button in the third row of the grid layout, but a sudden breeze blew some of it away. See if you can reconstruct the code using the magnets below.

```
<GridLayout...>
    <TextView... />
    <EditText.../>
    <EditText.../>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:layout_row= "2"
        android:layout_column= "0"
        android:layout_gravity= center_horizontal
        android:layout_columnSpan= "2"
        android:text="@string/send" />

</GridLayout>
```

The button starts at row 2, column 0.

We want to center it horizontally.

It spans two columns.

fill_horizontal

"0"

You didn't need to use these magnets.

"1"

"1"

The full code for the grid layout

Here's the full code for the grid layout.

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:columnCount="2"
    tools:context="com.hfad.gridlayout.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="0"
        android:layout_column="0"
        android:text="@string/to" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill_horizontal"
        android:layout_row="0"
        android:layout_column="1"
        android:hint="@string/to_hint" />
```

The To text view.

The email address text field.

The code for the Message text field and the Send button is on the next page.

The grid layout code (continued)

```
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:gravity="top"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="2"
        android:layout_column="0"
        android:layout_gravity="center_horizontal"
        android:layout_columnSpan="2"
        android:text="@string/send" />
</GridLayout>
```

The Message text field

The button spans two columns, starting from row 2 column 0. It's centered horizontally.

appendix ii: gradle

The Gradle Build Tool

Most Android apps are created using a build tool called Gradle.

Gradle works behind the scenes to find and download libraries, compile and deploy your code, run tests, clean the grouting, and so on. Most of the time you *might not even realize it's there* because Android Studio provides a graphical interface to it. Sometimes, however, it's helpful to dive into Gradle and **hack it manually**. In this appendix we'll introduce you to some of Gradle's many talents.

What ~~have the Romans~~ has Gradle ever done for us?

When you click the run button in Android Studio, most of the actual work is done by an external build tool called **Gradle**. Here are some of the things that Gradle does:

- Locates and downloads the correct versions of any third-party libraries you need.
- Calls the correct build tools in the correct sequence to turn all of your source code and resources into a deployable app.
- Installs and runs your app on an Android device.
- A whole bunch of other stuff, like running tests and checking the quality of your code.

It's hard to list all of the things that Gradle does because it's designed to be easily extensible. Unlike other XML-based build tools like Maven or Ant, Gradle is written in a procedural language (Groovy), which is used for both configuring a build and adding extra functionality.

Your project's Gradle files

Every time you create a new project, Android Studio creates two files called *build.gradle*. One of these files is in your project folder, and contains a small amount of information that specifies the basic settings of your app, such as what version of Gradle to use, and which online repository:

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.3.0'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

You will normally only need to change the code in this file if you want to install a third-party plug-in, or need to specify another place that contains downloadable libraries.

Your app's main Gradle file

The second *build.gradle* file lives inside the *app* folder within your project. This file tells Gradle how to build all of the code in your main Android module. It's where the majority of your application's properties are set, such as the level of the API that you're targeting, and the specifics of which external libraries your app will need:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.1"
    defaultConfig {
        applicationId "com.hfad.example"
        minSdkVersion 19
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.1.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    testCompile 'junit:junit:4.12'
}
```

Gradle comes built in to your project

Every time you create a new application, Android Studio includes an install of the Gradle build tool. If you look in the project directory, you will see two files called *gradlew* and *gradlew.bat*. These files contains scripts that allow you to build and deploy your app from the command line.

To get more familiar with Gradle, open a command prompt or terminal on your development machine and change into the top-level directory of your project. Then run one of the *gradlew* scripts with the parameter `tasks`. Gradle will tell you some of the tasks that it can perform for you:

We've cut down the actual output, because there are many, many tasks that you can run by default.

```
File Edit Window Help EmacsFTW
$ ./gradlew tasks
Build tasks
-----------
assemble - Assembles all variants of all applications and
        secondary packages.
build - Assembles and tests this project.
clean - Deletes the build directory.
compileDebugSources
mockableAndroidJar - Creates a version of android.jar that's
        suitable for unit tests.

Install tasks
-------------
installDebug - Installs the Debug build.
uninstallAll - Uninstall all applications.
uninstallDebug - Uninstalls the Debug build.

Verification tasks
------------------
check - Runs all checks.
connectedAndroidTest - Installs and runs instrumentation tests
        for all flavors on connected devices.
lint - Runs lint on all variants.
test - Run unit tests for all variants.

To see all tasks and more detail, run gradlew tasks --all

BUILD SUCCESSFUL

Total time: 6.209 secs
$
```

Let's take a quick tour of some of the most useful ones.

The check task

The `check` task performs static analysis on the source code in your application. Think of it as your coding buddy, who at a moment's notice can scoot through your files looking for coding errors. By default, the `check` task uses the *lint* tool to look for common Android programming errors. It will generate a report in *app/build/reports/lint-results.html*:

The clean installDebug task

This task will do a complete compile and install of your application on your connected device. You can obviously do this from the IDE, but it can be useful to do this from the command line if, for example, you want to automatically build your application on an Integration Server.

The androidDependencies task

For this task, Gradle will automatically pull down any libraries your application requires, and some of those libraries will automatically pull down other libraries, which might pull down other libraries and… well, you get the idea.

Even though your *app/build.gradle* file might only mention a couple of libraries, your application might need to install many dependent libraries for your app. So it's sometimes useful to see which libraries your application requires, and why. That's what the `androidDependencies` task is for: it displays a tree of all of the libraries in your app:

```
File  Edit  Window  Help  EmacsFTW
$ ./gradlew androidDependencies

Incremental java compilation is an incubating feature.
:app:androidDependencies
debug
+--- com.android.support:appcompat-v7:25.1.1@aar
|    +--- com.android.support:support-annotations:25.1.1@jar
|    +--- com.android.support:support-v4:25.1.1@aar
|    |    +--- com.android.support:support-compat:25.1.1@aar
|    |    |    \--- com.android.support:support-annotations:25.1.1@jar
|    |    +--- com.android.support:support-media-compat:25.1.1@aar
|    |    |    +--- com.android.support:support-annotations:25.1.1@jar
|    |    |    \--- com.android.support:support-compat:25.1.1@aar
|    |    |         \--- com.android.support:support-annotations:25.1.1@jar
|    |    +--- com.android.support:support-core-utils:25.1.1@aar
|    |    |    +--- com.android.support:support-annotations:25.1.1@jar
|    |    |    \--- com.android.support:support-compat:25.1.1@aar
...
```

gradlew <your-task-goes-here>

The real reason that Android apps are commonly built with Gradle is because it's easily extended. All Gradle files are written in Groovy, which is a general-purpose language designed to be run by Java. That means you can easily add your own custom tasks.

As an example, add the following code to the end of your *app/build.gradle* file:

```
task javadoc(type: Javadoc) {
    source = android.sourceSets.main.java.srcDirs
    classpath += project.files(android.getBootClasspath().join(File.pathSeparator))
    destinationDir = file("$project.buildDir/javadoc/")
    failOnError false
}
```

This code creates a new task called `javadoc`, which generates javadoc for your source code. You can run the task with:

```
./gradlew javadoc
```

The generated files will be published in *app/build/javadoc*:

Gradle plug-ins

As well as writing your own tasks, you can also install Gradle plug-ins. A plug-in can greatly extend your build environment. Want to write Android code in Clojure? Want your build to automatically interact with source control systems like Git? How about spinning up entire servers in Docker and then testing your application on them?

You can do these things, and many, many more by using Gradle plug-ins. For details on what plug-ins are available, see *https://plugins.gradle.org*.

appendix iii: art

The Android Runtime

Ever wonder how Android apps can run on so many kinds of devices? Android apps run in a virtual machine called the **Android runtime (ART)**, not the Oracle Java Virtual Machine (JVM). This means that your apps are quicker to start on small, low-powered devices and run more efficiently. In this appendix, we'll look at how ART works.

What is the Android runtime?

The Android Runtime (ART) is the system that runs your compiled code on an Android device. It first appeared on Android with the release of KitKat and became the standard way of running code in Lollipop.

ART is designed to run your compiled Android apps quickly and efficiently on small, low-powered devices. Android Studio uses the Gradle build system to do all the work of creating and installing apps for you, but it can be useful to understand what happens behind the scenes when you click the Run button. Let's see what really goes on.

You don't need to understand the info in this appendix in order to create cool Android apps. So if you're not into the nitty-gritty of what's going on behind the scenes when an Android device runs an app, feel free to skip this appendix.

Previously, Java code ran on the Oracle JVM

Java has been around for a very long time, and compiled Java applications have almost always been run on the Oracle Java Virtual Machine (JVM). In that scenario, Java source code gets compiled down to *.class* files. One *.class* file gets created for every Java class, interface, or enum in your source code:

The *.class* files contain Java bytecodes, which can be read and executed by the JVM. The JVM is a software emulation of a Central Processing Unit (CPU), like the chip at the heart of your development machine. But because it's emulated, it can be made to work on almost any device. That's why Java code is designed to be written once, run anywhere.

So is that what happens on Android devices? Well…not quite. The Android Runtime performs the same kind of job as the JVM, but it does it in a very different way.

ART compiles code into DEX files

When you do Android development, your Java source code is compiled down into *.dex* files. A *.dex* file serves a similar purpose to a *.class* file, because it contains executable bytecodes. But instead of being JVM bytecodes, they are in a different format called **Dalvik**. DEX stands for **D**alvik **EX**ecutable.

Rather than creating a *.dex* file for each and every class file, your app will normally be compiled down into a single file called *classes.dex*. That single *.dex* file will contain bytecodes for all of your source code, and for every library in your app.

The DEX format can only cope with 65,535 methods, so if your app contains a lot of code or some large libraries, your file might need to be converted into multiple *.dex* files:

You can find out more about creating multi-DEX apps here:

https://developer.android.com/studio/build/multidex.html.

How DEX files get created

When Android builds your app, it uses a tool called dx, which stitches `.class` files into a DEX file:

It may seem a bit weird that the compilation process involves two stages of compilation: first to *.class* files, and then from *.class* files to the DEX format. Why doesn't Google just create a single tool that goes straight from `.java` source code to DEX bytecodes?

For a while Google was developing a compiler called JACK and an associated linker called JILL that could create DEX code straight from Java code, but there was a problem. Some Java tools don't just work at the source code level; they work directly with *.class* files, and modify the code that those files contain.

For example, if you use a code coverage tool to see which code your tests are actually running, the coverage tool will likely want to modify the contents of the generated *.class* files to add in additional bytecodes to keep track of the code as it's executed. If you used the JACK compiler, no *.class* files were generated.

So back in March 2017, Google announced that it was shelving JACK, and was putting all of its efforts into making the dx tool work really well with the latest Java *.class* formats. This has the additional advantage that any new Java language features—so long as they don't require new Java byte codes—will automatically be supported by Android.

DEX files are zipped into APK files

But Android apps aren't shipped around as *.dex* files. There's a whole bunch of other files that make up your app: images, sounds, metadata, and so on. All of these resources and your DEX bytecode is wrapped up into a single zip file called an Android Package or *.apk* file. The *.apk* file is created by another tool called the Android Asset Packing Tool, or aapt.

When you download an app from the Google Play Store, what actually gets downloaded is an APK file. In fact, when you run your app from Android Studio, the build system will first build an *.apk* file and then install it onto your Android device.

You may need to sign the .apk file

If you're going to distribute your app through the Google Play Store, you will need to sign it. Signing an app package means that you store an additional file in the *.apk* that is based on a checksum of the contents of the *.apk* and a separately generated private key. The *.apk* file uses the standard jarsigner tool that comes as part of Oracle's Java Development Kit. The jarsigner tool was created to sign *.jar* files, but it will also work with *.apk* files.

If you sign the *.apk* file, you will then also need to run it through a tool called zipalign, which will make sure that the compressed parts of the file are lined up on byte boundaries. Android wants them byte-aligned so it can read them easily without needing to uncompress the file.

Android Studio will do all of this for you if you choose Generate Signed APK from the Build menu.

So that's how your app gets converted from Java source code into an installable file. But how does it get installed and run on your device?

Say hello to the Android Debug Bridge (adb)

All communication between your development machine and your Android device takes place over the Android Debug Bridge. There are two sides to the bridge: a command-line tool on your dev machine called adb, and a daemon process on your Android device called adbd (the Android Debug Bridge Daemon).

The `adb` command on your development machine will run a copy of itself in the background, called the ADB server. The server receives commands over network port 5037, and sends them to the adbd process on the device. If you want to copy or read a file, install an app, or look at the logcat information for an app, then all of this information passes back and forth across the debug bridge.

So when the build system wants to install your APK file, it does so by sending a command like this to the debug bridge:

```
adb install bitsandpizzas.apk
```

The file will then be transferred to a virtual device, or over a USB cable to a real device, and be installed into the */data/app/* directory, where it will sit waiting for the app to be run.

How apps come alive: running your APK file

When your app is run, whether it's been started by a user pressing its launch icon or because the IDE has started it, the Android device needs to turn that *.apk* file into a process running in memory.

It does this using a process called Zygote. Zygote is like a half-started process. It's allocated some memory and already contains the main Android libraries. It has everything it needs, in fact, except for the code that's specific to your app.

When Android runs your app, it first creates a copy (a.k.a. fork) of the Zygote process, and then tells the copied process to load your application code. So why does Android leave this half-started process hanging around? Why not just start a fresh process from scratch for each app? It's because of performance. It can take Android a long time to create a new process from scratch, but it can fork a copy of an existing process in a split second.

Android converts the .dex code to OAT format

The new app process now needs to load the code that's specific to your app. Remember, your app code is stored in the *classes.dex* file inside your *.apk* package. So the *classes.dex* file is extracted from the *.apk* and placed into a separate directory. But rather than simply use the *classes.dex* file, Android will convert the Dalvik byte codes in *classes.dex* into native machine code. Technically, the *classes.dex* will be converted into an ELF shared object. Android calls this library format OAT, and calls the tool that converts the *classes.dex* file dex2oat.

The converted file is stored into a directory named something like this:

/data/dalvik-cache/data@app@com.hfad.bitsandpizzas@base.apk@classes.dex

This file can then be loaded as a native library by the application process, and the app appears on the screen.

appendix iv: adb

The Android Debug Bridge

In this book, we've focused on using an IDE for all your Android needs. But there are times when using a command-line tool can be plain useful, like those times when Android Studio can't see your Android device but you just *know* it's there. In this chapter, we'll introduce you to the **Android Debug Bridge (or adb)**, a command-line tool you can use to communicate with the emulator or Android devices.

adb: your command-line pal

Every time your development machine needs to talk to an Android device, whether it's a real device connected with a USB cable, or a virtual device running in an emulator, it does so by using the **Android Debug Bridge** (**adb**). The adb is a process that's controlled by a command that's also called `adb`.

The `adb` command is stored in the *platform-tools* directory of the Android System Developer's Kit on your computer. If you add the *platform-tools* directory to your `PATH`, you will be able to run adb from the command line.

In a terminal or at a command prompt, you can use it like this:

Interactive Session

```
$ adb devices
List of devices attached
emulator-5554     device
$
```

The `adb devices` command means "Tell me which Android devices you are connected to." The `adb` command works by talking to an adb server process, which runs in the background. The adb server is sometimes called the *adb dæmon* or *adbd*. When you enter an `adb` command in a terminal, a request is sent to network port 5037 on your machine. The adbd listens for commands to come in on this port. When Android Studio wants to run an app, or check the log output, or do anything else that involves talking to an Android device, it will do it via port 5037.

When the adbd receives a command, it will forward it to a separate adbd process that's running in the relevant Android device. This process will then be able to make changes to the Android device or return the requested information.

Sometimes, if the adb server isn't running, the `adb` command will need to start it:

Interactive Session

```
$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
emulator-5554     device
$
```

Likewise, if ever you plug in an Android device and Android Studio can't see it, you can manually kill the adb server and restart it:

Interactive Session

```
$ adb devices
List of devices attached
$ adb kill-server
$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
$ adb devices
List of devices attached
emulator-5554     device
$
```

By killing and restarting the server, you force adb to get back in touch with any connected Android devices.

Running a shell

Most of the time you won't use adb directly; you'll let an IDE like Android Studio do the work for you. But there are times when it can be useful to go to the command line and interact with your devices directly.

One example is if you want to run a shell on your device:

Interactive Session

```
$ adb shell
root@generic_x86:/ #
```

The `adb shell` command will open up an interactive shell directly on the Android device. If you have more than one device attached, you can indicate which device you mean with the `-s` option, followed by the name given by the `adb devices` command. For example, `adb -s emulator-5554 shell` will open a shell on the emulator.

Once you open a shell to your device, you can run a lot of the standard Linux commands:

Interactive Session

```
$ adb shell
root@generic_x86:/ # ls
acct
cache
charger
config
d
data
default.prop
dev
etc
file_contexts
....
1|root@generic_x86:/ # df
Filesystem              Size     Used     Free   Blksize
/dev                  439.8M    60.0K   439.8M   4096
/mnt/asec             439.8M     0.0K   439.8M   4096
/mnt/obb              439.8M     0.0K   439.8M   4096
/system               738.2M   533.0M   205.2M   4096
/data                 541.3M   237.8M   303.5M   4096
/cache                 65.0M     4.3M    60.6M   4096
/mnt/media_rw/sdcard  196.9M     4.5K   196.9M   512
/storage/sdcard       196.9M     4.5K   196.9M   512
root@generic_x86:/ #
```

Useful shell commands

If you open a shell to Android, you have access to a whole bunch of command line tools. Here are just a few:

Command	Description	Example (and what it does)
pm	Package management tool.	`pm list packages` (this lists all installed apps) `pm path com.hfad.bitzandpizzas` (find where an app is installed) `pm –help` (show other options)
ps	Process status.	`ps` (lists all processes and their IDs)
dexdump	Display details of an APK.	`dexdump -d /data/app/com.hfad.bitzandpizzas-2/base.apk` (disassemble an app)
lsof	List a process's open files and other connections.	`lsof -p 1234` (show what process with id 1234 is doing)
screencap	Take a screenshot.	`screencap -p /sdcard/screenshot.png` (save current screenshot to */sdcard/screenshot.png*, and get it off the device with `adb pull /sdcard/screenshot.png`)
top	Show busiest processes.	`top -m 5` (show the top five processes)

Each of these examples works from an interactive shell prompt, but you can also pass them direct to the shell command from your development machine. For example, this command will show the apps installed on your device:

Interactive Session

```
$ adb shell pm list packages
```

Kill the adb server

Sometimes the connection can fail between your development machine and your device. If this happens, you can reset the connection by killing the adb server:

Interactive Session

```
$ adb kill-server
```

The next time you run an adb command, the server will restart and a fresh connection will be made.

Get the output from logcat

All of the apps running on your Android device send their output to a central stream called the logcat. You can see the live output from the logcat by running the `adb logcat` command:

Interactive Session

```
$ adb logcat
--------- beginning of system
I/Vold    (  936): Vold 2.1 (the revenge) firing up
D/Vold    (  936): Volume sdcard state changing -1
(Initializing) -> 0 (No-Media)
W/DirectVolume(  936): Deprecated implied prefix pattern
detected, please use '/devices/platform/goldfish_mmc.0*'
instead
...
```

The logcat output will keep streaming until you stop it. It can be useful to run `adb logcat` if you want to store the output in a file. The `adb logcat` command is used by Android Studio to produce the output you see in the Devices/logcat panel.

Copying files to/from your device

The `adb pull` and `adb push` commands can be used to transfer files back and forth. For example, here we are copying the */default.prop/* properties file into a local file called *1.txt*:

Interactive Session

```
$ adb pull /default.prop 1.txt
28 KB/s (281 bytes in 0.009s)
$ cat 1.txt
#
# ADDITIONAL_DEFAULT_PROPERTIES
#
ro.secure=0
ro.allow.mock.location=1
ro.debuggable=1
ro.zygote=zygote32
dalvik.vm.dex2oat-Xms=64m
dalvik.vm.dex2oat-Xmx=512m
dalvik.vm.image-dex2oat-Xms=64m
dalvik.vm.image-dex2oat-Xmx=64m
ro.dalvik.vm.native.bridge=0
persist.sys.usb.config=adb
$
```

And much, much more...

There are many, many commands that you can run using adb: you can back up and restore databases (very useful if you need to debug a problem with a database app), start the adb server on a different port, reboot machines, or just find out a lot of information about the running devices. To find out all the options available, just type `adb` on the command line:

Interactive Session

```
$ adb
Android Debug Bridge version 1.0.32
 -a                          - directs adb to listen on all
interfaces for a connection
 -d                          - directs command to the only
connected USB device
                               returns an error if more than
one USB device is present.
 -e                          - directs command to the only
running emulator.
returns an error if more than one emulator is ....
```

appendix v: the android emulator

Speeding Things Up

Ever felt like you were spending all your time waiting for the emulator? There's no doubt that using the Android emulator is useful. It allows you to see how your app will run on devices other than the physical ones you have access to. But at times it can feel a little...sluggish. In this appendix, we'll explain why the emulator can seem slow. Even better, we'll give you a few tips we've learned for **speeding it up**.

Why the emulator is so slow

When you're writing Android apps, you'll spend a lot of time waiting for the Android emulator to start up or deploy your code. Why is that? Why is the Android emulator so slooooow? If you've ever written iPhone code, you know how fast the iPhone simulator is. If it's possible for the iPhone, then why not for Android?

There's a clue in the names: the iPhone *simulator* and the Android *emulator*.

The iPhone simulator simulates a device running the iOS operating system. All of the code for iOS is compiled to run natively on the Mac and the iPhone simulator runs at Mac-native speed. That means it can simulate an iPhone boot-up in just a few seconds.

The Android emulator works in a completely different way. It uses an open source application called QEMU (or Quick Emulator) to emulate the entire Android hardware device. It runs code that interprets machine code that's intended to be run by the device's processor. It has code that emulates the storage system, the screen, and pretty much every other piece of physical equipment on an Android device.

All the Android Virtual Devices run on an emulator called QEMU.

An emulator like QEMU creates a much more realistic representation of a virtual device than something like the iPhone simulator does, but the downside is that it has to do far more work for even simple operations like reading from disk or displaying something on a screen. That's why the emulator takes so long to boot up a device. It has to pretend to be every little hardware component inside the device, and it has to interpret every single instruction.

How to speed up your Android development

1. Use a real device

The simplest way to speed up your development process is by using a real device. A real device will boot up much faster than an emulated one, and it will probably deploy and run apps a lot more quickly. If you want to develop on a real device, you may want to go into "Developer options" and check the Stay Awake option. This will prevent your device from locking the screen, which is useful if you are repeatedly deploying to it.

2. Use an emulator snapshot

Booting up is one of the slowest things the emulator does. If you save a snapshot of the device while it's running, the emulator will be able to reset itself to this state without having to go through the boot-up process. To use a snapshot with your device, open the AVD manager from the Android Studio menu by selecting Tools→Android→AVD Manager, edit the AVD by clicking on the Edit symbol, and then check the "Store a snapshot for faster startup" option.

This will save a snapshot of what the memory looks like when the device is running. The emulator will be able to restore the memory in this state without booting the device.

3. Use hardware acceleration

By default, the QEMU emulator will have to interpret each machine code instruction on the virtual device. That means it's very flexible because it can pretend to be lots of different CPUs, but it's one of the main reasons why the emulator is slow. Fortunately, there's a way to get your development machine to run the machine code instructions directly. There are two main types of Android Virtual Device: ARM machines and x86 machines. If you create an x86 Android device and your development machine is using a particular type of Intel x86 CPU, then you can configure your emulator to run the Android machine code instructions directly on your Intel CPU.

You will need to install Intel's Hardware Accelerated Execution Manager (HAXM). At the time of writing , you can find HAXM here:

https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager

If it's moved, a quick web search should track it down.

HAXM is a hypervisor. That means it can switch your CPU into a special mode to run virtual machine instructions directly. However, HAXM will only run on Intel processors that support Intel Virtualization Technology. But if your development machine is compatible, then HAXM will make your AVD run much faster.

4. Use Instant Run

Since Android Studio 2.3, it's been possible to redeploy apps much more quickly using the Instant Run utility. This utility allows Android Studio to recompile just the files that have changed, and then patch the application currently running on the device. This means that instead of waiting for a minute or so while the application is recompiled and deployed, you can see your changes running in just a few seconds.

To use Instant Run, click on the Apply Changes option in the Run menu, or click on the lightning bolt icon in the toolbar:

There are a couple of conditions on using Instant Run. First, your app's target needs to be at least API 15. Second, you need to deploy to a device that runs API 21 or above. So long as you satisfy both these conditions, you'll find that Instant Run is by far the fastest way of getting your code up and running.

appendix vi: leftovers

The Top Ten Things (we didn't cover)

Even after all that, there's still a little more.

There are just a few more things we think you need to know. We wouldn't feel right about ignoring them, and we really wanted to give you a book you'd be able to lift without extensive training at the local gym. Before you put down the book, **read through these tidbits**.

1. Distributing your app

Once you've developed your app, you'll probably want to make it available to other users. You'll likely want to do this by releasing your app through an app marketplace such as Google Play.

There are two stages to this process: preparing your app for release, and then releasing it.

Preparing your app for release

Before you can release your app, you need to configure, build, and test a release version of it. This includes tasks such as deciding on an icon for your app and modifying *AndroidManifest.xml* so that only devices that are able to run your app are able to download it.

Before you release your app, make sure that you test it on at least one tablet and one phone to check that it looks the way you expect and its performance is acceptable.

You can find further details of how to prepare your app for release here:

http://developer.android.com/tools/publishing/preparing.html

Releasing your app

This stage includes publicizing your app, selling it, and distributing it.

To release your app on the Play Store, you need to register for a publisher account and use the Developer Console to publish your app. You can find further details here:

http://developer.android.com/distribute/googleplay/start.html

For ideas on how to best target your app to your users and build a buzz about it, we suggest you explore the documents here:

http://developer.android.com/distribute/index.html

2. Content providers

You've seen how to use intents to start activities in other apps. As an example, you can start the Messaging app to send the text you pass to it. But what if you want to use another app's data in your own app? For example, what if you want to use Contacts data in your app to perform some task, or insert a new Calendar event?

You can't access another app's data by interrogating its database, Instead, you use a **content provider**, which is an interface that allows apps to share data in a controlled way. It allows you to perform queries to read the data, insert new records, and update or delete existing records.

If you want other apps to use your data, you can create your own content provider.

You can find out more about the concept of content providers here:

http://developer.android.com/guide/topics/providers/content-providers.html

Here's a guide on using Contacts data in your app:

http://developer.android.com/guide/topics/providers/contacts-provider.html

And here's a guide on using Calendar data:

http://developer.android.com/guide/topics/providers/calendar-provider.html

3. Loaders

If you do a lot of work with databases or content providers, sooner or later you'll encounter **loaders**. A loader helps you load data so that it can be displayed in an activity or fragment.

Loaders run on separate threads in the background, and make it easier to manage threads by providing callback methods. They persist and cache data across configuration changes so that if, for example, the user rotates their device, the app doesn't create duplicate queries. You can also get them to notify your app when the underlying data changes so that you can deal with the change in your views.

The Loader API includes a generic `Loader` class, which is the base class for all loaders. You can create your own loader by extending this class, or you can use one of the built-in subclasses: `AsyncTaskLoader` or `CursorLoader`. `AsyncTaskLoader` uses an `AsyncTask`, and `CursorLoader` loads data from a content provider.

You can find out more about loaders here:

https://developer.android.com/guide/components/loaders.html

4. Sync adapters

Sync adapters allow you to synchronize data between an Android device and a web server. This allows you to do things like back up the user's data to a web server, for instance, or transfer data to a device so that it can be used offline.

Sync adapters have a number of advantages over designing your own data transfer mechanism.

- They allow you to automate data transfer based on specific criteria—for example, time of day or data changes.
- They automatically check for network connectivity, and only run when the device has a network connection.
- Sync adapter–based data transfers run in batches, which helps improve battery performance.
- They let you add user credentials or a server login to the data transfer.

You can find out how to use sync adapters here:

https://developer.android.com/training/sync-adapters/index.html

5. Broadcasts

Suppose you want your app to react in some way when a system event occurs. You may, for example, have built a music app, and you want it to stop playing music if the headphones are removed. How can your app tell when these events occur?

Android broadcasts system events when they occur, including things like the device running low on power, a new incoming phone call, or the system getting booted. You can listen for these events by creating a **broadcast receiver**. Broadcast receivers allow you to subscribe to particular broadcast messages so that your app can respond to particular system events.

Your app can also send custom broadcast messages to notify other apps of events.

You can find out more about broadcasts here:

https://developer.android.com/guide/components/broadcasts.html

6. The WebView class

If you want to provide your users with access to web content, you have two options. The first option is to open the web content with an external app like Chrome or Firefox. The second option is to display the content inside your app using the `WebView` class.

The `WebView` class allows you to display the contents of a web page inside your activity's layout. You can use it to deliver an entire web app as a client application, or to deliver individual web pages. This approach is useful if there's content in your app you might need to update, such as an end-user agreement or user guide.

You add a `WebView` to your app by including it in your layout:

```
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

You then tell it which web page it should load by calling its `loadUrl()` method:

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.loadUrl("http://www.oreilly.com/");
```

You also need to specify that the app must have Internet access by adding the `INTERNET` permission to *AndroidManifest.xml*:

```
<manifest ... >
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

You can find out more about using web content in your apps here:

http://developer.android.com/guide/webapps/index.html

7. Settings

Many apps include a settings screen so that the user can record their preferences. As an example, an email app may allow the user to specify whether they want to see a confirmation dialog before sending an email:

This is the settings screen for the Gmail app.

You build a settings screen for your app using the Preferences API. This allows you to add individual preferences, and record a value for each preference. These values are recorded in a shared preferences file for your app.

You can find out more about creating settings screens here:

https://developer.android.com/guide/topics/ui/settings.html

8. Animation

As Android devices increasingly take advantage of the power of their built-in graphics hardware, animation is being used more and more to improve the user's app experience.

There are several types of animation that you can perform in Android:

Property animation

Property animation relies on the fact that the visual components in an Android app use a lot of numeric properties to describe their appearance. If you change the value of a property like the height or the width of a view, you can make it animate. That's what property animation is: smoothly animating the properties of visual components over time.

View animations

A lot of animations can be created declaratively as XML resources. So you can have XML files that use a standard set of animations (like scaling, translation, and rotation) to create effects that you can call from your code. The wonderful thing about declarative view animations is that they are decoupled from your Java code, so they are very easy to port from one app project to another.

Activity transitions

Let's say you write an app that displays a list of items with names and images. You click on an item and you're taken to a detail view of it. The activity that shows you more detail will probably use the same image that appeared in the previous list activity.

Activity transitions allow you to animate a view from one activity that will also appear in the next activity. So you can make an image from a list smoothly animate across the screen to the position it takes in the next activity. This will give your app a more seamless feel.

To learn more about Android animations see:

https://developer.android.com/guide/topics/graphics/index.html

To learn about activity transitions and material design, see:

https://developer.android.com/training/material/animations.html

9. App widgets

An app widget is a small application view that you can add to other apps or your home screen. It gives you direct access to an app's core content or functionality from your home screen without you having to launch the app.

Here's an example of an app widget:

This is an app widget. It gives you direct access to the app's core functionality without you having to launch the app.

To find out how you create your own app widgets, look here:

http://developer.android.com/guide/topics/appwidgets/index.html

10. Automated testing

All modern development relies heavily on automated testing. If you create an app that's intended to be used by thousands or even millions of people, you will quickly lose users if it's flaky or keeps crashing.

There are many ways to automatically test your app, but they generally fall into two categories: **unit testing** and **on-device testing** (sometimes called *instrumentation testing*).

Unit tests

Unit tests run on your development machine, and they check the individual pieces—or units—of your code. The most popular unit testing framework is **JUnit**, and Android Studio will probably bundle JUnit into your project. Unit tests live in the *app/src/test* folder of your project, and a typical test method looks something like this:

```
@Test
public void returnsTheCorrectAmberBeers() {
    BeerExpert beerExpert = new BeerExpert();
    assertArrayEquals(new String[]{"Jack Amber", "Red Moose"},
            beerExpert.getBrands("amber").toArray());
}
```

You can find more out about JUnit here:

http://junit.org

On-device tests

On-device tests run inside an emulator or physical device and check the fully assembled app. They work by installing a separate package alongside your app that uses a software layer called *instrumentation* to interact with your app in the same way as a user. An increasingly popular framework for on-device testing is called **Espresso**, and Android Studio will probably bundle it in to your project. On-device tests live in the *app/src/androidTest* folder, and Espresso tests look something like this:

```
@Test
public void ifYouDoNotChangeTheColorThenYouGetAmber() {
    onView(withId(R.id.find_beer)).perform(click());
    onView(withId(R.id.brands)).check(matches(withText(
            "Jail Pale Ale\nGout Stout\n")));
}
```

When you run an on-device test, you will see the app running on your phone or tablet and responding to keypresses and gestures just as if a person were using it.

You can find out more about Espresso testing here:

https://developer.android.com/training/testing/ui-testing/espresso-testing.html

Index

A

action bar, 314. *See also* app bar
ActionBarActivity class, 298
ActionBar class, 306, 314
ActionBarDrawerToggle class, 607
actions
  about, 100
  in app bar
    adding, 315–317, 319–325
    icons for, 320, 322
    menu for, 321, 323
    method for, 324–325
    title for, 320, 322
  category specified with, 106, 117
  determining activities for, 105–106
  share actions, 332–335
  specifying in an intent, 100–104
  types of, 101
activities
  about, 2, 12, 31, 38, 120–121
  backward compatibility for, 295, 298
  calling custom Java classes from, 71–74
  calling methods from, 59–62
  class hierarchy for, 139, 297
  class name for, 85
  converting to fragments, 438–449
  creating, 13–14, 61, 67–68
  declaring in manifest file, 85
  default, 41
  editing. *See* code editor; design editor
  location in project, 17
  main, specifying, 84, 120
  multiple
    chaining, 78
    creating, 78–83
    passing text to, 90–95
    retrieving text from, 96–97
    starting with intents, 86–89
  navigating. *See* navigation
  organizing, 249–255, 290–291
    category activities, 249–250, 252–255, 267–268, 277–278
    detail/edit activities, 249–250, 253–255, 279–283
    top-level activities, 249–250, 252–255
  in other apps
    determining from actions, 105–106
    none available, 117
    starting with intents, 99–104
    users always choosing, 112–117
    users choosing default, 111
    users choosing if multiple, 100, 104, 111
  parent activity, 328
  states of. *See* activity lifecycle
Activity class, 61, 139, 297
activity lifecycle
  about, 137–138, 146–147
  class hierarchy for, 139
  compared to fragment lifecycle, 439
  methods associated with, 137–139, 147, 167, 439
  states in
    based on app's focus, 154–157
    based on app's visibility, 146–147
    list of, 137–138, 146–147
    saving and restoring, 140–141, 145
ActivityNotFoundException, 103, 117
activity transitions, 868
adapters
  array. *See* ArrayAdapter class
  decoupling, with interface, 572–574
  recycler view. *See* recycler view adapter
  sync adapters, 864
AdapterView class, 261, 269
adb (Android Debug Bridge)
  about, 846, 850–851
  copying files, 855
  killing adb server, 854
  logcat output, 854
  running a shell, 852–853

ADD COLUMN command, 650
addDrawerListener() method, DrawerLayout, 607
ALTER TABLE command, 649
Android application package. *See* APK files
Android apps. *See also* projects
  about, 120–121
  activities in. *See* activities
  adapting for different device sizes, 340–341, 394–398, 402–412
  app bar for. *See* app bar
  back button functionality for, 413–415
  building, 7–10, 13–15, 39
  configuration of, affected by rotation, 134–136, 145, 156
  designing, 248–250
  devices supported by, 10, 340–341. *See also* fragments
  distributing, 862
  examples of. *See* examples
  focus/visibility of, 146–147, 154–157
  icons for, 84, 299
  Java for. *See* Java
  languages used in, 4
  layouts for. *See* layouts
  manifest file for. *See* AndroidManifest.xml file
  minimum SDK for, 10–11
  navigating. *See* navigation
  package name for, 9, 84
  processes used by, 120–121, 133
  resource files for. *See* resource files
  running activities in other apps, 100–108
  running in emulator, 23–30, 49, 117
  running on physical devices, 109–111, 117
  settings screen for, 867
  structuring, 249–250, 252–255, 290–291
  theme for. *See* themes
Android debug bridge. *See* ADB
Android devices
  about, 2
  adapting apps for different sizes of, 340–341, 394–398, 402–412. *See also* fragments
  API level on. *See* Android SDK (API level)
  rotation of
    configuration affected by, 134–136, 145, 156
    saving activity state for, 140–141
    saving fragment state for, 427–431
  running apps on, 109–111, 117
  virtual, for testing. *See* AVD (Android virtual device)
Android emulator
  about, 23
  performance of, 858–860
  running apps in, 23–30, 49, 117
AndroidManifest.xml file
  about, 17, 84–85
  activity hierarchy in, 328
  app bar attributes in, 299–301, 318–319
  intent filters in, 105–106
  language settings in, 172
  main activity in, 120, 132
  permissions needed in, 791
  screen size settings in, 403
  started services in, 745
  themes in, 589
Android platform
  about, 2–3
  versions of. *See* Android SDK (API level)
Android Runtime (ART), 3, 30, 842–843
Android SDK (API level)
  about, 5
  features specific to, 171, 173, 293, 297, 298, 314, 328
  libraries for, 16, 294
  list of, 11
  permissions affected by, 791, 794
  specifying for AVD, 25, 400
  specifying minimum for apps, 10
Android Studio
  about, 5, 7
  alternatives to, 7
  console in, 28
  editors. *See* code editor; design editor
  installing, 6
  projects, creating, 8–10
  system requirements, 6
Android virtual device. *See* AVD
animated toolbar. *See* collapsing toolbar; scrolling toolbar
animation, 868
API level. *See* Android SDK (API level)
APK files, 27, 30, 845–846
app bar
  about, 291–293
  activity titles in, 317
  adding actions to, 315–317, 319–325
  attributes in manifest file for, 299–301, 318–319
  icon in, 299
  label for, 299, 318–319

removing, 308
replacing with toolbar, 292, 306–313
sharing content on, 331–335
tabs in, 499–500
themes required for, 293, 296–298, 299–300
Up button for, 327–330
AppBarLayout element, 499–500, 518–519
AppCompatActivity class, 297–298, 307
AppCompat Library. *See* v7 AppCompat Library
app folder, 17
application framework, 3. *See also* APIs
application name, 9
applications, core. *See* core applications
app_name string resource, 51
apps. *See* Android apps
AppTheme attribute, 589–590
app widgets, 869
ArrayAdapter class, 269–271, 275, 287, 375–377
@array reference, 57
array resources, 56–57, 210, 259
ART (Android Runtime), 3, 30, 842–843
AsyncTask class, 724, 729, 731, 737
?attr reference, 309
automated testing, 870
AVD (Android virtual device)
about, 23
compared to design editor, 49
creating for smartphone, 24–26
creating for tablet, 399–401
AWT, 4

B

Back button
compared to Up button, 327
enabling, 413–415
background, services running in. *See* services
background thread, 720–731, 736
back stack, 414–415
backward compatibility, for activities, 295, 298
Bates, Bert (Head First Java), 4
Beer Adviser app, 38–76
activities, 61–69
button, 59–60
Java class, 70–74
layout, 41–48
project, 40
spinner, 56–58
String resources, 50–54
Beighley, Lynn (Head First SQL), 675
biases for views, 231
Binder class, 786
bindService() method, activity, 778
blueprint tool
about, 226
aligning views, 238
biases, setting, 231
centering views, 230
horizontal constraints, 227
inferring contraints, 240–241, 245
margins, setting, 228
size of views, changing, 232–233
vertical constraints, 228
view properties, editing, 241
widgets, adding, 226, 240, 242–243
books and publications
Head First Java (Sierra; Bates), 4
Head First SQL (Beighley), 675
bound services
about, 740, 768
binding to an activity, 771, 774–778
calling methods from, 780–785
in combination with started services, 786
compared to started services, 786
creating, 770, 772
displaying results of, 773
lifecycle of, 787–788
other apps using, 786
broadcasts, 865
build folder, 17
build.gradle files, 834–835
built-in services
about, 740
location. *See* Location Services
notification. *See* notification service
Bundle class
about, 145
restoring state data from, 141, 144
saving state data in, 140, 143
Button element. *See also* FAB (floating action button)
about, 203
adding, 43

calling methods
in activity, 60–62
in fragment, 452–460
code for, 44–47, 80
images added to, 213
widget for, in blueprint tool, 226

C

CalledFromWrongThreadException, 127
cardCornerRadius attribute, card view, 543
cardElevation attribute, card view, 543
CardView element, 543
CardView Library, 294, 542
card views
about, 542
adding data to, 545, 550
creating, 543–544
displaying in recycler view, 558, 560
CatChat app, 581–620
about, 581–584
Drafts option, 586
Feedback option, 592
header for, 594–595
Help option, 591
Inbox option, 585
menu for, 596–601
SentItems option, 587
Trash option, 588
category activities, 249–250, 252–255, 267–268, 277–278, 290
centering views, 230
chaining activities, 78
changeCursor() method, cursor adapter, 715–717
checkableBehavior attribute, 598
CheckBox element, 206–207, 696–700
checkSelfPermission() method, ContextCompat, 794, 800
check task, Gradle, 837
choosers, 112–117
classes. *See* Java classes
clean task, Gradle, 837
closeDrawer() method, DrawerLayout, 614
close() method, cursor, 672, 683
close() method, SQLiteDatabase, 672, 683
code editor
about, 18, 32
activity file in, 21–22
layout file in, 19–20, 41, 44–46
collapsing toolbar, 507, 517–525
CollapsingToolbarLayout element, 518–521
color resources, 304
colors, for themes, 303
columnCount attribute, GridLayout, 823
company domain, 9
constraint layout
about, 223
aligning views, 238
alternatives to, 245
biases, setting, 231
centering views, 230
horizontal constraints, adding, 227
inferring contraints, 240–241, 245
library for, 224
margins, setting, 228–229
size of views, changing, 232–233
specifying in main activity, 225
vertical constraints, adding, 228
widgets, adding, 226, 240, 242–243
XML code for, 229
Constraint Layout Library, 224, 294
contentDescription attribute, ImageView, 212
content providers, 863
contentScrim attribute, CollapsingToolbarLayout, 523
ContentValues object, 632, 645
Context class, 139, 752
ContextThemeWrapper class, 139
ContextWrapper class, 139, 752
CoordinatorLayout element, 508–511, 518, 527
core applications, 3
core libraries, 3. *See also* libraries
createChooser() method, Intent, 112–117
CREATE TABLE command, 631
Cursor class, 624
cursors, SQLite
about, 624, 663
adapter for, 679–681, 688–689, 715
closing, 672, 682–683
creating, 663–666, 678
navigating to records in, 670–671

refreshing, 714–718
retrieving values from, 672–674
custom Java classes. *See* Java classes

D

data
adapters for. *See* adapters
from other apps, content providers for, 863
loaders for, 864
sharing with other apps. *See* share action provider
storing as string resources, 259
storing in classes, 256, 268–271, 360
web content, 866
database. *See* SQLite database
delete() method, SQLiteDatabase, 647
density-independent pixels (dp), 171
design editor. *See also* blueprint tool
about, 18, 32, 42
adding GUI components, 43
compared to AVD, 49
XML changes reflected in, 48
Design Support Library
about, 294, 506
AppBarLayout element, 499–500
collapsing toolbar, 517–525
FAB (floating action button), 507, 526–529
navigation drawers, 584
scrolling toolbar, 508–515
snackbar, 507, 526, 530–533
TabLayout element, 499, 501
detail/edit activities, 249–250, 253–255, 279–283, 290
development environment. *See* Android SDK; Android Studio
devices. *See* Android devices
DEX files, 843–845, 847
dimension resources, 174
dimens.xml file, 174
distributing apps, 862
doInBackground() method, AsyncTask, 724, 726, 731
dp (density-independent pixels), 171
drawable attributes, Button, 213
@drawable reference, 212
drawable resource folders. *See* image resources
DrawerLayout element, 602–603. *See also* navigation drawers
drop-down list. *See* Spinner element
DROP TABLE command, 650
dynamic fragments. *See* fragments: dynamic

E

editors. *See* blueprint tool; code editor; design editor
EditText element
about, 80, 178, 202
code for, 80
hint, 178
email app example. *See* CatChat app
email grid layout example, 825–832
emulator. *See* Android emulator
entries attribute, ListView, 259
entries attribute, Spinner, 57, 210
Espresso tests, 871
event listeners
about, 199
for card views, 572–576
compared to onClick attribute, 264
for ListView element, 261–263
for fragments, 455–460
for list fragments, 385–387
for ListView element, 708, 710–712
location listeners, 792
for navigation drawers, 608
for snackbar actions, 530
examples
Beer Adviser app. *See* Beer Adviser app
CatChat app. *See* CatChat app
email grid layout, 825–832
Joke app. *See* Joke app
My Constraint Layout app. *See* My Constraint Layout app
My First App. *See* My First App
My Messenger app. *See* My Messenger app
Odometer app. *See* Odometer app
Pizza app. *See* Pizza app
source code for, xxxvi
Starbuzz Coffee app. *See* Starbuzz Coffee app
Stopwatch app. *See* Stopwatch app
Workout app. *See* Workout app
execSQL() method, SQLiteDatabase, 631, 650
execute() method, AsyncTask, 731

F

FAB (floating action button), 507, 526–529
findViewById() method, view, 63–64, 68
FloatingActionButton element, 527
focus
  activities having, 154–157
  views handling, 199
FragmentActivity class, 354
fragment element, 352, 463
fragment lifecycle, 365–366, 439
fragment manager, 362, 419
FragmentPagerAdapter class, 491–493
fragments
  about, 342
  activities using
    adding fragment to, 352–353
    button linking main to detail activity, 346–347
    creating, 345
    interactions with, 359–369
    referencing fragment from, 363, 367
    setting values for, 367–369
  adding to project, 348–349
  app structure for, 343–344
  back button functionality with, 413–415
  code for, 349–350
  converting activities to, 438–449
  data for, 360
  dynamic
    about, 434
    adding, 435–449, 469–477
    calling methods from, 450–460
    code for, 440–446, 458–460
    layout for, 447–449
  ID for, 361–362
  layouts for, 349–351, 408–410
  list fragments
    connecting data to, 375–377
    connecting to detail activity, 384–389
    creating, 372–374
    displaying in main activity, 378–381
    listener for, in main activity, 385–387
  methods associated with, 365–366, 439
  nested, 474–477
  replacing programmatically, 416–425
  state of, saving, 427–431, 464–467
  swiping through, 489–493
  for tab navigation, 483–488
  v7 AppCompat Library for, 345, 354
  view for, creating, 350
FragmentStatePagerAdapter class, 491, 493
fragment transactions, 419–423, 463–467, 472–475
FrameLayout element
  about, 188–190, 193
  gravity for view placement, 190
  height, 188
  nesting, 191–192
  order of views in, 190
  replacing fragments programmatically using, 416–425, 464–467, 471
  width, 188

G

getActivity() method, PendingIntent, 758
getBestProvider() method, LocationManager, 793
getCheckedRadioButtonId() method, RadioGroup, 208
getChildFragmentManager() method, fragment, 474–475
getContext() method, LayoutInflator, 376
getCount() method, FragmentPagerAdapter, 491–492
getFragmentManager() method
  activity, 362
  fragment, 472–473
getIntent() method, activity, 92, 96, 280
getIntExtra() method, Intent, 92
getItemCount() method, RecyclerView.Adapter, 547
getItem() method, FragmentPagerAdapter, 491–492
get*() methods, cursor, 672
getReadableDatabase() method, SQLiteOpenHelper, 662, 678
getSelectedItem() method, view, 67, 69, 210
getStringExtra() method, Intent, 92, 96
getString() method, string resource, 115
getSupportActionBar() method, activity, 329
getSupportFragmentManager() method, activity, 362
getText() method, EditText, 202
getView() method, fragment, 367, 370
getWritableDatabase() method, SQLiteOpenHelper, 662
GPS location provider, 793
Gradle build tool
  about, 7, 834
  build.gradle files, 834–835

check task, 837
custom tasks, 839
dependencies used by, 838
gradlew and gradlew.bat files, 836
plugins, 840
gradlew and gradlew.bat files, 836
gravity attribute, view, 182–183
GridLayout element
about, 823
adding views to, 824, 827–829
creating, 825–832
dimensions, defining, 823, 826
GridLayoutManager class, 556–557
group element, 598–599
GUI components. *See also* views; specific GUI components
adding, 43
inherited from View class, 44, 197–198
referencing, 63–64

H

Handler class, 128–129
HAXM (Hardware Accelerated Execution Manager), 859
headerLayout attribute, 602
Head First Java (Sierra; Bates), 4
Head First SQL (Beighley), 675
heads-up notification, 755, 757
hint attribute, EditText, 178, 202
horizontal constraints, 227
HorizontalScrollView element, 215

I

IBinder interface, 786
IBinder object, 770–771, 776
icon attribute, 299, 322
icons
for actions in app bar, 320
for app, default, 84, 299
built-in icons, 597
id attribute, view, 44, 175
IDE
Android Studio as. *See* Android Studio
not using, 7
ImageButton element, 214
image resources. *See also* icons; mipmap resource folders
adding, 189, 211, 257, 512
for Button elements, 213
for ImageButton elements, 214
for ImageView element, 212, 258
for navigation drawer header, 594
R.drawable references for, 212, 257
resolution options, folders for, 211
ImageView element, 211–212, 258, 523
include tag, 312, 314
inflate() method, LayoutInflator, 350
inflators, layout, 350, 365
inputType attribute, EditText, 202
insert() method, SQLiteDatabase, 632–633
installDebug task, Gradle, 837
Instant Run, 860
IntelliJ IDEA, 5
intent filter, 105–106
intents
about, 86
alternatives to, 91
category for, 106, 117
creating, 86–87, 277
implicit and explicit, 101, 105, 117
passing text using, 90–95
resolving activities and actions, 105–106
retrieving data from, 280–282
retrieving text from, 96–97
sharing content using, 331
starting activities, 86–89
starting activities in other apps, 99–104
IntentService class, 742, 744, 752
internationalization. *See* localization
isChecked() method, CheckBox, 206

J

Java
about, 2
activities. *See* activities
required knowledge of, xxxvi, 4
source file location, 16–17
Java classes
about, 38
calling from activities, 71–74
custom, creating, 70, 256
data stored in, 256, 268–271

java folder, 17
Java Virtual Machine (JVM), 842
JDBC, 624
Joke app
  logging messages, 743–746
  notifications, 755–766
  started services, 741–766
JUnit, 870
JVM (Java Virtual Machine), 842

L

label attribute, application, 299, 314, 318
layout_above attribute, RelativeLayout, 822
layout_align* attributes, RelativeLayout, 820, 822
layout_behavior attribute, ViewPager, 510, 519
layout_below attribute, RelativeLayout, 822
layout_center* attributes, RelativeLayout, 820
layout_collapseMode attribute, Toolbar, 519, 523
layout_column attribute, GridLayout, 827
layout_columnSpan attribute, GridLayout, 828
layout_gravity attribute, view, 47, 184–185, 190
layout_height attribute
  FrameLayout, 188
  LinearLayout, 171
  view, 44, 46, 47, 175, 180, 232–233
LayoutInflator class, 350, 365, 376
layout manager, for recycler view, 556–557
layout_margin attributes, view, 47, 176, 228–229
layout resource folders, 402–408
layout_row attribute, GridLayout, 827
layouts
  about, 2, 12, 31, 38, 170
  code for, 19–20, 33, 41, 44–46, 80
  constraint. *See* constraint layout
  creating, 13–14
  default, 41
  editing, 41–48
  for fragments. *See* fragments
  frame. *See* FrameLayout element
  grid. *See* GridLayout element
  inherited from ViewGroup class, 198, 200
  linear. *See* LinearLayout element
  nesting, 191–192, 222
  relative. *See* RelativeLayout element
  toolbars as, 311–313, 316
layout_scrollFlags attribute, Toolbar, 510, 519
layout_to* attributes, RelativeLayout, 822
layout_weight attribute, view, 179–181
layout_width attribute
  FrameLayout, 188
  LinearLayout, 171
  view, 44, 46, 47, 175, 232–233
libraries. *See also* specific libraries
  about, 3
  adding to project, 224
  location in project, 16
  Support Libraries, list of, 294
LinearLayout element
  about, 41, 45, 171, 187
  dimension resources, 174
  gravity for view contents, 182–183
  gravity for view placement, 184–185
  height, 171
  nesting, 191–192, 222
  order of views in, 175
  orientation, 172
  padding, 173
  weight of views in, 179–181
  width, 171
  xmlns:android attribute, 171
LinearLayoutManager class, 556–557
Linux kernel, 3
Listener interface, 385–387, 572–576
listeners. *See* event listeners
ListFragment class
  connecting data to, 375–377
  connecting to detail activity, 384–389
  creating, 372–374
  displaying in main activity, 378–381
  listener for, in main activity, 385–387
ListView element
  about, 251
  class hierarchy for, 570
  creating, 259–260, 705–709
  event listeners for, 261–263, 708, 710–712
loaders, 864
localization
  right-to-left languages, 172
  String resources, 50, 55
LocationListener class, 792
LocationManager class, 793

Location Services
about, 789
distance traveled, calculating, 796–797
library for, 790
listener for, 792
manager for, 793
permissions needed
checking for, 794, 800
declaring, 791
notification if denied, 806, 809–810
requesting from user, 802–804, 806
provider for, 793
requesting location updates, 794–795
stopping location updates, 797
logcat, viewing, 743, 854
Log class, 743
logging messages, 743–746

M

main activity, 84, 120
manifest file. *See* AndroidManifest.xml file
material design, 506. *See also* Design Support Library
menu attribute, NavigationView, 602
menu element, 321
menu resource folders, 321
messages. *See also* notification service
logging, 743–746
pop-up messages (toasts), 216
methods. *See also* specific methods
calling from activities, 59–62, 81
creating, 62–63, 65–66, 81
name of, 69
mipmap resource folders, 299
moveToFirst() method, cursor, 671
moveToLast() method, cursor, 671
moveToNext() method, cursor, 671
moveToPrevious() method, cursor, 671
My Constraint Layout app, 223–246
activities, 225
layout, 226–243
library, 224
String resources, 225
My First App, 7–36
activities, 12–15, 31
editors, 18–22
emulator, 23–30
folder structure, 16–17
layout, 31–34
project, 8–11
My Messenger app, 79–118
activities, 82–83, 90
choosers, 112–117
intents, 86–89, 92–108
layout, 80, 82–83, 91
manifest file, 84–85
running on device, 109–111
String resources, 81

N

navigation. *See also* app bar; ListView element; navigation drawers; tab navigation; toolbar
about, 250–251, 253–255, 291
Back button, 327, 413–415
Up button, 292, 327–330
navigation drawers
about, 580–583
adding to main activity, 583, 602–604
closing the drawer, 606, 614
compared to tab navigation, 580
drawer toggle for, 606–607
fragments for, 583, 585–592
header for, 583, 593–595
libraries for, 584
menu click behavior, 606, 608–613
menu options for, 583, 593, 596–601
multiple, 614
submenu in, 600
theme for, 589–590
toolbar for, 589
NavigationView element, 602–603, 608
NestedScrollView element, 513, 518–519
network location provider, 793
NotificationManager class, 759
Notification object, 759
notification service
about, 755, 762–763
action for, adding, 758
heads-up notification, 755, 757
issuing a notification, 759, 806, 809–810
library for, 756

notification builder for, 757
notification manager for, 759
notify() method, NotificationManager, 759

O

OAT format, 847
Odometer app, 769–816
bound services, 769–788
library, 790
Location Services, 789–800
permissions, 791, 802–816
onActivityCreated() method, fragment, 365
onAttach() method, fragment, 365
onBind() method, Service, 770, 787–788
onBindViewHolder() method, RecyclerView.Adapter, 550, 571
onButtonClicked() method, activity, 203, 214
onCheckboxClicked() method, CheckBox, 207
onClick attribute
Button, 60, 125, 203, 264
CheckBox, 207, 698–700
ImageButton, 214
RadioButton, 209
Switch, 205
ToggleButton, 204
onClickDone() method, activity, 529
OnClickListener interface, 455–460
onClickListener() method, view, 530
onClick() method
fragment, 456
Listener, 572
OnClickListener, 455
onCreate() method
activity, 61, 81, 96, 121, 133, 137, 138, 147, 167, 787–788
fragment, 365, 429, 430
service, 751
SQLiteOpenHelper, 628, 634–635
onCreateOptionsMenu() method, activity, 323
onCreateViewHolder() method, ViewHolder, 549
onCreateView() method, fragment, 349–350, 365, 374, 376
onDestroy() method
activity, 137, 138, 147, 167, 683, 787–788
fragment, 365
service, 751
onDestroyView() method, fragment, 365
onDetach() method, fragment, 365
on-device tests, 871
onDowngrade() method, SQLiteOpenHelper, 637, 641
onHandleEvent() method, IntentService, 742
onHandleIntent() method, IntentService, 744
OnItemClickListener class, 261–262
onItemClick() method, OnItemClickListener, 261–262
onListItemClick() method, list fragment, 373, 386
onLocationChanged() method, LocationListener, 792
onNavigationItemSelected() method, activity, 608–609
onOptionsItemSelected() method, activity, 324
onPause() method
activity, 154–157, 159, 167
fragment, 365
onPostExecute() method, AsyncTask, 724, 728
onPreExecute() method, AsyncTask, 724–725
onProgressUpdate() method, AsyncTask, 727
onProviderDisabled() method, LocationListener, 792
onProviderEnabled() method, LocationListener, 792
onRadioButtonClicked() method, RadioGroup, 209
onRequestPermissionResult() method, activity, 805
onRestart() method, activity, 146, 147, 153, 167
onResume() method
activity, 154–157, 159, 167
fragment, 365
onSaveInstanceState() method
activity, 140, 142, 143, 146
fragment, 429, 431
onServiceConnected() method, ServiceConnection, 775–776
onServiceDisconnected() method, ServiceConnection, 775, 777
onStartCommand() method, Service, 751
onStart() method
activity, 146, 147, 153, 167
fragment, 365, 367
onStatusChanged() method, LocationListener, 792
onStop() method
activity, 146, 147, 148–150, 167
fragment, 365
onSwitchClicked() method, Switch, 205
onToggleClicked() method, ToggleButton, 204

onUnbind() method, Service, 787–788
onUpgrade() method, SQLiteOpenHelper, 628, 637, 640, 642–650
Oracle JVM (Java Virtual Machine), 842
orderInCategory attribute, 322
organizing ideas. *See also* Starbuzz Coffee app
orientation attribute, LinearLayout, 45, 172

P

package name, 84
padding attributes, LinearLayout, 173
parent activity, 328
PendingIntent class, 758
performance
  of Android emulator, 858–860
  SQLite database affecting, 720
permissions
  API levels affecting, 791, 794
  checking for permissions granted, 794, 800
  declaring permissions needed, 791
  denied, issuing notification of, 806, 809–810
  requesting from user, 802–804, 806
Pizza app, 290–338, 482–536, 538–578
  actions, 315–326
  activities, 297–298
  adapters, 571–574
  app bar, 291–293, 299
  card views, 543–546, 550, 560
  collapsing toolbar, 517–525
  color resources, 304
  FABs, 526–529
  fragments, 485–488
  layout, 305
  libraries, 294–296, 307, 506
  recycler views, 538–539, 545–570, 575–576
  scrollbars, 508–515
  share action provider, 331–336
  snackbar, 526, 530–534
  tab navigation, 498–504
  themes and styles, 300–303
  toolbar, 306–314
  Up button, 327–329
  ViewPager, 489–493
Play Store, releasing apps to, 862
pop-up messages (toasts), 216
postDelayed() method, Handler, 128–129
post() method, Handler, 128–129
Preferences API, 867
processes, apps using, 120–121, 133. *See also* services; threads
projects. *See also* Android apps
  application name, 9
  company domain, 9
  creating, 8–10, 40–41
  files in, 15–17, 34
  libraries for, adding, 224
  location, 9
property animation, 868
publishProgress() method, AsyncTask, 727
putExtra() method, Intent, 92, 101
put() method, ContentValues, 632

Q

QEMU (Quick Emulator), 858
query() method, SQLiteDatabase, 663–666

R

RadioButton element, 208
RadioGroup element, 208–209
Random class, 772
R.drawable reference, 212, 257
recycler view adapter, 545–550, 554, 571–574
RecyclerView.Adapter class, 545–546
RecyclerView element, 554
RecyclerView Library, 294
recycler views
  about, 538–539
  class hierarchy for, 570
  creating, 553–555, 562–565
  data for, 547, 550
  layout manager for, 556–557
  responding to clicks, 566–576
  scrollbars for, 554
  views for, 548–549
RecyclerView-v7 Library, 542
RelativeLayout element
  about, 818
  positioning relative to other views, 821–822
  positioning relative to parent, 818–820

releasing apps, 862
removeUpdates() method, LocationManager, 797
RENAME TO command, 649
render thread, 720
requestLocationUpdates() method, LocationManager, 794
requestPermissions() method, ActivityCompat, 803–804, 806
res-auto namespace, 543
res folder, 17
resolution of images, 211
resource files. *See also* array resources; image resources; String resources
  about, 2
  color resources, 304
  dimension resources, 174
  layout resources, 402–408
  menu resources, 321
  mipmap resources, 299
  style resources, 300–301
  types of, folders for, 16–17, 403
resources. *See* books and publications; website resources
resources element, 54
R.java file, 17, 63, 69
rotation of device
  configuration changed by, 134–136, 145, 156
  saving activity state for, 140–141
  saving fragment state for, 427–431
roundIcon attribute, 299
rowCount attribute, GridLayout, 823
Runnable class, 128

S

Safari Books Online, xl
scale-independent pixels (sp), 201
scheduled services, 740
screens
  activities in. *See* activities
  density of, images based on, 211
  layouts for. *See* layouts
  size of, adapting apps for, 340–341, 394–398, 402–412
scrollbars attribute, recycler view, 554
scrolling toolbar, 507, 508–515
ScrollView element, 215
SDK. *See* Android SDK (API level)
Service class, 752, 770
ServiceConnection interface, 775–777
service element, 745
services
  about, 740
  bound. *See* bound services
  built-in, 740
  location. *See* Location Services
  notification. *See* notification service
  scheduled, 740
  started. *See* started services
setAction() method, Snackbar, 530
setAdapter() method
  ListView, 270
  RecyclerView, 554
  ViewPager, 493
setContentDescription() method, ImageView, 212, 282
setContentIntent() method, notification builder, 758
setContentView() method, activity, 61, 133, 137, 363
setDisplayHomeAsUpEnabled() method, ActionBar, 329
setImageResource() method, ImageView, 212, 282
setListAdapter() method, fragment, 376
setNavigationItemSelectedListener() method, NavigationView, 608
setShareIntent() method, ShareActionProvider, 333
setSupportActionBar() method, AppCompatActivity, 313, 314, 317
setText() method, TextView, 64, 201, 282
settings screen, 867
setType() method, Intent, 101
setupWithViewPager() method, TabLayout, 501
share action provider, 292, 331–335
shell, running with adb, 852–853
shortcuts. *See* app bar
showAsAction attribute, 322
Sierra, Kathy (Head First Java), 4
signing APK files, 845
SimpleCursorAdapter class, 681
slider. *See* Switch element
snackbar, 507, 526, 530–533
SnackBar class, 530
Spinner element
  about, 47, 48, 210
  setting values in, 64
  values for, 56–57
sp (scale-independent pixels), 201

SQLite database
about, 623–624
accessing in background thread, 720–731, 736
adding columns, 649, 650
alternatives to, 624
closing, 672, 682–683
conditions for columns, 647
conditions for queries, 666
creating, 629, 634–635
cursors
about, 624, 663
adapter for, 679–681, 688–689, 715
closing, 672, 682–683
creating, 663–666, 678
navigating to records in, 670–671
refreshing, 714–718
retrieving values from, 672–674
data types in, 630
deleting records, 647
deleting tables, 650
downgrading, 641
DrinkActivity. *See* DrinkActivity
files for, 623
getting a reference to, 662, 678
helper, 624, 626–628, 634–635
inserting data in, 632–633
location of, 623, 624
ordering records from query, 665
performance of, 720, 736
querying records, 663–666
renaming tables, 649
security for, 624
tables in, 630–631
updating records
programmatically, 645–647
from user input, 695–703, 705–706, 708–712
upgrading, 637–640, 642–650
version number of, 637–639
SQLiteDatabase class, 624
SQLiteException, 662, 675
SQLiteOpenHelper class, 624, 627–628
SQL (Structured Query Language), 631, 675
src attribute, ImageButton, 214
src attribute, ImageView, 212
src folder, 17
StaggeredGridLayoutManager class, 556–557
Starbuzz Coffee app, 248–288, 622–656, 658–692, 694–738
activities, 248–249, 252–255, 262–264, 267–272
adapters, 269–271
database, 626–656, 659–692, 694–718
DrinkActivity. *See* DrinkActivity
image resources, 257
intents, 277–285
Java classes, 256
layout, 258–260
listeners, 261–262, 276
navigation, 250–251
threads, 720–738
top level activity
adding favorites to. *See* top level activity
startActivity() method, activity, 86, 112, 117, 121
started services
about, 740–741, 748
class hierarchy for, 752
in combination with bound services, 786
compared to bound services, 786
creating, 741–742, 744
declaring in AndroidManifest.xml, 745
lifecycle of, 750–751
methods associated with, 750–752
starting, 746–747
startService() method, Intent, 747, 751
states, of activities. *See* activity lifecycle
Stopwatch app, 122–168, 434–480
activities, 125–127, 130–133
activity lifecycle, 138–139, 146–163
activity states, 134–144
dynamic fragments, 434–436, 444–460
fragment lifecycle, 439
fragment transactions, 463–469, 472–475
handlers, 128–129
layout, 123–124, 471
project, 122
String resources, 123
string-array element, 56
string element, 54
@string reference, 52, 81
String resources
about, 38, 50, 54, 55
action titles in, 320
adding, 512, 517

arrays of, 56–57, 210, 259
creating, 51, 81
getting value of, 115
location of, 54, 55
referencing strings in, 52, 81
updated in R.java file, 69
strings.xml file. *See* Array resources; String resources
Structured Query Language. *See* SQL
style element, 301
@style reference, 300
style resources, 300–301
styles
applying themes using, 300–301
customizing themes using, 303
Support Libraries. *See also* v7 AppCompat Library
adding to project, 296
list of, 294
supportsRtl attribute, application, 172
Swing, 4
swiping through fragments, 489–493
Switch element, 205
sync adapters, 864
syncState() method, ActionBarDrawerToggle, 607
system image. *See* Android SDK (API level)

T

TabLayout element, 499, 501
tab navigation
about, 482–484, 493
adding tabs to layout, 498–504
compared to navigation drawers, 580
fragments for, 483–488
swiping between tabs, 489–493
tasks, 78
testing
automated testing, 870
emulator compared to device, 117
on-device tests, 871
unit tests, 870
text attribute
Button, 44, 203
CheckBox, 206
TextView, 33, 34, 50, 201
text field. *See* EditText element
textOff attribute
Switch, 205
ToggleButton, 204
textOn attribute
Switch, 205
ToggleButton, 204
textSize attribute, TextView, 201
TextView element
about, 33, 44, 201
code for, 33–34, 44–47, 91
setting text in, 64
theme attribute
AppBarLayout, 519
application, 299, 300
themes
about, 84
app bar requiring, 293
applying to project, 299–300
built-in, list of, 302
customizing, 303
for navigation drawers, 589–590
removing app bar using, 308
v7 AppCompat Library for, 294, 296–298
threads
about, 720
background thread, 720–731, 736
main thread, 127, 720
render thread, 720
title attribute, 322
toasts, 216
ToggleButton element, 204
toolbar
adding as layout, 311–313, 316
collapsing, 507, 517–525
for navigation drawers, 589
replacing app bar with, 292, 306–313
scrolling, 507, 508–515
Toolbar class, 306, 309, 314
top-level activities, 249–250, 252–255, 290
transactions, for fragments. *See* fragment transactions

U

unbindService() method, ServiceConnection, 779
unit tests, 870

Up button, 292, 327–330
update() method, SQLiteDatabase, 646, 698
USB debugging, 109
USB driver, installing, 109
uses-permission element, 791

V

v4 Support Library, 294. *See also* Design Support Library
v7 AppCompat Library
  about, 294
  adding to project, 296, 307
  AppCompatActivity class in, 297–298
  fragments using, 345
  Location Services using, 790
  navigation drawers using, 584
  notifications in, 756
v7 CardView Library, 294, 542
v7 RecyclerView Library, 294
values resource folders. *See* string resources; dimension resources
variables, setting, 126, 127
vertical constraints, 228
view animations, 868
ViewGroup class, 198, 200
ViewHolder class, 546, 548
ViewPager class, 489–493, 501
views. *See also* specific GUI components
  about, 44, 198–199
  aligning, 238
  biases for, 231
  centering, 230
  getting and setting properties, 199
  gravity for view contents, 182–183
  gravity for view placement, 184–185, 190
  height, 175, 180, 232–233
  ID, 44, 175, 199
  margins, 176, 228–229
  methods associated with, 199
  weight, 179–181
  width, 175, 232–233

W

website resources
  activity actions, types of, 101
  source code for examples, xxxvi
  USB drivers, 109
WebView class, 866
widgets. *See also* GUI components
  about, 869
  adding, in blueprint tool, 226
  constraints for. *See* constraint layout
Workout app, 340–392, 394–432, 434–480
  activities, 346–347, 359–363, 381, 389, 421–423, 470
  adapters, 375–376
  back button, 413–415
  device sizes, supporting, 340–341, 394–398
  dynamic fragments, 434–436, 444–460
  fragment lifecycle, 439
  fragments, 342–344, 348–356, 359–363, 365–369, 416–421
  fragment state, 427–431
  fragment transactions, 463–469, 472–475
  Java classes, 360
  layout, 471
  libraries, 345
  listener interface, 384–388
  list fragments, 372–374, 377–378
  screen-specific resources, 402–409
  tablet AVD, 399–401
wrap_content setting, width and height, 232

X

xmlns:android attribute, LinearLayout, 171

Z

zipalign tool, 845
Zygote process, 846

Get up to speed on Agile.

Quickly learn what Agile development is all about on Safari, O'Reilly's online learning platform. Through books, videos, interactive tutorials, and live online training, you'll learn how to design and build products incrementally, test constantly, and release as often as you need to.

Check out *The Agile Sketchpad*, a video course by David and Dawn Griffiths that helps you:

- See what it's like to work on an Agile team.
- Discover how Agile helps you deliver value early, respond to change, and manage risk.
- Master the secrets of accurate estimation.
- Explore how test-driven development, continuous integration, and the 10-minute build help you deliver better software.
- Learn why programming in a pair can be more effective than programming alone.
- Find out how Agile techniques such as Kanban can help you identify and avoid disruptions to your development process.
- See how the five key values—communication, courage, feedback, simplicity, and respect—drive everything you do in an Agile project.

Try Safari for free for 10 days and get up to speed with Agile.

oreilly.com/go/agile-sketchpad

O'REILLY®
Safari

D3610

PERCEPTIONS OF A RENEGADE MIND

DAVID ICKE

PERCEPTIONS OF A RENEGADE MIND

DAVID ICKE

PERCEPTIONS OF A RENEGADE MIND

ickonic
publishing

First published in July 2021.

New Enterprise House
St Helens Street
Derby
DE1 3GY
UK

email: gareth.icke@davidicke.com

Copyright © 2021 David Icke

No part of this book may be reproduced in any form without permission from the Publisher, except for the quotation of brief passages in criticism

Cover Design: Gareth Icke
Book Design: Neil Hague

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

eISBN 978-18384153-1-0

PERCEPTIONS OF A RENEGADE MIND

DAVID ICKE

Dedication:

To *Freeeeeedom!*

ICKONIC
THE ALTERNATIVE

NEW. DIFFERENT. REVOLUTIONARY

HUNDREDS OF CUTTING EDGE DOCUMENTARIES,
FEATURE FILMS, SERIES & PODCASTS.

SIGN UP NOW AT ICKONIC.COM

THE LIFE STORY OF DAVID ICKE

RENEGADE

THE FEATURE LENGTH FILM

AVAILABLE NOW AT DAVIDICKE.COM

Renegade:
Adjective
'Having rejected tradition: Unconventional.'
Merriam-Webster Dictionary

Acquiescence to tyranny is the death of the spirit

You may be 38 years old, as I happen to be. And one day, some great opportunity stands before you and calls you to stand up for some great principle, some great issue, some great cause. And you refuse to do it because you are afraid … You refuse to do it because you want to live longer … You're afraid that you will lose your job, or you are afraid that you will be criticised or that you will lose your popularity, or you're afraid that somebody will stab you, or shoot at you or bomb your house; so you refuse to take the stand.

Well, you may go on and live until you are 90, but you're just as dead at 38 as you would be at 90. And the cessation of breathing in your life is but the belated announcement of an earlier death of the spirit.

Martin Luther King

How the few control the many and always have – the many do whatever they're told

'Forward, the Light Brigade!'
Was there a man dismayed?
Not though the soldier knew
Someone had blundered.
Theirs not to make reply,
Theirs not to reason why,
Theirs but to do and die.
Into the valley of Death
Rode the six hundred.

Cannon to right of them,
Cannon to left of them,
Cannon in front of them
Volleyed and thundered;
Stormed at with shot and shell,
Boldly they rode and well,
Into the jaws of Death,
Into the mouth of hell
Rode the six hundred

Alfred Lord Tennyson (1809-1892)

The mist is lifting slowly
I can see the way ahead
And I've left behind the empty streets
That once inspired my life
And the strength of the emotion
Is like thunder in the air
'Cos the promise that we made each other
Haunts me to the end

The secret of your beauty
And the mystery of your soul
I've been searching for in everyone I meet
And the times I've been mistaken
It's impossible to say
And the grass is growing
Underneath our feet

The words that I remember
From my childhood still are true
That there's none so blind
As those who will not see
And to those who lack the courage
And say it's dangerous to try
Well they just don't know
That love eternal will not be denied

I know you're out there somewhere
Somewhere, somewhere
I know you're out there somewhere

Somewhere you can hear my voice
I know I'll find you somehow
Somehow, somehow
I know I'll find you somehow
And somehow I'll return again to you

The Moody Blues

Are you a gutless wonder - or a Renegade Mind?

Monuments put from pen to paper,
Turns me into a gutless wonder,
And if you tolerate this,
Then your children will be next.
Gravity keeps my head down,
Or is it maybe shame ...

Manic Street Preachers

Rise like lions after slumber
In unvanquishable number.
Shake your chains to earth like dew
Which in sleep have fallen on you.
Ye are many – they are few.

Percy Shelley

Contents

CHAPTER 1 'I'm thinking' – Oh, but *are* you?
CHAPTER 2 Renegade perception
CHAPTER 3 The Pushbacker sting
CHAPTER 4 'Covid': The calculated catastrophe
CHAPTER 5 There *is no* 'virus'
CHAPTER 6 Sequence of deceit
CHAPTER 7 War on your mind
CHAPTER 8 'Reframing' insanity
CHAPTER 9 We must have it? So what is it?
CHAPTER 10 Human 2.0
CHAPTER 11 Who controls the Cult?
CHAPTER 12 Escaping Wetiko

POSTSCRIPT
APPENDIX Cowan-Kaufman-Morell Statement on Virus Isolation
BIBLIOGRAPHY
INDEX

CHAPTER ONE

I'm thinking' – Oh, but *are* you?

Think for yourself and let others enjoy the privilege of doing so too
Voltaire

French-born philosopher, mathematician and scientist René Descartes became famous for his statement in Latin in the 17th century which translates into English as: 'I think, therefore I am.'

On the face of it that is true. Thought reflects perception and perception leads to both behaviour and self-identity. In that sense 'we' are what we think. But who or what is doing the thinking and is thinking the only route to perception? Clearly, as we shall see, 'we' are not always the source of 'our' perception, indeed with regard to humanity as a whole this is rarely the case; and thinking is far from the only means of perception. Thought is the village idiot compared with other expressions of consciousness that we all have the potential to access and tap into. This has to be true when we *are* those other expressions of consciousness which are infinite in nature. We have forgotten this, or, more to the point, been manipulated to forget.

These are not just the esoteric musings of the navel. The whole foundation of human control and oppression is control of perception. Once perception is hijacked then so is behaviour which is dictated by perception. Collective perception becomes collective behaviour and collective behaviour is what we call human society. Perception is all and those behind human control know that which is

why perception is the target 24/7 of the psychopathic manipulators that I call the Global Cult. They know that if they dictate perception they will dictate behaviour and collectively dictate the nature of human society. They are further aware that perception is formed from information received and if they control the circulation of information they will to a vast extent direct human behaviour. Censorship of information and opinion has become globally Nazi-like in recent years and never more blatantly than since the illusory 'virus pandemic' was triggered out of China in 2019 and across the world in 2020. Why have billions submitted to house arrest and accepted fascistic societies in a way they would have never believed possible? Those controlling the information spewing from government, mainstream media and Silicon Valley (all controlled by the same Global Cult networks) told them they were in danger from a 'deadly virus' and only by submitting to house arrest and conceding their most basic of freedoms could they and their families be protected. This monumental and provable lie became the *perception* of the billions and therefore the *behaviour* of the billions. In those few words you have the whole structure and modus operandi of human control. Fear is a perception – **F**alse **E**motion **A**ppearing **R**eal – and fear is the currency of control. In short … get them by the balls (or give them the impression that you have) and their hearts and minds will follow. Nothing grips the dangly bits and freezes the rear-end more comprehensively than fear.

World number 1

There are two 'worlds' in what appears to be one 'world' and the prime difference between them is knowledge. First we have the mass of human society in which the population is maintained in coldly-calculated ignorance through control of information and the 'education' (indoctrination) system. That's all you really need to control to enslave billions in a perceptual delusion in which what are perceived to be *their* thoughts and opinions are ever-repeated mantras that the system has been downloading all their lives through 'education', media, science, medicine, politics and academia

in which the personnel and advocates are themselves overwhelmingly the perceptual products of the same repetition. Teachers and academics in general are processed by the same programming machine as everyone else, but unlike the great majority they never leave the 'education' program. It gripped them as students and continues to grip them as programmers of subsequent generations of students. The programmed become the programmers – the programmed programmers. The same can largely be said for scientists, doctors and politicians and not least because as the American writer Upton Sinclair said: 'It is difficult to get a man to understand something when his salary depends upon his not understanding it.' If your career and income depend on thinking the way the system demands then you will – bar a few free-minded exceptions – concede your mind to the Perceptual Mainframe that I call the Postage Stamp Consensus. This is a tiny band of perceived knowledge and possibility 'taught' (downloaded) in the schools and universities, pounded out by the mainstream media and on which all government policy is founded. Try thinking, and especially speaking and acting, outside of the 'box' of consensus and see what that does for your career in the Mainstream Everything which bullies, harasses, intimidates and ridicules the population into compliance. Here we have the simple structure which enslaves most of humanity in a perceptual prison cell for an entire lifetime and I'll go deeper into this process shortly. Most of what humanity is taught as fact is nothing more than programmed belief. American science fiction author Frank Herbert was right when he said: 'Belief can be manipulated. Only knowledge is dangerous.' In the 'Covid' age belief is promoted and knowledge is censored. It was always so, but never to the extreme of today.

World number 2

A 'number 2' is slang for 'doing a poo' and how appropriate that is when this other 'world' is doing just that on humanity every minute of every day. World number 2 is a global network of secret societies and semi-secret groups dictating the direction of society via

governments, corporations and authorities of every kind. I have spent more than 30 years uncovering and exposing this network that I call the Global Cult and knowing its agenda is what has made my books so accurate in predicting current and past events. Secret societies are secret for a reason. They want to keep their hoarded knowledge to themselves and their chosen initiates and to hide it from the population which they seek through ignorance to control and subdue. The whole foundation of the division between World 1 and World 2 is *knowledge*. What number 1 knows number 2 must not. Knowledge they have worked so hard to keep secret includes (a) the agenda to enslave humanity in a centrally-controlled global dictatorship, and (b) the nature of reality and life itself. The latter (b) must be suppressed to allow the former (a) to prevail as I shall be explaining. The way the Cult manipulates and interacts with the population can be likened to a spider's web. The 'spider' sits at the centre in the shadows and imposes its will through the web with each strand represented in World number 2 by a secret society, satanic or semi-secret group, and in World number 1 – the world of the seen – by governments, agencies of government, law enforcement, corporations, the banking system, media conglomerates and Silicon Valley (Fig 1 overleaf). The spider and the web connect and coordinate all these organisations to pursue the same global outcome while the population sees them as individual entities working randomly and independently. At the level of the web governments *are* the banking system *are* the corporations *are* the media *are* Silicon Valley *are* the World Health Organization working from their inner cores as one unit. Apparently unconnected countries, corporations, institutions, organisations and people are on the *same team* pursuing the same global outcome. Strands in the web immediately around the spider are the most secretive and exclusive secret societies and their membership is emphatically restricted to the Cult inner-circle emerging through the generations from particular bloodlines for reasons I will come to. At the core of the core you would get them in a single room. That's how many people are dictating the direction of human society and its transformation

through the 'Covid' hoax and other means. As the web expands out from the spider we meet the secret societies that many people will be aware of – the Freemasons, Knights Templar, Knights of Malta, Opus Dei, the inner sanctum of the Jesuit Order, and such like. Note how many are connected to the Church of Rome and there is a reason for that. The Roman Church was established as a revamp, a rebranding, of the relocated 'Church' of Babylon and the Cult imposing global tyranny today can be tracked back to Babylon and Sumer in what is now Iraq.

Figure 1: The global web through which the few control the many. (Image Neil Hague.)

Inner levels of the web operate in the unseen away from the public eye and then we have what I call the cusp organisations located at the point where the hidden meets the seen. They include a series of satellite organisations answering to a secret society founded in London in the late 19th century called the Round Table and among them are the Royal Institute of International Affairs (UK, founded in 1920); Council on Foreign Relations (US, 1921); Bilderberg Group (worldwide, 1954); Trilateral Commission (US/worldwide, 1972); and the Club of Rome (worldwide, 1968) which was created to exploit environmental concerns to justify the centralisation of global power to 'save the planet'. The Club of Rome instigated with others the human-caused climate change hoax which has led to all the 'green

new deals' demanding that very centralisation of control. Cusp organisations, which include endless 'think tanks' all over the world, are designed to coordinate a single global policy between political and business leaders, intelligence personnel, media organisations and anyone who can influence the direction of policy in their own sphere of operation. Major players and regular attenders will know what is happening – or some of it – while others come and go and are kept overwhelmingly in the dark about the big picture. I refer to these cusp groupings as semi-secret in that they can be publicly identified, but what goes on at the inner-core is kept very much 'in house' even from most of their members and participants through a fiercely-imposed system of compartmentalisation. Only let them know what they need to know to serve your interests and no more. The structure of secret societies serves as a perfect example of this principle. Most Freemasons never get higher than the bottom three levels of 'degree' (degree of knowledge) when there are 33 official degrees of the Scottish Rite. Initiates only qualify for the next higher 'compartment' or degree if those at that level choose to allow them. Knowledge can be carefully assigned only to those considered 'safe'. I went to my local Freemason's lodge a few years ago when they were having an 'open day' to show how cuddly they were and when I chatted to some of them I was astonished at how little the rank and file knew even about the most ubiquitous symbols they use. The mushroom technique – keep them in the dark and feed them bullshit – applies to most people in the web as well as the population as a whole. Sub-divisions of the web mirror in theme and structure transnational corporations which have a headquarters somewhere in the world dictating to all their subsidiaries in different countries. Subsidiaries operate in their methodology and branding to the same centrally-dictated plan and policy in pursuit of particular ends. The Cult web functions in the same way. Each country has its own web as a subsidiary of the global one. They consist of networks of secret societies, semi-secret groups and bloodline families and their job is to impose the will of the spider and the global web in their particular country. Subsidiary networks control and manipulate the national political system, finance, corporations, media, medicine, etc. to

ensure that they follow the globally-dictated Cult agenda. These networks were the means through which the 'Covid' hoax could be played out with almost every country responding in the same way.

The 'Yessir' pyramid

Compartmentalisation is the key to understanding how a tiny few can dictate the lives of billions when combined with a top-down sequence of imposition and acquiescence. The inner core of the Cult sits at the peak of the pyramidal hierarchy of human society (Fig 2 overleaf). It imposes its will – its agenda for the world – on the level immediately below which acquiesces to that imposition. This level then imposes the Cult will on the level below them which acquiesces and imposes on the next level. Very quickly we meet levels in the hierarchy that have no idea there even is a Cult, but the sequence of imposition and acquiescence continues down the pyramid in just the same way. 'I don't know why we are doing this but the order came from "on-high" and so we better just do it.' Alfred Lord Tennyson said of the cannon fodder levels in his poem *The Charge of the Light Brigade*: 'Theirs not to reason why; theirs but to do and die.' The next line says that 'into the valley of death rode the six hundred' and they died because they obeyed without question what their perceived 'superiors' told them to do. In the same way the population capitulated to 'Covid'. The whole hierarchical pyramid functions like this to allow the very few to direct the enormous many. Eventually imposition-acquiescence-imposition-acquiescence comes down to the mass of the population at the foot of the pyramid. If they acquiesce to those levels of the hierarchy imposing on them (governments/law enforcement/doctors/media) a circuit is completed between the population and the handful of super-psychopaths in the Cult inner core at the top of the pyramid. Without a circuit-breaking refusal to obey, the sequence of imposition and acquiescence allows a staggeringly few people to impose their will upon the entirety of humankind. We are looking at the very sequence that has subjugated billions since the start of 2020. Our freedom has not been taken from us. Humanity has given it

away. Fascists do not impose fascism because there are not enough of them. Fascism is imposed by the population acquiescing to fascism. Put another way allowing their perceptions to be programmed to the extent that leads to the population giving their freedom away by giving their perceptions – their mind – away. If this circuit is not broken by humanity ceasing to cooperate with their own enslavement then nothing can change. For that to happen people have to critically think and see through the lies and window dressing and then summon the backbone to act upon what they see. The Cult spends its days working to stop either happening and its methodology is systematic and highly detailed, but it can be overcome and that is what this book is all about.

Figure 2: The simple sequence of imposition and compliance that allows a handful of people at the peak of the pyramid to dictate the lives of billions.

The Life Program

Okay, back to world number 1 or the world of the 'masses'. Observe the process of what we call 'life' and it is a perceptual download from cradle to grave. The Cult has created a global structure in which perception can be programmed and the program continually topped-up with what appears to be constant confirmation that the program is indeed true reality. The important word here is 'appears'.

This is the structure, the fly-trap, the Postage Stamp Consensus or Perceptual Mainframe, which represents that incredibly narrow band of perceived possibility delivered by the 'education' system, mainstream media, science and medicine. From the earliest age the download begins with parents who have themselves succumbed to the very programming their children are about to go through. Most parents don't do this out of malevolence and mostly it is quite the opposite. They do what they believe is best for their children and that is what the program has told them is best. Within three or four years comes the major transition from parental programming to full-blown state (Cult) programming in school, college and university where perceptually-programmed teachers and academics pass on their programming to the next generations. Teachers who resist are soon marginalised and their careers ended while children who resist are called a problem child for whom Ritalin may need to be prescribed. A few years after entering the 'world' children are under the control of authority figures representing the state telling them when they have to be there, when they can leave and when they can speak, eat, even go to the toilet. This is calculated preparation for a lifetime of obeying authority in all its forms. Reflex-action fear of authority is instilled by authority from the start. Children soon learn the carrot and stick consequences of obeying or defying authority which is underpinned daily for the rest of their life. Fortunately I daydreamed through this crap and never obeyed authority simply because it told me to. This approach to my alleged 'betters' continues to this day. There can be consequences of pursuing open-minded freedom in a world of closed-minded conformity. I spent a lot of time in school corridors after being ejected from the classroom for not taking some of it seriously and now I spend a lot of time being ejected from Facebook, YouTube and Twitter. But I can tell you that being true to yourself and not compromising your self-respect is far more exhilarating than bowing to authority for authority's sake. You don't have to be a sheep to the shepherd (authority) and the sheep dog (fear of not obeying authority).

The perceptual download continues throughout the formative years in school, college and university while script-reading 'teachers', 'academics' 'scientists', 'doctors' and 'journalists' insist that ongoing generations must be as programmed as they are. Accept the program or you will not pass your 'exams' which confirm your 'degree' of programming. It is tragic to think that many parents pressure their offspring to work hard at school to download the program and qualify for the next stage at college and university. The late, great, American comedian George Carlin said: 'Here's a bumper sticker I'd like to see: We are proud parents of a child who has resisted his teachers' attempts to break his spirit and bend him to the will of his corporate masters.' Well, the best of luck finding many of those, George. Then comes the moment to leave the formal programming years in academia and enter the 'adult' world of work. There you meet others in your chosen or prescribed arena who went through the same Postage Stamp Consensus program before you did. There is therefore overwhelming agreement between almost everyone on the basic foundations of Postage Stamp reality and the rejection, even contempt, of the few who have a mind of their own and are prepared to use it. This has two major effects. Firstly, the consensus confirms to the programmed that their download is really how things are. I mean, everyone knows that, right? Secondly, the arrogance and ignorance of Postage Stamp adherents ensure that anyone questioning the program will have unpleasant consequences for seeking their own truth and not picking their perceptions from the shelf marked: 'Things you must believe without question and if you don't you're a dangerous lunatic conspiracy theorist and a harebrained nutter'.

Every government, agency and corporation is founded on the same Postage Stamp prison cell and you can see why so many people believe the same thing while calling it their own 'opinion'. Fusion of governments and corporations in pursuit of the same agenda was the definition of fascism described by Italian dictator Benito Mussolini. The pressure to conform to perceptual norms downloaded for a lifetime is incessant and infiltrates society right

down to family groups that become censors and condemners of their own 'black sheep' for not, ironically, being sheep. We have seen an explosion of that in the 'Covid' era. Cult-owned global media unleashes its propaganda all day every day in support of the Postage Stamp and targets with abuse and ridicule anyone in the public eye who won't bend their mind to the will of the tyranny. Any response to this is denied (certainly in my case). They don't want to give a platform to expose official lies. Cult-owned-and-created Internet giants like Facebook, Google, YouTube and Twitter delete you for having an unapproved opinion. Facebook boasts that its AI censors delete 97-percent of 'hate speech' before anyone even reports it. Much of that 'hate speech' will simply be an opinion that Facebook and its masters don't want people to see. Such perceptual oppression is widely known as fascism. Even Facebook executive Benny Thomas, a 'CEO Global Planning Lead', said in comments secretly recorded by investigative journalism operation Project Veritas that Facebook is 'too powerful' and should be broken up:

> I mean, no king in history has been the ruler of two billion people, but Mark Zuckerberg is … And he's 36. That's too much for a 36-year-old ... You should not have power over two billion people. I just think that's wrong.

Thomas said Facebook-owned platforms like Instagram, Oculus, and WhatsApp needed to be separate companies. 'It's too much power when they're all one together'. That's the way the Cult likes it, however. We have an executive of a Cult organisation in Benny Thomas that doesn't know there is a Cult such is the compartmentalisation. Thomas said that Facebook and Google 'are no longer companies, they're countries'. Actually they are more powerful than countries on the basis that if you control information you control perception and control human society.

I love my oppressor

Another expression of this psychological trickery is for those who realise they are being pressured into compliance to eventually

convince themselves to believe the official narratives to protect their self-respect from accepting the truth that they have succumbed to meek and subservient compliance. Such people become some of the most vehement defenders of the system. You can see them everywhere screaming abuse at those who prefer to think for themselves and by doing so reminding the compliers of their own capitulation to conformity. 'You are talking dangerous nonsense you Covidiot!!' Are you trying to convince me or yourself? It is a potent form of Stockholm syndrome which is defined as: 'A psychological condition that occurs when a victim of abuse identifies and attaches, or bonds, positively with their abuser.' An example is hostages bonding and even 'falling in love' with their kidnappers. The syndrome has been observed in domestic violence, abused children, concentration camp inmates, prisoners of war and many and various Satanic cults. These are some traits of Stockholm syndrome listed at goodtherapy.org:

- Positive regard towards perpetrators of abuse or captor [see 'Covid'].
- Failure to cooperate with police and other government authorities when it comes to holding perpetrators of abuse or kidnapping accountable [or in the case of 'Covid' cooperating with the police to enforce and defend their captors' demands].
- Little or no effort to escape [see 'Covid'].
- Belief in the goodness of the perpetrators or kidnappers [see 'Covid'].
- Appeasement of captors. This is a manipulative strategy for maintaining one's safety. As victims get rewarded – perhaps with less abuse or even with life itself – their appeasing behaviours are reinforced [see 'Covid'].
- Learned helplessness. This can be akin to 'if you can't beat 'em, join 'em'. As the victims fail to escape the abuse or captivity, they may start giving up and soon realize it's just easier for everyone if they acquiesce all their power to their captors [see 'Covid'].

- Feelings of pity toward the abusers, believing they are actually victims themselves. Because of this, victims may go on a crusade or mission to 'save' [protect] their abuser [see the venom unleashed on those challenging the official 'Covid' narrative].
- Unwillingness to learn to detach from their perpetrators and heal. In essence, victims may tend to be less loyal to themselves than to their abuser [*definitely* see 'Covid'].

Ponder on those traits and compare them with the behaviour of great swathes of the global population who have defended governments and authorities which have spent every minute destroying their lives and livelihoods and those of their children and grandchildren since early 2020 with fascistic lockdowns, house arrest and employment deletion to 'protect' them from a 'deadly virus' that their abusers' perceptually created to bring about this very outcome. We are looking at mass Stockholm syndrome. All those that agree to concede their freedom will believe those perceptions are originating in their own independent 'mind' when in fact by conceding their reality to Stockholm syndrome they have by definition conceded any independence of mind. Listen to the 'opinions' of the acquiescing masses in this 'Covid' era and what gushes forth is the repetition of the official version of everything delivered unprocessed, unfiltered and unquestioned. The whole programming dynamic works this way. I must be free because I'm told that I am and so I think that I am.

You can see what I mean with the chapter theme of 'I'm thinking – Oh, but *are* you?' The great majority are not thinking, let alone for themselves. They are repeating what authority has told them to believe which allows them to be controlled. Weaving through this mentality is the fear that the 'conspiracy theorists' are right and this again explains the often hysterical abuse that ensues when you dare to contest the official narrative of anything. Denial is the mechanism of hiding from yourself what you don't want to be true. Telling people what they want to hear is easy, but it's an infinitely greater challenge to tell them what they would rather not be happening.

One is akin to pushing against an open door while the other is met with vehement resistance no matter what the scale of evidence. I don't want it to be true so I'll convince myself that it's not. Examples are everywhere from the denial that a partner is cheating despite all the signs to the reflex-action rejection of any idea that world events in which country after country act in exactly the same way are centrally coordinated. To accept the latter is to accept that a force of unspeakable evil is working to destroy your life and the lives of your children with nothing too horrific to achieve that end. Who the heck wants that to be true? But if we don't face reality the end is duly achieved and the consequences are far worse and ongoing than breaking through the walls of denial today with the courage to make a stand against tyranny.

Connect the dots – but how?

A crucial aspect of perceptual programming is to portray a world in which everything is random and almost nothing is connected to anything else. Randomness cannot be coordinated by its very nature and once you perceive events as random the idea they could be connected is waved away as the rantings of the tinfoil-hat brigade. You can't plan and coordinate random you idiot! No, you can't, but you can hide the coldly-calculated and long-planned behind the *illusion* of randomness. A foundation manifestation of the Renegade Mind is to scan reality for patterns that connect the apparently random and turn pixels and dots into pictures. This is the way I work and have done so for more than 30 years. You look for similarities in people, modus operandi and desired outcomes and slowly, then ever quicker, the picture forms. For instance: There would seem to be no connection between the 'Covid pandemic' hoax and the human-caused global-warming hoax and yet they are masks (appropriately) on the same face seeking the same outcome. Those pushing the global warming myth through the Club of Rome and other Cult agencies are driving the lies about 'Covid' – Bill Gates is an obvious one, but they are endless. Why would the same people be involved in both when they are clearly not connected? Oh, but they

are. Common themes with personnel are matched by common goals. The 'solutions' to both 'problems' are centralisation of global power to impose the will of the few on the many to 'save' humanity from 'Covid' and save the planet from an 'existential threat' (we need 'zero Covid' and 'zero carbon emissions'). These, in turn, connect with the 'dot' of globalisation which was coined to describe the centralisation of global power in every area of life through incessant political and corporate expansion, trading blocks and superstates like the European Union. If you are the few and you want to control the many you have to centralise power and decision-making. The more you centralise power the more power the few at the centre will have over the many; and the more that power is centralised the more power those at the centre have to centralise even quicker. The momentum of centralisation gets faster and faster which is exactly the process we have witnessed. In this way the hoaxed 'pandemic' and the fakery of human-caused global warming serve the interests of globalisation and the seizure of global power in the hands of the Cult inner-circle which is behind 'Covid', 'climate change' *and* globalisation. At this point random 'dots' become a clear and obvious picture or pattern.

Klaus Schwab, the classic Bond villain who founded the Cult's Gates-funded World Economic Forum, published a book in 2020, *The Great Reset*, in which he used the 'problem' of 'Covid' to justify a total transformation of human society to 'save' humanity from 'climate change'. Schwab said: 'The pandemic represents a rare but narrow window of opportunity to reflect, reimagine, and reset our world.' What he didn't mention is that the Cult he serves is behind both hoaxes as I show in my book *The Answer*. He and the Cult don't have to reimagine the world. They know precisely what they want and that's why they destroyed human society with 'Covid' to 'build back better' in their grand design. Their job is not to imagine, but to get humanity to imagine and agree with their plans while believing it's all random. It must be pure coincidence that 'The Great Reset' has long been the Cult's code name for the global imposition of fascism and replaced previous code-names of the 'New World

Order' used by Cult frontmen like Father George Bush and the 'New Order of the Ages' which emerged from Freemasonry and much older secret societies. New Order of the Ages appears on the reverse of the Great Seal of the United States as 'Novus ordo seclorum' underneath the Cult symbol used since way back of the pyramid and all seeing-eye (Fig 3). The pyramid is the hierarchy of human control headed by the illuminated eye that symbolises the force behind the Cult which I will expose in later chapters. The term 'Annuit Coeptis' translates as 'He favours our undertaking'. We are told the 'He' is the Christian god, but 'He' is not as I will be explaining.

Figure 3: The all-seeing eye of the Cult 'god' on the Freemason-designed Great Seal of the United States and also on the dollar bill.

Having you on

Two major Cult techniques of perceptual manipulation that relate to all this are what I have called since the 1990s Problem-Reaction-Solution (PRS) and the Totalitarian Tiptoe (TT). They can be uncovered by the inquiring mind with a simple question: Who benefits? The answer usually identifies the perpetrators of a given action or happening through the concept of 'he who most benefits from a crime is the one most likely to have committed it'. The Latin 'Cue bono?' – Who benefits? – is widely attributed to the Roman orator and statesman Marcus Tullius Cicero. No wonder it goes back so far when the concept has been relevant to human behaviour since

history was recorded. Problem-Reaction-Solution is the technique used to manipulate us every day by covertly creating a problem (or the illusion of one) and offering the solution to the problem (or the illusion of one). In the first phase you create the problem and blame someone or something else for why it has happened. This may relate to a financial collapse, terrorist attack, war, global warming or pandemic, anything in fact that will allow you to impose the 'solution' to change society in the way you desire at that time. The 'problem' doesn't have to be real. PRS is manipulation of perception and all you need is the population to believe the problem is real. Human-caused global warming and the 'Covid pandemic' only have to be *perceived* to be real for the population to accept the 'solutions' of authority. I refer to this technique as NO-Problem-Reaction-Solution. Billions did not meekly accept house arrest from early 2020 because there was a real deadly 'Covid pandemic' but because they perceived – believed – that to be the case. The antidote to Problem-Reaction-Solution is to ask who benefits from the proposed solution. Invariably it will be anyone who wants to justify more control through deletion of freedom and centralisation of power and decision-making.

The two world wars were Problem-Reaction-Solutions that transformed and realigned global society. Both were manipulated into being by the Cult as I have detailed in books since the mid-1990s. They dramatically centralised global power, especially World War Two, which led to the United Nations and other global bodies thanks to the overt and covert manipulations of the Rockefeller family and other Cult bloodlines like the Rothschilds. The UN is a stalking horse for full-blown world government that I will come to shortly. The land on which the UN building stands in New York was donated by the Rockefellers and the same Cult family was behind Big Pharma scalpel and drug 'medicine' and the creation of the World Health Organization as part of the UN. They have been stalwarts of the eugenics movement and funded Hitler's race-purity expert' Ernst Rudin. The human-caused global warming hoax has been orchestrated by the Club of Rome through the UN which is

manufacturing both the 'problem' through its Intergovernmental Panel on Climate Change and imposing the 'solution' through its Agenda 21 and Agenda 2030 which demand the total centralisation of global power to 'save the world' from a climate hoax the United Nations is itself perpetrating. What a small world the Cult can be seen to be particularly among the inner circles. The bedfellow of Problem-Reaction-Solution is the Totalitarian Tiptoe which became the Totalitarian Sprint in 2020. The technique is fashioned to hide the carefully-coordinated behind the cover of apparently random events. You start the sequence at 'A' and you know you are heading for 'Z'. You don't want people to know that and each step on the journey is presented as a random happening while all the steps strung together lead in the same direction. The speed may have quickened dramatically in recent times, but you can still see the incremental approach of the Tiptoe in the case of 'Covid' as each new imposition takes us deeper into fascism. Tell people they have to do this or that to get back to 'normal', then this and this and this. With each new demand adding to the ones that went before the population's freedom is deleted until it disappears. The spider wraps its web around the flies more comprehensively with each new diktat. I'll highlight this in more detail when I get to the 'Covid' hoax and how it has been pulled off. Another prime example of the Totalitarian Tiptoe is how the Cult-created European Union went from a 'free-trade zone' to a centralised bureaucratic dictatorship through the Tiptoe of incremental centralisation of power until nations became mere administrative units for Cult-owned dark suits in Brussels.

The antidote to ignorance is knowledge which the Cult seeks vehemently to deny us, but despite the systematic censorship to that end the Renegade Mind can overcome this by vociferously seeking out the facts no matter the impediments put in the way. There is also a method of thinking and perceiving – *knowing* – that doesn't even need names, dates, place-type facts to identify the patterns that reveal the story. I'll get to that in the final chapter. All you need to know about the manipulation of human society and to what end is still out there – *at the time of writing* – in the form of books, videos

and websites for those that really want to breach the walls of programmed perception. To access this knowledge requires the abandonment of the mainstream media as a source of information in the awareness that this is owned and controlled by the Cult and therefore promotes mass perceptions that suit the Cult. Mainstream media lies all day, every day. That is its function and very reason for being. Where it does tell the truth, here and there, is only because the truth and the Cult agenda very occasionally coincide. If you look for fact and insight to the BBC, CNN and virtually all the rest of them you are asking to be conned and perceptually programmed.

Know the outcome and you'll see the journey

Events seem random when you have no idea where the world is being taken. Once you do the random becomes the carefully planned. Know the outcome and you'll see the journey is a phrase I have been using for a long time to give context to daily happenings that appear unconnected. Does a problem, or illusion of a problem, trigger a proposed 'solution' that further drives society in the direction of the outcome? Invariably the answer will be yes and the random – *abracadabra* – becomes the clearly coordinated. So what is this outcome that unlocks the door to a massively expanded understanding of daily events? I will summarise its major aspects – the fine detail is in my other books – and those new to this information will see that the world they thought they were living in is a very different place. The foundation of the Cult agenda is the incessant centralisation of power and all such centralisation is ultimately in pursuit of Cult control on a global level. I have described for a long time the planned world structure of top-down dictatorship as the Hunger Games Society. The term obviously comes from the movie series which portrayed a world in which a few living in military-protected hi-tech luxury were the overlords of a population condemned to abject poverty in isolated 'sectors' that were not allowed to interact. 'Covid' lockdowns and travel bans anyone? The 'Hunger Games' pyramid of structural control has the inner circle of the Cult at the top with pretty much the entire

population at the bottom under their control through dependency for survival on the Cult. The whole structure is planned to be protected and enforced by a military-police state (Fig 4).

Here you have the reason for the global lockdowns of the fake pandemic to coldly destroy independent incomes and livelihoods and make everyone dependent on the 'state' (the Cult that controls the 'states'). I have warned in my books for many years about the plan to introduce a 'guaranteed income' – a barely survivable pittance – designed to impose dependency when employment was destroyed by AI technology and now even more comprehensively at great speed by the 'Covid' scam. Once the pandemic was played and lockdown consequences began to delete independent income the authorities began to talk right on cue about the need for a guaranteed income and a 'Great Reset'. Guaranteed income will be presented as benevolent governments seeking to help a desperate people – desperate as a direct result of actions of the same governments. The truth is that such payments are a trap. You will only get them if you do exactly what the authorities demand including mass vaccination (genetic manipulation). We have seen this theme already in Australia where those dependent on government benefits have them reduced if parents don't agree to have their children vaccinated according to an insane health-destroying government-dictated schedule. Calculated economic collapse applies to governments as well as people. The Cult wants rid of countries through the creation of a world state with countries broken up into regions ruled by a world government and super states like the European Union. Countries must be bankrupted, too, to this end and it's being achieved by the trillions in 'rescue packages' and furlough payments, trillions in lost taxation, and money-no-object spending on 'Covid' including constant all-medium advertising (programming) which has made the media dependent on government for much of its income. The day of reckoning is coming – as planned – for government spending and given that it has been made possible by printing money and not by production/taxation there is inflation on the way that has the

potential to wipe out monetary value. In that case there will be no need for the Cult to steal your money. It just won't be worth anything (see the German Weimar Republic before the Nazis took over). Many have been okay with lockdowns while getting a percentage of their income from so-called furlough payments without having to work. Those payments are dependent, however, on people having at least a theoretical job with a business considered non-essential and ordered to close. As these business go under because they are closed by lockdown after lockdown the furlough stops and it will for everyone eventually. Then what? The 'then what?' is precisely the idea.

Figure 4: The Hunger Games Society structure I have long warned was planned and now the 'Covid' hoax has made it possible. This is the real reason for lockdowns.

Hired hands

Between the Hunger Games Cult elite and the dependent population is planned to be a vicious military-police state (a fusion of the two into one force). This has been in the making for a long time with police looking ever more like the military and carrying weapons to match. The pandemic scam has seen this process accelerate so fast as

lockdown house arrest is brutally enforced by carefully recruited fascist minds and gormless system-servers. The police and military are planned to merge into a centrally-directed world army in a global structure headed by a world government which wouldn't be elected even by the election fixes now in place. The world army is not planned even to be human and instead wars would be fought, primarily against the population, using robot technology controlled by artificial intelligence. I have been warning about this for decades and now militaries around the world are being transformed by this very AI technology. The global regime that I describe is a particular form of fascism known as a technocracy in which decisions are not made by clueless and co-opted politicians but by unelected technocrats – scientists, engineers, technologists and bureaucrats. Cult-owned-and-controlled Silicon Valley giants are examples of technocracy and they already have far more power to direct world events than governments. They are with their censorship *selecting* governments. I know that some are calling the 'Great Reset' a Marxist communist takeover, but fascism and Marxism are different labels for the same tyranny. Tell those who lived in fascist Germany and Stalinist Russia that there was a difference in the way their freedom was deleted and their lives controlled. I could call it a fascist technocracy or a Marxist technocracy and they would be equally accurate. The Hunger Games society with its world government structure would oversee a world army, world central bank and single world cashless currency imposing its will on a microchipped population (Fig 5). Scan its different elements and see how the illusory pandemic is forcing society in this very direction at great speed. Leaders of 23 countries and the World Health Organization (WHO) backed the idea in March, 2021, of a global treaty for 'international cooperation' in 'health emergencies' and nations should 'come together as a global community for peaceful cooperation that extends beyond this crisis'. Cut the Orwellian bullshit and this means another step towards global government. The plan includes a cashless digital money system that I first warned about in 1993. Right at the start of 'Covid' the deeply corrupt Tedros

Adhanom Ghebreyesus, the crooked and merely gofer 'head' of the World Health Organization, said it was possible to catch the 'virus' by touching cash and it was better to use cashless means. The claim was ridiculous nonsense and like the whole 'Covid' mind-trick it was nothing to do with 'health' and everything to do with pushing every aspect of the Cult agenda. As a result of the Tedros lie the use of cash has plummeted. The Cult script involves a single world digital currency that would eventually be technologically embedded in the body. China is a massive global centre for the Cult and if you watch what is happening there you will know what is planned for everywhere. The Chinese government is developing a digital currency which would allow fines to be deducted immediately via AI for anyone caught on camera breaking its fantastic list of laws and the money is going to be programmable with an expiry date to ensure that no one can accrue wealth except the Cult and its operatives.

Figure 5: The structure of global control the Cult has been working towards for so long and this has been enormously advanced by the 'Covid' illusion.

Serfdom is so smart

The Cult plan is far wider, extreme, and more comprehensive than even most conspiracy researchers appreciate and I will come to the true depths of deceit and control in the chapters 'Who controls the

Cult?' and 'Escaping Wetiko'. Even the world that we know is crazy enough. We are being deluged with ever more sophisticated and controlling technology under the heading of 'smart'. We have smart televisions, smart meters, smart cards, smart cars, smart driving, smart roads, smart pills, smart patches, smart watches, smart skin, smart borders, smart pavements, smart streets, smart cities, smart communities, smart environments, smart growth, smart planet ... smart *everything* around us. Smart technologies and methods of operation are designed to interlock to create a global Smart Grid connecting the entirety of human society including human minds to create a centrally-dictated 'hive' mind. 'Smart cities' is code for densely-occupied megacities of total surveillance and control through AI. Ever more destructive frequency communication systems like 5G have been rolled out without any official testing for health and psychological effects (colossal). 5G/6G/7G systems are needed to run the Smart Grid and each one becomes more destructive of body and mind. Deleting independent income is crucial to forcing people into these AI-policed prisons by ending private property ownership (except for the Cult elite). The Cult's Great Reset now openly foresees a global society in which no one will own any possessions and everything will be rented while the Cult would own literally everything under the guise of government and corporations. The aim has been to use the lockdowns to destroy sources of income on a mass scale and when the people are destitute and in unrepayable amounts of debt (problem) Cult assets come forward with the pledge to write-off debt in return for handing over all property and possessions (solution). Everything – literally everything including people – would be connected to the Internet via AI. I was warning years ago about the coming Internet of Things (IoT) in which all devices and technology from your car to your fridge would be plugged into the Internet and controlled by AI. Now we are already there with much more to come. The next stage is the Internet of Everything (IoE) which is planned to include the connection of AI to the human brain and body to replace the human mind with a centrally-controlled AI mind. Instead of perceptions

being manipulated through control of information and censorship those perceptions would come direct from the Cult through AI. What do you think? You think whatever AI decides that you think. In human terms there would be no individual 'think' any longer. Too incredible? The ravings of a lunatic? Not at all. Cult-owned crazies in Silicon Valley have been telling us the plan for years without explaining the real motivation and calculated implications. These include Google executive and 'futurist' Ray Kurzweil who highlights the year 2030 for when this would be underway. He said:

> Our thinking ... will be a hybrid of biological and non-biological thinking ... humans will be able to extend their limitations and 'think in the cloud' ... We're going to put gateways to the cloud in our brains ... We're going to gradually merge and enhance ourselves ... In my view, that's the nature of being human – we transcend our limitations.
>
> As the technology becomes vastly superior to what we are then the small proportion that is still human gets smaller and smaller and smaller until it's just utterly negligible.

The sales-pitch of Kurzweil and Cult-owned Silicon Valley is that this would make us 'super-human' when the real aim is to make us post-human and no longer 'human' in the sense that we have come to know. The entire global population would be connected to AI and become the centrally-controlled 'hive-mind' of externally-delivered perceptions. The Smart Grid being installed to impose the Cult's will on the world is being constructed to allow particular locations – even one location – to control the whole global system. From these prime control centres, which absolutely include China and Israel, anything connected to the Internet would be switched on or off and manipulated at will. Energy systems could be cut, communication via the Internet taken down, computer-controlled driverless autonomous vehicles driven off the road, medical devices switched off, the potential is limitless given how much AI and Internet connections now run human society. We have seen nothing yet if we allow this to continue. Autonomous vehicle makers are working with law enforcement to produce cars designed to automatically pull over if they detect a police or emergency vehicle flashing from up to 100 feet away. At a police stop the car would be unlocked and the

window rolled down automatically. Vehicles would only take you where the computer (the state) allowed. The end of petrol vehicles and speed limiters on all new cars in the UK and EU from 2022 are steps leading to electric computerised transport over which ultimately you have no control. The picture is far bigger even than the Cult global network or web and that will become clear when I get to the nature of the 'spider'. There is a connection between all these happenings and the instigation of DNA-manipulating 'vaccines' (which aren't 'vaccines') justified by the 'Covid' hoax. That connection is the unfolding plan to transform the human body from a biological to a synthetic biological state and this is why synthetic biology is such a fast-emerging discipline of mainstream science. 'Covid vaccines' are infusing self-replicating synthetic genetic material into the cells to cumulatively take us on the Totalitarian Tiptoe from Human 1.0 to the synthetic biological Human 2.0 which will be physically and perceptually attached to the Smart Grid to one hundred percent control every thought, perception and deed. Humanity needs to wake up and *fast*.

This is the barest explanation of where the 'outcome' is planned to go but it's enough to see the journey happening all around us. Those new to this information will already see 'Covid' in a whole new context. I will add much more detail as we go along, but for the minutiae evidence see my mega-works, *The Answer, The Trigger* and *Everything You Need to Know But Have Never Been Told*.

Now – how does a Renegade Mind see the 'world'?

CHAPTER TWO

Renegade Perception

It is one thing to be clever and another to be wise
George R.R. Martin

A simple definition of the difference between a programmed mind and a Renegade Mind would be that one sees only dots while the other connects them to see the picture. Reading reality with accuracy requires the observer to (a) know the planned outcome and (b) realise that everything, but *everything*, is connected.

The entirety of infinite reality is connected – that's its very nature – and with human society an expression of infinite reality the same must apply. Simple cause and effect is a connection. The effect is triggered by the cause and the effect then becomes the cause of another effect. Nothing happens in isolation because it *can't*. Life in whatever reality is simple choice and consequence. We make choices and these lead to consequences. If we don't like the consequences we can make different choices and get different consequences which lead to other choices and consequences. The choice and the consequence are not only connected they are indivisible. You can't have one without the other as an old song goes. A few cannot control the world unless those being controlled allow that to happen – cause and effect, choice and consequence. Control – who has it and who doesn't – is a two-way process, a symbiotic relationship, involving the controller and controlled. 'They took my freedom away!!' Well, yes, but you also gave it to them. Humanity is

subjected to mass control because humanity has acquiesced to that control. This is all cause and effect and literally a case of give and take. In the same way world events of every kind are connected and the Cult works incessantly to sell the illusion of the random and coincidental to maintain the essential (to them) perception of dots that hide the picture. Renegade Minds know this and constantly scan the world for patterns of connection. This is absolutely pivotal in understanding the happenings in the world and without that perspective clarity is impossible. First you know the planned outcome and then you identify the steps on the journey – the day-by-day apparently random which, when connected in relation to the outcome, no longer appear as individual events, but as the proverbial *chain* of events leading in the same direction. I'll give you some examples:

Political puppet show

We are told to believe that politics is 'adversarial' in that different parties with different beliefs engage in an endless tussle for power. There may have been some truth in that up to a point – and only a point – but today divisions between 'different' parties are rhetorical not ideological. Even the rhetorical is fusing into one-speak as the parties eject any remaining free thinkers while others succumb to the ever-gathering intimidation of anyone with the 'wrong' opinion. The Cult is not a new phenomenon and can be traced back thousands of years as my books have documented. Its intergenerational initiates have been manipulating events with increasing effect the more that global power has been centralised. In ancient times the Cult secured control through the system of monarchy in which 'special' bloodlines (of which more later) demanded the right to rule as kings and queens simply by birthright and by vanquishing others who claimed the same birthright. There came a time, however, when people had matured enough to see the unfairness of such tyranny and demanded a say in who governed them. Note the word – *governed* them. Not served them – *governed* them, hence government defined as 'the political direction and control exercised over the

actions of the members, citizens, or inhabitants of communities, societies, and states; direction of the affairs of a state, community, etc.' Governments exercise control over rather than serve just like the monarchies before them. Bizarrely there are still countries like the United Kingdom which are ruled by a monarch *and* a government that officially answers to the monarch. The UK head of state and that of Commonwealth countries such as Canada, Australia and New Zealand is 'selected' by who in a *single family* had unprotected sex with whom and in what order. Pinch me it can't be true. Ouch! Shit, it is. The demise of monarchies in most countries offered a potential vacuum in which some form of free and fair society could arise and the Cult had that base covered. Monarchies had served its interests but they couldn't continue in the face of such widespread opposition and, anyway, replacing a 'royal' dictatorship that people could see with a dictatorship 'of the people' hiding behind the concept of 'democracy' presented far greater manipulative possibilities and ways of hiding coordinated tyranny behind the illusion of 'freedom'.

Democracy is quite wrongly defined as government selected by the population. This is not the case at all. It is government selected by *some* of the population (and then only in theory). This 'some' doesn't even have to be the majority as we have seen so often in first-past-the-post elections in which the so-called majority party wins fewer votes than the 'losing' parties combined. Democracy can give total power to a party in government from a minority of the votes cast. It's a sleight of hand to sell tyranny as freedom. Seventy-four million Trump-supporting Americans didn't vote for the 'Democratic' Party of Joe Biden in the distinctly dodgy election in 2020 and yet far from acknowledging the wishes and feelings of that great percentage of American society the Cult-owned Biden government set out from day one to destroy them and their right to a voice and opinion. Empty shell Biden and his Cult handlers said they were doing this to 'protect democracy'. Such is the level of lunacy and sickness to which politics has descended. Connect the dots and relate them to the desired outcome – a world government run by self-appointed technocrats and no longer even elected

politicians. While operating through its political agents in government the Cult is at the same time encouraging public distain for politicians by putting idiots and incompetents in theoretical power on the road to deleting them. The idea is to instil a public reaction that says of the technocrats: 'Well, they couldn't do any worse than the pathetic politicians.' It's all about controlling perception and Renegade Minds can see through that while programmed minds cannot when they are ignorant of both the planned outcome and the manipulation techniques employed to secure that end. This knowledge can be learned, however, and fast if people choose to get informed.

Politics may at first sight appear very difficult to control from a central point. I mean look at the 'different' parties and how would you be able to oversee them all and their constituent parts? In truth, it's very straightforward because of their structure. We are back to the pyramid of imposition and acquiescence. Organisations are structured in the same way as the system as a whole. Political parties are not open forums of free expression. They are hierarchies. I was a national spokesman for the British Green Party which claimed to be a different kind of politics in which influence and power was devolved; but I can tell you from direct experience – and it's far worse now – that Green parties are run as hierarchies like all the others however much they may try to hide that fact or kid themselves that it's not true. A very few at the top of all political parties are directing policy and personnel. They decide if you are elevated in the party or serve as a government minister and to do that you have to be a yes man or woman. Look at all the maverick political thinkers who never ascended the greasy pole. If you want to progress within the party or reach 'high-office' you need to fall into line and conform. Exceptions to this are rare indeed. Should you want to run for parliament or Congress you have to persuade the local or state level of the party to select you and for that you need to play the game as dictated by the hierarchy. If you secure election and wish to progress within the greater structure you need to go on conforming to what is acceptable to those running the hierarchy

from the peak of the pyramid. Political parties are perceptual gulags and the very fact that there are party 'Whips' appointed to 'whip' politicians into voting the way the hierarchy demands exposes the ridiculous idea that politicians are elected to serve the people they are supposed to represent. Cult operatives and manipulation has long seized control of major parties that have any chance of forming a government and at least most of those that haven't. A new party forms and the Cult goes to work to infiltrate and direct. This has reached such a level today that you see video compilations of 'leaders' of all parties whether Democrats, Republicans, Conservative, Labour and Green parroting the same Cult mantra of 'Build Back Better' and the 'Great Reset' which are straight off the Cult song-sheet to describe the transformation of global society in response to the Cult-instigated hoaxes of the 'Covid pandemic' and human-caused 'climate change'. To see Caroline Lucas, the Green Party MP that I knew when I was in the party in the 1980s, speaking in support of plans proposed by Cult operative Klaus Schwab representing the billionaire global elite is a real head-shaker.

Many parties – one master

The party system is another mind-trick and was instigated to change the nature of the dictatorship by swapping 'royalty' for dark suits that people believed – though now ever less so – represented their interests. Understanding this trick is to realise that a single force (the Cult) controls all parties either directly in terms of the major ones or through manipulation of perception and ideology with others. You don't need to manipulate Green parties to demand your transformation of society in the name of 'climate change' when they are obsessed with the lie that this is essential to 'save the planet'. You just give them a platform and away they go serving your interests while believing they are being environmentally virtuous. America's political structure is a perfect blueprint for how the two or multi-party system is really a one-party state. The Republican Party is controlled from one step back in the shadows by a group made up of billionaires and their gofers known as neoconservatives or Neocons.

I have exposed them in fine detail in my books and they were the driving force behind the policies of the imbecilic presidency of Boy George Bush which included 9/11 (see *The Trigger* for a comprehensive demolition of the official story), the subsequent 'war on terror' (war *of* terror) and the invasions of Afghanistan and Iraq. The latter was a No-Problem-Reaction-Solution based on claims by Cult operatives, including Bush and British Prime Minister Tony Blair, about Saddam Hussein's 'weapons of mass destruction' which did not exist as war criminals Bush and Blair well knew.

Figure 6: Different front people, different parties – same control system.

The Democratic Party has its own 'Neocon' group controlling from the background which I call the 'Democons' and here's the penny-drop – the Neocons and Democons answer to the same masters one step further back into the shadows (Fig 6). At that level of the Cult the Republican and Democrat parties are controlled by the same people and no matter which is in power the Cult is in power. This is how it works in almost every country and certainly in Britain with Conservative, Labour, Liberal Democrat and Green parties now all on the same page whatever the rhetoric may be in their feeble attempts to appear different. Neocons operated at the time of Bush through a think tank called The Project for the New American Century which in September, 2000, published a document entitled *Rebuilding America's Defenses: Strategies, Forces, and Resources*

For a New Century demanding that America fight 'multiple, simultaneous major theatre wars' as a 'core mission' to force regime-change in countries including Iraq, Libya and Syria. Neocons arranged for Bush ('Republican') and Blair ('Labour Party') to front-up the invasion of Iraq and when they departed the Democons orchestrated the targeting of Libya and Syria through Barack Obama ('Democrat') and British Prime Minister David Cameron ('Conservative Party'). We have 'different' parties and 'different' people, but the same unfolding script. The more the Cult has seized the reigns of parties and personnel the more their policies have transparently pursued the same agenda to the point where the fascist 'Covid' impositions of the Conservative junta of Jackboot Johnson in Britain were opposed by the Labour Party because they were not fascist enough. The Labour Party is likened to the US Democrats while the Conservative Party is akin to a British version of the Republicans and on both sides of the Atlantic they all speak the same language and support the direction demanded by the Cult although some more enthusiastically than others. It's a similar story in country after country because it's all centrally controlled. Oh, but what about Trump? I'll come to him shortly. Political 'choice' in the 'party' system goes like this: You vote for Party A and they get into government. You don't like what they do so next time you vote for Party B and they get into government. You don't like what they do when it's pretty much the same as Party A and why wouldn't that be with both controlled by the same force? Given that only two, sometimes three, parties have any chance of forming a government to get rid of Party B that you don't like you have to vote again for Party A which … you don't like. This, ladies and gentlemen, is what they call 'democracy' which we are told – wrongly – is a term interchangeable with 'freedom'.

The cult of cults

At this point I need to introduce a major expression of the Global Cult known as Sabbatian-Frankism. Sabbatian is also spelt as Sabbatean. I will summarise here. I have published major exposés

and detailed background in other works. Sabbatian-Frankism combines the names of two frauds posing as 'Jewish' men, Sabbatai Zevi (1626-1676), a rabbi, black magician and occultist who proclaimed he was the Jewish messiah; and Jacob Frank (1726-1791), the Polish 'Jew', black magician and occultist who said he was the reincarnation of 'messiah' Zevi and biblical patriarch Jacob. They worked across two centuries to establish the Sabbatian-Frankist cult that plays a major, indeed central, role in the manipulation of human society by the Global Cult which has its origins much further back in history than Sabbatai Zevi. I should emphasise two points here in response to the shrill voices that will scream 'anti-Semitism': (1) Sabbatian-Frankists are NOT Jewish and only pose as such to hide their cult behind a Jewish façade; and (2) my information about this cult has come from Jewish sources who have long realised that their society and community has been infiltrated and taken over by interloper Sabbatian-Frankists. Infiltration has been the foundation technique of Sabbatian-Frankism from its official origin in the 17th century. Zevi's Sabbatian sect attracted a massive following described as the biggest messianic movement in Jewish history, spreading as far as Africa and Asia, and he promised a return for the Jews to the 'Promised Land' of Israel. Sabbatianism was not Judaism but an inversion of everything that mainstream Judaism stood for. So much so that this sinister cult would have a feast day when Judaism had a fast day and whatever was forbidden in Judaism the Sabbatians were encouraged and even commanded to do. This included incest and what would be today called Satanism. Members were forbidden to marry outside the sect and there was a system of keeping their children ignorant of what they were part of until they were old enough to be trusted not to unknowingly reveal anything to outsiders. The same system is employed to this day by the Global Cult in general which Sabbatian-Frankism has enormously influenced and now largely controls.

Zevi and his Sabbatians suffered a setback with the intervention by the Sultan of the Islamic Ottoman Empire in the Middle East and what is now the Republic of Turkey where Zevi was located. The

Sultan gave him the choice of proving his 'divinity', converting to Islam or facing torture and death. Funnily enough Zevi chose to convert or at least appear to. Some of his supporters were disillusioned and drifted away, but many did not with 300 families also converting – only in theory – to Islam. They continued behind this Islamic smokescreen to follow the goals, rules and rituals of Sabbatianism and became known as 'crypto-Jews' or the 'Dönmeh' which means 'to turn'. This is rather ironic because they didn't 'turn' and instead hid behind a fake Islamic persona. The process of appearing to be one thing while being very much another would become the calling card of Sabbatianism especially after Zevi's death and the arrival of the Satanist Jacob Frank in the 18th century when the cult became Sabbatian-Frankism and plumbed still new depths of depravity and infiltration which included – still includes – human sacrifice and sex with children. Wherever Sabbatians go paedophilia and Satanism follow and is it really a surprise that Hollywood is so infested with child abuse and Satanism when it was established by Sabbatian-Frankists and is still controlled by them? Hollywood has been one of the prime vehicles for global perceptual programming and manipulation. How many believe the version of 'history' portrayed in movies when it is a travesty and inversion (again) of the truth? Rabbi Marvin Antelman describes Frankism in his book, *To Eliminate the Opiate*, as 'a movement of complete evil' while Jewish professor Gershom Scholem said of Frank in *The Messianic Idea in Judaism*: 'In all his actions [he was] a truly corrupt and degenerate individual ... one of the most frightening phenomena in the whole of Jewish history.' Frank was excommunicated by traditional rabbis, as was Zevi, but Frank was undeterred and enjoyed vital support from the House of Rothschild, the infamous banking dynasty whose inner-core are Sabbatian-Frankists and not Jews. Infiltration of the Roman Church and Vatican was instigated by Frank with many Dönmeh 'turning' again to convert to Roman Catholicism with a view to hijacking the reins of power. This was the ever-repeating modus operandi and continues to be so. Pose as an advocate of the religion, culture or country that you want to control and then

manipulate your people into the positions of authority and influence largely as advisers, administrators and Svengalis for those that appear to be in power. They did this with Judaism, Christianity (Christian Zionism is part of this), Islam and other religions and nations until Sabbatian-Frankism spanned the world as it does today.

Sabbatian Saudis and the terror network

One expression of the Sabbatian-Frankist Dönmeh within Islam is the ruling family of Saudi Arabia, the House of Saud, through which came the vile distortion of Islam known as Wahhabism. This is the violent creed followed by terrorist groups like Al-Qaeda and ISIS or Islamic State. Wahhabism is the hand-chopping, head-chopping 'religion' of Saudi Arabia which is used to keep the people in a constant state of fear so the interloper House of Saud can continue to rule. Al-Qaeda and Islamic State were lavishly funded by the House of Saud while being created and directed by the Sabbatian-Frankist network in the United States that operates through the Pentagon, CIA and the government in general of whichever 'party'. The front man for the establishment of Wahhabism in the middle of the 18th century was a Sabbatian-Frankist 'crypto-Jew' posing as Islamic called Muhammad ibn Abd al-Wahhab. His daughter would marry the son of Muhammad bin Saud who established the first Saudi state before his death in 1765 with support from the British Empire. Bin Saud's successors would establish modern Saudi Arabia in league with the British and Americans in 1932 which allowed them to seize control of Islam's major shrines in Mecca and Medina. They have dictated the direction of Sunni Islam ever since while Iran is the major centre of the Shiite version and here we have the source of at least the public conflict between them. The Sabbatian network has used its Wahhabi extremists to carry out Problem-Reaction-Solution terrorist attacks in the name of 'Al-Qaeda' and 'Islamic State' to justify a devastating 'war on terror', ever-increasing surveillance of the population and to terrify people into compliance. Another insight of the Renegade Mind is the streetwise understanding that

just because a country, location or people are attacked doesn't mean that those apparently representing that country, location or people are not behind the attackers. Often they are *orchestrating* the attacks because of the societal changes that can be then justified in the name of 'saving the population from terrorists'.

I show in great detail in *The Trigger* how Sabbatian-Frankists were the real perpetrators of 9/11 and not '19 Arab hijackers' who were blamed for what happened. Observe what was justified in the name of 9/11 alone in terms of Middle East invasions, mass surveillance and control that fulfilled the demands of the Project for the New American Century document published by the Sabbatian Neocons. What appear to be enemies are on the deep inside players on the same Sabbatian team. Israel and Arab 'royal' dictatorships are all ruled by Sabbatians and the recent peace agreements between Israel and Saudi Arabia, the United Arab Emirates (UAE) and others are only making formal what has always been the case behind the scenes. Palestinians who have been subjected to grotesque tyranny since Israel was bombed and terrorised into existence in 1948 have never stood a chance. Sabbatian-Frankists have controlled Israel (so the constant theme of violence and war which Sabbatians love) and they have controlled the Arab countries that Palestinians have looked to for real support that never comes. 'Royal families' of the Arab world in Saudi Arabia, Bahrain, UAE, etc., are all Sabbatians with allegiance to the aims of the cult and not what is best for their Arabic populations. They have stolen the oil and financial resources from their people by false claims to be 'royal dynasties' with a genetic right to rule and by employing vicious militaries to impose their will.

Satanic 'illumination'

The Satanist Jacob Frank formed an alliance in 1773 with two other Sabbatians, Mayer Amschel Rothschild (1744-1812), founder of the Rothschild banking dynasty, and Jesuit-educated fraudulent Jew, Adam Weishaupt, and this led to the formation of the Bavarian Illuminati, firstly under another name, in 1776. The Illuminati would

be the manipulating force behind the French Revolution (1789-1799) and was also involved in the American Revolution (1775-1783) before and after the Illuminati's official creation. Weishaupt would later become (in public) a Protestant Christian in archetypal Sabbatian style. I read that his name can be decoded as Adam-Weishaupt or 'the first man to lead those who know'. He wasn't a leader in the sense that he was a subordinate, but he did lead those below him in a crusade of transforming human society that still continues today. The theme was confirmed as early as 1785 when a horseman courier called Lanz was reported to be struck by lighting and extensive Illuminati documents were found in his saddlebags. They made the link to Weishaupt and detailed the plan for world takeover. Current events with 'Covid' fascism have been in the making for a very long time. Jacob Frank was jailed for 13 years by the Catholic Inquisition after his arrest in 1760 and on his release he headed for Frankfurt, Germany, home city and headquarters of the House of Rothschild where the alliance was struck with Mayer Amschel Rothschild and Weishaupt. Rothschild arranged for Frank to be given the title of Baron and he became a wealthy nobleman with a big following of Jews in Germany, the Austro-Hungarian Empire and other European countries. Most of them would have believed he was on their side.

The name 'Illuminati' came from the Zohar which is a body of works in the Jewish mystical 'bible' called the Kabbalah. 'Zohar' is the foundation of Sabbatian-Frankist belief and in Hebrew 'Zohar' means 'splendour', 'radiance', 'illuminated', and so we have 'Illuminati'. They claim to be the 'Illuminated Ones' from their knowledge systematically hidden from the human population and passed on through generations of carefully-chosen initiates in the global secret society network or Cult. Hidden knowledge includes an awareness of the Cult agenda for the world and the nature of our collective reality that I will explore later. Cult 'illumination' is symbolised by the torch held by the Statue of Liberty which was gifted to New York by French Freemasons in Paris who knew exactly what it represents. 'Liberty' symbolises the goddess worshipped in

Babylon as Queen Semiramis or Ishtar. The significance of this will become clear. Notice again the ubiquitous theme of inversion with the Statue of 'Liberty' really symbolising mass control (Fig 7). A mirror-image statute stands on an island in the River Seine in Paris from where New York Liberty originated (Fig 8). A large replica of the Liberty flame stands on top of the Pont de l'Alma tunnel in Paris where Princess Diana died in a Cult ritual described in *The Biggest Secret*. Lucifer 'the light bringer' is related to all this (and much more as we'll see) and 'Lucifer' is a central figure in Sabbatian-Frankism and its associated Satanism. Sabbatians reject the Jewish Torah, or Pentateuch, the 'five books of Moses' in the Old Testament known as Genesis, Exodus, Leviticus, Numbers, and Deuteronomy which are claimed by Judaism and Christianity to have been dictated by 'God' to Moses on Mount Sinai. Sabbatians say these do not apply to them and they seek to replace them with the Zohar to absorb Judaism and its followers into their inversion which is an expression of a much greater global inversion. They want to delete all religions and force humanity to worship a one-world religion – Sabbatian Satanism that also includes worship of the Earth goddess. Satanic themes are being more and more introduced into mainstream society and while Christianity is currently the foremost target for destruction the others are planned to follow.

Figure 7: The Cult goddess of Babylon disguised as the Statue of Liberty holding the flame of Lucifer the 'light bringer'.

Figure 8: Liberty's mirror image in Paris where the New York version originated.

Marx brothers

Rabbi Marvin Antelman connects the Illuminati to the Jacobins in *To Eliminate the Opiate* and Jacobins were the force behind the French Revolution. He links both to the Bund der Gerechten, or League of the Just, which was the network that inflicted communism/Marxism on the world. Antelman wrote:

> The original inner circle of the Bund der Gerechten consisted of born Catholics, Protestants and Jews [Sabbatian-Frankist infiltrators], and those representatives of respective subdivisions formulated schemes for the ultimate destruction of their faiths. The heretical Catholics laid plans which they felt would take a century or more for the ultimate destruction of the church; the apostate Jews for the ultimate destruction of the Jewish religion.

Sabbatian-created communism connects into this anti-religion agenda in that communism does not allow for the free practice of religion. The Sabbatian 'Bund' became the International Communist Party and Communist League and in 1848 'Marxism' was born with the Communist Manifesto of Sabbatian assets Karl Marx and Friedrich Engels. It is absolutely no coincidence that Marxism, just a different name for fascist and other centrally-controlled tyrannies, is being imposed worldwide as a result of the 'Covid' hoax and nor that Marxist/fascist China was the place where the hoax originated. The reason for this will become very clear in the chapter 'Covid: The calculated catastrophe'. The so-called 'Woke' mentality has hijacked

traditional beliefs of the political left and replaced them with far-right make-believe 'social justice' better known as Marxism. Woke will, however, be swallowed by its own perceived 'revolution' which is really the work of billionaires and billionaire corporations feigning being 'Woke'. Marxism is being touted by Wokers as a replacement for 'capitalism' when we don't have 'capitalism'. We have cartelism in which the market is stitched up by the very Cult billionaires and corporations bankrolling Woke. Billionaires love Marxism which keeps the people in servitude while they control from the top. Terminally naïve Wokers think they are 'changing the world' when it's the Cult that is doing the changing and when they have played their vital part and become surplus to requirements they, too, will be targeted. The Illuminati-Jacobins were behind the period known as 'The Terror' in the French Revolution in 1793 and 1794 when Jacobin Maximillian de Robespierre and his Orwellian 'Committee of Public Safety' killed 17,000 'enemies of the Revolution' who had once been 'friends of the Revolution'. Karl Marx (1818-1883), whose Sabbatian creed of Marxism has cost the lives of at least 100 million people, is a hero once again to Wokers who have been systematically kept ignorant of real history by their 'education' programming. As a result they now promote a Sabbatian 'Marxist' abomination destined at some point to consume them. Rabbi Antelman, who spent decades researching the Sabbatian plot, said of the League of the Just and Karl Marx:

> Contrary to popular opinion Karl Marx did not originate the Communist Manifesto. He was paid for his services by the League of the Just, which was known in its country of origin, Germany, as the Bund der Geaechteten.

Antelman said the text attributed to Marx was the work of other people and Marx 'was only repeating what others already said'. Marx was 'a hired hack – lackey of the wealthy Illuminists'. Marx famously said that religion was the 'opium of the people' (part of the Sabbatian plan to demonise religion) and Antelman called his books, *To Eliminate the Opiate*. Marx was born Jewish, but his family converted to Christianity (Sabbatian modus operandi) and he

attacked Jews, not least in his book, *A World Without Jews*. In doing so he supported the Sabbatian plan to destroy traditional Jewishness and Judaism which we are clearly seeing today with the vindictive targeting of orthodox Jews by the Sabbatian government of Israel over 'Covid' laws. I don't follow any religion and it has done much damage to the world over centuries and acted as a perceptual straightjacket. Renegade Minds, however, are always asking *why* something is being done. It doesn't matter if they agree or disagree with what is happening – *why* is it happening is the question. The 'why?' can be answered with regard to religion in that religions create interacting communities of believers when the Cult wants to dismantle all discourse, unity and interaction (see 'Covid' lockdowns) and the ultimate goal is to delete all religions for a one-world religion of Cult Satanism worshipping their 'god' of which more later. We see the same 'why?' with gun control in America. I don't have guns and don't want them, but why is the Cult seeking to disarm the population at the same time that law enforcement agencies are armed to their molars and why has every tyrant in history sought to disarm people before launching the final takeover? They include Hitler, Stalin, Pol Pot and Mao who followed confiscation with violent seizing of power. You know it's a Cult agenda by the people who immediately race to the microphones to exploit dead people in multiple shootings. Ultra-Zionist Cult lackey Senator Chuck Schumer was straight on the case after ten people were killed in Boulder, Colorado in March, 2121. Simple rule … if Schumer wants it the Cult wants it and the same with his ultra-Zionist mate the wild-eyed Senator Adam Schiff. At the same time they were calling for the disarmament of Americans, many of whom live a long way from a police response, Schumer, Schiff and the rest of these pampered clowns were sitting on Capitol Hill behind a razor-wired security fence protected by thousands of armed troops in addition to their own armed bodyguards. Mom and pop in an isolated home? They're just potential mass shooters.

Zion Mainframe

Sabbatian-Frankists and most importantly the Rothschilds were behind the creation of 'Zionism', a political movement that demanded a Jewish homeland in Israel as promised by Sabbatai Zevi. The very symbol of Israel comes from the German meaning of the name Rothschild. Dynasty founder Mayer Amschel Rothschild changed the family name from Bauer to Rothschild, or 'Red-Shield' in German, in deference to the six-pointed 'Star of David' hexagram displayed on the family's home in Frankfurt. The symbol later appeared on the flag of Israel after the Rothschilds were centrally involved in its creation. Hexagrams are not a uniquely Jewish symbol and are widely used in occult ('hidden') networks often as a symbol for Saturn (see my other books for why). Neither are Zionism and Jewishness interchangeable. Zionism is a political movement and philosophy and not a 'race' or a people. Many Jews oppose Zionism and many non-Jews, including US President Joe Biden, call themselves Zionists as does Israel-centric Donald Trump. America's support for the Israel government is pretty much a gimme with ultra-Zionist billionaires and corporations providing fantastic and dominant funding for both political parties. Former Congresswoman Cynthia McKinney has told how she was approached immediately she ran for office to 'sign the pledge' to Israel and confirm that she would always vote in that country's best interests. All American politicians are approached in this way. Anyone who refuses will get no support or funding from the enormous and all-powerful Zionist lobby that includes organisations like mega-lobby group AIPAC, the American Israel Public Affairs Committee. Trump's biggest funder was ultra-Zionist casino and media billionaire Sheldon Adelson while major funders of the Democratic Party include ultra-Zionist George Soros and ultra-Zionist financial and media mogul, Haim Saban. Some may reel back at the suggestion that Soros is an Israel-firster (Sabbatian-controlled Israel-firster), but Renegade Minds watch the actions not the words and everywhere Soros donates his billions the Sabbatian agenda benefits. In the spirit of Sabbatian inversion Soros pledged $1 billion for a new university network to promote 'liberal values and tackle intolerance'. He made the announcement during his annual speech

at the Cult-owned World Economic Forum in Davos, Switzerland, in January, 2020, after his 'harsh criticism' of 'authoritarian rulers' around the world. You can only laugh at such brazen mendacity. How *he* doesn't laugh is the mystery. Translated from the Orwellian 'liberal values and tackle intolerance' means teaching non-white people to hate white people and for white people to loathe themselves for being born white. The reason for that will become clear.

The 'Anti-Semitism' fraud

Zionists support the Jewish homeland in the land of Palestine which has been the Sabbatian-Rothschild goal for so long, but not for the benefit of Jews. Sabbatians and their global Anti-Semitism Industry have skewed public and political opinion to equate opposing the violent extremes of Zionism to be a blanket attack and condemnation of all Jewish people. Sabbatians and their global Anti-Semitism Industry have skewed public and political opinion to equate opposing the violent extremes of Zionism to be a blanket attack and condemnation of all Jewish people. This is nothing more than a Sabbatian protection racket to stop legitimate investigation and exposure of their agendas and activities. The official definition of 'anti-Semitism' has more recently been expanded to include criticism of Zionism – a *political movement* – and this was done to further stop exposure of Sabbatian infiltrators who created Zionism as we know it today in the 19th century. Renegade Minds will talk about these subjects when they know the shit that will come their way. People must decide if they want to know the truth or just cower in the corner in fear of what others will say. Sabbatians have been trying to label me as 'anti-Semitic' since the 1990s as I have uncovered more and more about their background and agendas. Useless, gutless, fraudulent 'journalists' then just repeat the smears without question and on the day I was writing this section a pair of unquestioning repeaters called Ben Quinn and Archie Bland (how appropriate) outright called me an 'anti-Semite' in the establishment propaganda sheet, the London *Guardian*, with no supporting evidence. The

Sabbatian Anti-Semitism Industry said so and who are they to question that? They wouldn't dare. Ironically 'Semitic' refers to a group of languages in the Middle East that are almost entirely Arabic. 'Anti-Semitism' becomes 'anti-Arab' which if the consequences of this misunderstanding were not so grave would be hilarious. Don't bother telling Quinn and Bland. I don't want to confuse them, bless 'em. One reason I am dubbed 'anti-Semitic' is that I wrote in the 1990s that Jewish operatives (Sabbatians) were heavily involved in the Russian Revolution when Sabbatians overthrew the Romanov dynasty. This apparently made me 'anti-Semitic'. Oh, really? Here is a section from *The Trigger*:

British journalist Robert Wilton confirmed these themes in his 1920 book *The Last Days of the Romanovs* when he studied official documents from the Russian government to identify the members of the Bolshevik ruling elite between 1917 and 1919. The Central Committee included 41 Jews among 62 members; the Council of the People's Commissars had 17 Jews out of 22 members; and 458 of the 556 most important Bolshevik positions between 1918 and 1919 were occupied by Jewish people. Only 17 were Russian. Then there were the 23 Jews among the 36 members of the vicious Cheka Soviet secret police established in 1917 who would soon appear all across the country.

Professor Robert Service of Oxford University, an expert on 20th century Russian history, found evidence that ['Jewish'] Leon Trotsky had sought to make sure that Jews were enrolled in the Red Army and were disproportionately represented in the Soviet civil bureaucracy that included the Cheka which performed mass arrests, imprisonment and executions of 'enemies of the people'. A US State Department Decimal File (861.00/5339) dated November 13th, 1918, names [Rothschild banking agent in America] Jacob Schiff and a list of ultra-Zionists as funders of the Russian Revolution leading to claims of a 'Jewish plot', but the key point missed by all is they were not 'Jews' – they were Sabbatian-Frankists.

Britain's Winston Churchill made the same error by mistake or otherwise. He wrote in a 1920 edition of the *Illustrated Sunday Herald* that those behind the Russian revolution were part of a 'worldwide conspiracy for the overthrow of civilisation and for the reconstitution of society on the basis of arrested development, of envious malevolence, and impossible equality' (see 'Woke' today because that has been created by the same network). Churchill said there was no need to exaggerate the part played in the creation of Bolshevism and in the actual bringing about of the Russian

Revolution 'by these international and for the most part atheistical Jews' ['atheistical Jews' = Sabbatians]. Churchill said it is certainly a very great one and probably outweighs all others: 'With the notable exception of Lenin, the majority of the leading figures are Jews.' He went on to describe, knowingly or not, the Sabbatian modus operandi of placing puppet leaders nominally in power while they control from the background:

> Moreover, the principal inspiration and driving power comes from the Jewish leaders. Thus Tchitcherin, a pure Russian, is eclipsed by his nominal subordinate, Litvinoff, and the influence of Russians like Bukharin or Lunacharski cannot be compared with the power of Trotsky, or of Zinovieff, the Dictator of the Red Citadel (Petrograd), or of Krassin or Radek – all Jews. In the Soviet institutions the predominance of Jews is even more astonishing. And the prominent, if not indeed the principal, part in the system of terrorism applied by the Extraordinary Commissions for Combatting Counter-Revolution has been taken by Jews, and in some notable cases by Jewesses.

What I said about seriously disproportionate involvement in the Russian Revolution by Jewish 'revolutionaries' (Sabbatians) is provable fact, but truth is no defence against the Sabbatian Anti-Semitism Industry, its repeater parrots like Quinn and Bland, and the now breathtaking network of so-called 'Woke' 'anti-hate' groups with interlocking leaderships and funding which have the role of discrediting and silencing anyone who gets too close to exposing the Sabbatians. We have seen 'truth is no defence' confirmed in legal judgements with the Saskatchewan Human Rights Commission in Canada decreeing this: 'Truthful statements can be presented in a manner that would meet the definition of hate speech, and not all truthful statements must be free from restriction.' Most 'anti-hate' activists, who are themselves consumed by hatred, are too stupid and ignorant of the world to know how they are being used. They are far too far up their own virtue-signalling arses and it's far too dark for them to see anything.

The 'revolution' game

The background and methods of the 'Russian' Revolution are straight from the Sabbatian playbook seen in the French Revolution

and endless others around the world that appear to start as a revolution of the people against tyrannical rule and end up with a regime change to more tyrannical rule overtly or covertly. Wars, terror attacks and regime overthrows follow the Sabbatian cult through history with its agents creating them as Problem-Reaction-Solutions to remove opposition on the road to world domination. Sabbatian dots connect the Rothschilds with the Illuminati, Jacobins of the French Revolution, the 'Bund' or League of the Just, the International Communist Party, Communist League and the Communist Manifesto of Karl Marx and Friedrich Engels that would lead to the Rothschild-funded Russian Revolution. The sequence comes under the heading of 'creative destruction' when you advance to your global goal by continually destroying the status quo to install a new status quo which you then also destroy. The two world wars come to mind. With each new status quo you move closer to your planned outcome. Wars and mass murder are to Sabbatians a collective blood sacrifice ritual. They are obsessed with death for many reasons and one is that death is an inversion of life. Satanists and Sabbatians are obsessed with death and often target churches and churchyards for their rituals. Inversion-obsessed Sabbatians explain the use of inverted symbolism including the *inverted* pentagram and *inverted* cross. The inversion of the cross has been related to targeting Christianity, but the cross was a religious symbol long before Christianity and its inversion is a statement about the Sabbatian mentality and goals more than any single religion.

Sabbatians operating in Germany were behind the rise of the occult-obsessed Nazis and the subsequent Jewish exodus from Germany and Europe to Palestine and the United States after World War Two. The Rothschild dynasty was at the forefront of this both as political manipulators and by funding the operation. Why would Sabbatians help to orchestrate the horrors inflicted on Jews by the Nazis and by Stalin after they organised the Russian Revolution? Sabbatians hate Jews and their religion, that's why. They pose as Jews and secure positions of control within Jewish society and play the 'anti-Semitism' card to protect themselves from exposure

through a global network of organisations answering to the Sabbatian-created-and-controlled globe-spanning intelligence network that involves a stunning web of military-intelligence operatives and operations for a tiny country of just nine million. Among them are Jewish assets who are not Sabbatians but have been convinced by them that what they are doing is for the good of Israel and the Jewish community to protect them from what they have been programmed since childhood to believe is a Jew-hating hostile world. The Jewish community is just a highly convenient cover to hide the true nature of Sabbatians. Anyone getting close to exposing their game is accused by Sabbatian place-people and gofers of 'anti-Semitism' and claiming that all Jews are part of a plot to take over the world. I am not saying that. I am saying that Sabbatians – the *real* Jew-haters – have infiltrated the Jewish community to use them both as a cover and an 'anti-Semitic' defence against exposure. Thus we have the Anti-Semitism Industry targeted researchers in this way and most Jewish people think this is justified and genuine. They don't know that their 'Jewish' leaders and institutions of state, intelligence and military are not controlled by Jews at all, but cultists and stooges of Sabbatian-Frankism. I once added my name to a pro-Jewish freedom petition online and the next time I looked my name was gone and text had been added to the petition blurb to attack me as an 'anti-Semite' such is the scale of perceptual programming.

Moving on America

I tell the story in *The Trigger* and a chapter called 'Atlantic Crossing' how particularly after Israel was established the Sabbatians moved in on the United States and eventually grasped control of government administration, the political system via both Democrats and Republicans, the intelligence community like the CIA and National Security Agency (NSA), the Pentagon and mass media. Through this seriously compartmentalised network Sabbatians and their operatives in Mossad, Israeli Defense Forces (IDF) and US agencies pulled off 9/11 and blamed it on 19 'Al-Qaeda hijackers' dominated by men from, or connected to, Sabbatian-ruled Saudi

Arabia. The '19' were not even on the planes let alone flew those big passenger jets into buildings while being largely incompetent at piloting one-engine light aircraft. 'Hijacker' Hani Hanjour who is said to have flown American Airlines Flight 77 into the Pentagon with a turn and manoeuvre most professional pilots said they would have struggled to do was banned from renting a small plane by instructors at the Freeway Airport in Bowie, Maryland, just *six weeks* earlier on the grounds that he was an incompetent pilot. The Jewish population of the world is just 0.2 percent with even that almost entirely concentrated in Israel (75 percent Jewish) and the United States (around two percent). This two percent and globally 0.2 percent refers to *Jewish* people and not Sabbatian interlopers who are a fraction of that fraction. What a sobering thought when you think of the fantastic influence on world affairs of tiny Israel and that the Project for the New America Century (PNAC) which laid out the blueprint in September, 2000, for America's war on terror and regime change wars in Iraq, Libya and Syria was founded and dominated by Sabbatians known as 'Neocons'. The document conceded that this plan would not be supported politically or publicly without a major attack on American soil and a Problem-Reaction-Solution excuse to send troops to war across the Middle East. Sabbatian Neocons said:

> ... [The] process of transformation ... [war and regime change] ... is likely to be a long one, absent some catastrophic and catalysing event – like a new Pearl Harbor.

Four months later many of those who produced that document came to power with their inane puppet George Bush from the long-time Sabbatian Bush family. They included Sabbatian Dick Cheney who was officially vice-president, but really de-facto president for the entirety of the 'Bush' government. Nine months after the 'Bush' inauguration came what Bush called at the time 'the Pearl Harbor of the 21st century' and with typical Sabbatian timing and symbolism 2001 was the 60th anniversary of the attack in 1941 by the Japanese Air Force on Pearl Harbor, Hawaii, which allowed President Franklin Delano Roosevelt to take the United States into a Sabbatian-

instigated Second World War that he said in his election campaign that he never would. The evidence is overwhelming that Roosevelt and his military and intelligence networks knew the attack was coming and did nothing to stop it, but they did make sure that America's most essential naval ships were not in Hawaii at the time. Three thousand Americans died in the Pearl Harbor attacks as they did on September 11th. By the 9/11 year of 2001 Sabbatians had widely infiltrated the US government, military and intelligence operations and used their compartmentalised assets to pull off the 'Al-Qaeda' attacks. If you read *The Trigger* it will blow your mind to see the utterly staggering concentration of 'Jewish' operatives (Sabbatian infiltrators) in essential positions of political, security, legal, law enforcement, financial and business power before, during, and after the attacks to make them happen, carry them out, and then cover their tracks – and I do mean *staggering* when you think of that 0.2 percent of the world population and two percent of Americans which are Jewish while Sabbatian infiltrators are a fraction of that. A central foundation of the 9/11 conspiracy was the hijacking of government, military, Air Force and intelligence computer systems in real time through 'back-door' access made possible by Israeli (Sabbatian) 'cyber security' software. Sabbatian-controlled Israel is on the way to rivalling Silicon Valley for domination of cyberspace and is becoming the dominant force in cyber-security which gives them access to entire computer systems and their passcodes across the world. Then add to this that Zionists head (officially) Silicon Valley giants like Google (Larry Page and Sergey Brin), Google-owned YouTube (Susan Wojcicki), Facebook (Mark Zuckerberg and Sheryl Sandberg), and Apple (Chairman Arthur D. Levinson), and that ultra-Zionist hedge fund billionaire Paul Singer has a $1 billion stake in Twitter which is only nominally headed by 'CEO' pothead Jack Dorsey. As cable news host Tucker Carlson said of Dorsey: 'There used to be debate in the medical community whether dropping a ton of acid had permanent effects and I think that debate has now ended.' Carlson made the comment after Dorsey told a hearing on Capitol Hill (if you cut through his bullshit) that he

believed in free speech so long as he got to decide what you can hear and see. These 'big names' of Silicon Valley are only front men and women for the Global Cult, not least the Sabbatians, who are the true controllers of these corporations. Does anyone still wonder why these same people and companies have been ferociously censoring and banning people (like me) for exposing any aspect of the Cult agenda and especially the truth about the 'Covid' hoax which Sabbatians have orchestrated?

The Jeffrey Epstein paedophile ring was a Sabbatian operation. He was officially 'Jewish' but he was a Sabbatian and women abused by the ring have told me about the high number of 'Jewish' people involved. The Epstein horror has Sabbatian written all over it and matches perfectly their modus operandi and obsession with sex and ritual. Epstein was running a Sabbatian blackmail ring in which famous people with political and other influence were provided with young girls for sex while everything was being filmed and recorded on hidden cameras and microphones at his New York house, Caribbean island and other properties. Epstein survivors have described this surveillance system to me and some have gone public. Once the famous politician or other figure knew he or she was on video they tended to do whatever they were told. Here we go again …when you've got them by the balls their hearts and minds will follow. Sabbatians use this blackmail technique on a wide scale across the world to entrap politicians and others they need to act as demanded. Epstein's private plane, the infamous 'Lolita Express', had many well-known passengers including Bill Clinton while Bill Gates has flown on an Epstein plane and met with him four years after Epstein had been jailed for paedophilia. They subsequently met many times at Epstein's home in New York according to a witness who was there. Epstein's infamous side-kick was Ghislaine Maxwell, daughter of Mossad agent and ultra-Zionist mega-crooked British businessman, Bob Maxwell, who at one time owned the *Daily Mirror* newspaper. Maxwell was murdered at sea on his boat in 1991 by Sabbatian-controlled Mossad when he became a liability with his

business empire collapsing as a former Mossad operative has confirmed (see *The Trigger*).

Money, money, money, funny money ...

Before I come to the Sabbatian connection with the last three US presidents I will lay out the crucial importance to Sabbatians of controlling banking and finance. Sabbatian Mayer Amschel Rothschild set out to dominate this arena in his family's quest for total global control. What is freedom? It is, in effect, choice. The more choices you have the freer you are and the fewer your choices the more you are enslaved. In the global structure created over centuries by Sabbatians the biggest decider and restrictor of choice is … money. Across the world if you ask people what they would like to do with their lives and why they are not doing that they will reply 'I don't have the money'. This is the idea. A global elite of multi-billionaires are described as 'greedy' and that is true on one level; but control of money – who has it and who doesn't – is not primarily about greed. It's about control. Sabbatians have seized ever more control of finance and sucked the wealth of the world out of the hands of the population. We talk now, after all, about the 'One-percent' and even then the wealthiest are a lot fewer even than that. This has been made possible by a money scam so outrageous and so vast it could rightly be called the scam of scams founded on creating 'money' out of nothing and 'loaning' that with interest to the population. Money out of nothing is called 'credit'. Sabbatians have asserted control over governments and banking ever more completely through the centuries and secured financial laws that allow banks to lend hugely more than they have on deposit in a confidence trick known as fractional reserve lending. Imagine if you could lend money that doesn't exist and charge the recipient interest for doing so. You would end up in jail. Bankers by contrast end up in mansions, private jets, Malibu and Monaco.

Banks are only required to keep a fraction of their deposits and wealth in their vaults and they are allowed to lend 'money' they don't have called 'credit. Go into a bank for a loan and if you succeed

the banker will not move any real wealth into your account. They will type into your account the amount of the agreed 'loan' – say £100,000. This is not wealth that really exists; it is non-existent, fresh-air, created-out-of-nothing 'credit' which has never, does not, and will never exist except in theory. Credit is backed by nothing except wind and only has buying power because people think that it has buying power and accept it in return for property, goods and services. I have described this situation as like those cartoon characters you see chasing each other and when they run over the edge of a cliff they keep running forward on fresh air until one of them looks down, realises what's happened, and they all crash into the ravine. The whole foundation of the Sabbatian financial system is to stop people looking down except for periodic moments when they want to crash the system (as in 2008 and 2020 ongoing) and reap the rewards from all the property, businesses and wealth their borrowers had signed over as 'collateral' in return for a 'loan' of fresh air. Most people think that money is somehow created by governments when it comes into existence from the start as a debt through banks 'lending' illusory money called credit. Yes, the very currency of exchange is a *debt* from day one issued as an interest-bearing loan. Why don't governments create money interest-free and lend it to their people interest-free? Governments are controlled by Sabbatians and the financial system is controlled by Sabbatians for whom interest-free money would be a nightmare come true. Sabbatians underpin their financial domination through their global network of central banks, including the privately-owned US Federal Reserve and Britain's Bank of England, and this is orchestrated by a privately-owned central bank coordination body called the Bank for International Settlements in Basle, Switzerland, created by the usual suspects including the Rockefellers and Rothschilds. Central bank chiefs don't answer to governments or the people. They answer to the Bank for International Settlements or, in other words, the Global Cult which is dominated today by Sabbatians.

Built-in disaster

There are so many constituent scams within the overall banking scam. When you take out a loan of thin-air credit only the amount of that loan is theoretically brought into circulation to add to the amount in circulation; but you are paying back the principle plus interest. The additional interest is not created and this means that with every 'loan' there is a shortfall in the money in circulation between what is borrowed and what has to be paid back. There is never even close to enough money in circulation to repay all outstanding public and private debt including interest. Coldly weaved in the very fabric of the system is the certainty that some will lose their homes, businesses and possessions to the banking 'lender'. This is less obvious in times of 'boom' when the amount of money in circulation (and the debt) is expanding through more people wanting and getting loans. When a downturn comes and the money supply contracts it becomes painfully obvious that there is not enough money to service all debt and interest. This is less obvious in times of 'boom' when the amount of money in circulation (and the debt) is expanding through more people wanting and getting loans. When a downturn comes and the money supply contracts and it becomes painfully obvious – as in 2008 and currently – that there is not enough money to service all debt and interest. Sabbatian banksters have been leading the human population through a calculated series of booms (more debt incurred) and busts (when the debt can't be repaid and the banks get the debtor's tangible wealth in exchange for non-existent 'credit'). With each 'bust' Sabbatian bankers have absorbed more of the world's tangible wealth and we end up with the One-percent. Governments are in bankruptcy levels of debt to the same system and are therefore owned by a system they do not control. The Federal Reserve, 'America's central bank', is privately-owned and American presidents only nominally appoint its chairman or woman to maintain the illusion that it's an arm of government. It's not. The 'Fed' is a cartel of private banks which handed billions to its associates and friends after the crash of 2008 and has been Sabbatian-controlled since it was manipulated into being in 1913 through the covert trickery of Rothschild banking agents Jacob Schiff and Paul

Warburg, and the Sabbatian Rockefeller family. Somehow from a Jewish population of two-percent and globally 0.2 percent (Sabbatian interlopers remember are far smaller) ultra-Zionists headed the Federal Reserve for 31 years between 1987 and 2018 in the form of Alan Greenspan, Bernard Bernanke and Janet Yellen (now Biden's Treasury Secretary) with Yellen's deputy chairman a Israeli-American duel citizen and ultra-Zionist Stanley Fischer, a former governor of the Bank of Israel. Ultra-Zionist Fed chiefs spanned the presidencies of Ronald Reagan ('Republican'), Father George Bush ('Republican'), Bill Clinton ('Democrat'), Boy George Bush ('Republican') and Barack Obama ('Democrat'). We should really add the pre-Greenspan chairman, Paul Adolph Volcker, 'appointed' by Jimmy Carter ('Democrat') who ran the Fed between 1979 and 1987 during the Carter and Reagan administrations before Greenspan took over. Volcker was a long-time associate and business partner of the Rothschilds. No matter what the 'party' officially in power the United States economy was directed by the same force. Here are members of the Obama, Trump and Biden administrations and see if you can make out a common theme.

Barack Obama ('Democrat')

Ultra-Zionists Robert Rubin, Larry Summers, and Timothy Geithner ran the US Treasury in the Clinton administration and two of them reappeared with Obama. Ultra-Zionist Fed chairman Alan Greenspan had manipulated the crash of 2008 through deregulation and jumped ship just before the disaster to make way for ultra-Zionist Bernard Bernanke to hand out trillions to Sabbatian 'too big to fail' banks and businesses, including the ubiquitous ultra-Zionist Goldman Sachs which has an ongoing staff revolving door operation between itself and major financial positions in government worldwide. Obama inherited the fallout of the crash when he took office in January, 2009, and fortunately he had the support of his ultra-Zionist White House Chief of Staff Rahm Emmanuel, son of a terrorist who helped to bomb Israel into being in 1948, and his ultra-Zionist senior adviser David Axelrod, chief strategist in Obama's two

successful presidential campaigns. Emmanuel, later mayor of Chicago and former senior fundraiser and strategist for Bill Clinton, is an example of the Sabbatian policy after Israel was established of migrating insider families to America so their children would be born American citizens. 'Obama' chose this financial team throughout his administration to respond to the Sabbatian-instigated crisis:

Timothy Geithner (ultra-Zionist) Treasury Secretary; Jacob J. Lew, Treasury Secretary; Larry Summers (ultra-Zionist), director of the White House National Economic Council; Paul Adolph Volcker (Rothschild business partner), chairman of the Economic Recovery Advisory Board; Peter Orszag (ultra-Zionist), director of the Office of Management and Budget overseeing all government spending; Penny Pritzker (ultra-Zionist), Commerce Secretary; Jared Bernstein (ultra-Zionist), chief economist and economic policy adviser to Vice President Joe Biden; Mary Schapiro (ultra-Zionist), chair of the Securities and Exchange Commission (SEC); Gary Gensler (ultra-Zionist), chairman of the Commodity Futures Trading Commission (CFTC); Sheila Bair (ultra-Zionist), chair of the Federal Deposit Insurance Corporation (FDIC); Karen Mills (ultra-Zionist), head of the Small Business Administration (SBA); Kenneth Feinberg (ultra-Zionist), Special Master for Executive [bail-out] Compensation. Feinberg would be appointed to oversee compensation (with strings) to 9/11 victims and families in a campaign to stop them having their day in court to question the official story. At the same time ultra-Zionist Bernard Bernanke was chairman of the Federal Reserve and these are only some of the ultra-Zionists with allegiance to Sabbatian-controlled Israel in the Obama government. Obama's biggest corporate donor was ultra-Zionist Goldman Sachs which had employed many in his administration.

Donald Trump ('Republican')

Trump claimed to be an outsider (he wasn't) who had come to 'drain the swamp'. He embarked on this goal by immediately appointing ultra-Zionist Steve Mnuchin, a Goldman Sachs employee for 17

years, as his Treasury Secretary. Others included Gary Cohn (ultra-Zionist), chief operating officer of Goldman Sachs, his first Director of the National Economic Council and chief economic adviser, who was later replaced by Larry Kudlow (ultra-Zionist). Trump's senior adviser throughout his four years in the White House was his sinister son-in-law Jared Kushner, a life-long friend of Israel Prime Minister Benjamin Netanyahu. Kushner is the son of a convicted crook who was pardoned by Trump in his last days in office. Other ultra-Zionists in the Trump administration included: Stephen Miller, Senior Policy Adviser; Avrahm Berkowitz, Deputy Adviser to Trump and his Senior Adviser Jared Kushner; Ivanka Trump, Adviser to the President, who converted to Judaism when she married Jared Kushner; David Friedman, Trump lawyer and Ambassador to Israel; Jason Greenblatt, Trump Organization executive vice president and chief legal officer, who was made Special Representative for International Negotiations and the Israeli-Palestinian Conflict; Rod Rosenstein, Deputy Attorney General; Elliot Abrams, Special Representative for Venezuela, then Iran; John Eisenberg, National Security Council Legal Adviser and Deputy Council to the President for National Security Affairs; Anne Neuberger, Deputy National Manager, National Security Agency; Ezra Cohen-Watnick, Acting Under Secretary of Defense for Intelligence; Elan Carr, Special Envoy to monitor and combat anti-Semitism; Len Khodorkovsky, Deputy Special Envoy to monitor and combat anti-Semitism; Reed Cordish, Assistant to the President, Intragovernmental and Technology Initiatives. Trump Vice President Mike Pence and Secretary of State Mike Pompeo, both Christian Zionists, were also vehement supporters of Israel and its goals and ambitions.

Donald 'free-speech believer' Trump pardoned a number of financial and violent criminals while ignoring calls to pardon Julian Assange and Edward Snowden whose crimes are revealing highly relevant information about government manipulation and corruption and the widespread illegal surveillance of the American people by US 'security' agencies. It's so good to know that Trump is on the side of freedom and justice and not mega-criminals with

allegiance to Sabbatian-controlled Israel. These included a pardon for Israeli spy Jonathan Pollard who was jailed for life in 1987 under the Espionage Act. Aviem Sella, the Mossad agent who recruited Pollard, was also pardoned by Trump while Assange sat in jail and Snowden remained in exile in Russia. Sella had 'fled' (was helped to escape) to Israel in 1987 and was never extradited despite being charged under the Espionage Act. A Trump White House statement said that Sella's clemency had been 'supported by Benjamin Netanyahu, Ron Dermer, Israel's US Ambassador, David Friedman, US Ambassador to Israel and Miriam Adelson, wife of leading Trump donor Sheldon Adelson who died shortly before. Other friends of Jared Kushner were pardoned along with Sholom Weiss who was believed to be serving the longest-ever white-collar prison sentence of more than 800 years in 2000. The sentence was commuted of Ponzi-schemer Eliyahu Weinstein who defrauded Jews and others out of $200 million. I did mention that Assange and Snowden were ignored, right? Trump gave Sabbatians almost everything they asked for in military and political support, moving the US Embassy from Tel Aviv to Jerusalem with its critical symbolic and literal implications for Palestinian statehood, and the 'deal of the Century' designed by Jared Kushner and David Friedman which gave the Sabbatian Israeli government the green light to substantially expand its already widespread program of building illegal Jewish-only settlements in the occupied land of the West Bank. This made a two-state 'solution' impossible by seizing all the land of a potential Palestinian homeland and that had been the plan since 1948 and then 1967 when the Arab-controlled Gaza Strip, West Bank, Sinai Peninsula and Syrian Golan Heights were occupied by Israel. All the talks about talks and road maps and delays have been buying time until the West Bank was physically occupied by Israeli real estate. Trump would have to be a monumentally ill-informed idiot not to see that this was the plan he was helping to complete. The Trump administration was in so many ways the Kushner administration which means the Netanyahu administration which means the Sabbatian administration. I understand why many opposing Cult fascism in all its forms gravitated to Trump, but he

was a crucial part of the Sabbatian plan and I will deal with this in the next chapter.

Joe Biden ('Democrat')

A barely cognitive Joe Biden took over the presidency in January, 2021, along with his fellow empty shell, Vice-President Kamala Harris, as the latest Sabbatian gofers to enter the White House. Names on the door may have changed and the 'party' – the force behind them remained the same as Zionists were appointed to a stream of pivotal areas relating to Sabbatian plans and policy. They included: Janet Yellen, Treasury Secretary, former head of the Federal Reserve, and still another ultra-Zionist running the US Treasury after Mnuchin (Trump), Lew and Geithner (Obama), and Summers and Rubin (Clinton); Anthony Blinken, Secretary of State; Wendy Sherman, Deputy Secretary of State (so that's 'Biden's' Sabbatian foreign policy sorted); Jeff Zients, White House coronavirus coordinator; Rochelle Walensky, head of the Centers for Disease Control; Rachel Levine, transgender deputy health secretary (that's 'Covid' hoax policy under control); Merrick Garland, Attorney General; Alejandro Mayorkas, Secretary of Homeland Security; Cass Sunstein, Homeland Security with responsibility for new immigration laws; Avril Haines, Director of National Intelligence; Anne Neuberger, National Security Agency cybersecurity director (note, cybersecurity); David Cohen, CIA Deputy Director; Ronald Klain, Biden's Chief of Staff (see Rahm Emanuel); Eric Lander, a 'leading geneticist', Office of Science and Technology Policy director (see Smart Grid, synthetic biology agenda); Jessica Rosenworcel, acting head of the Federal Communications Commission (FCC) which controls Smart Grid technology policy and electromagnetic communication systems including 5G. How can it be that so many pivotal positions are held by two-percent of the American population and 0.2 percent of the world population administration after administration no matter who is the president and what is the party? It's a coincidence? Of course it's not and this is why Sabbatians have built their colossal global web of interlocking 'anti-

hate' hate groups to condemn anyone who asks these glaring questions as an 'anti-Semite'. The way that Jewish people horrifically abused in Sabbatian-backed Nazi Germany are exploited to this end is stomach-turning and disgusting beyond words.

Political fusion

Sabbatian manipulation has reversed the roles of Republicans and Democrats and the same has happened in Britain with the Conservative and Labour Parties. Republicans and Conservatives were always labelled the 'right' and Democrats and Labour the 'left', but look at the policy positions now and the Democrat-Labour 'left' has moved further to the 'right' than Republicans and Conservatives under the banner of 'Woke', the Cult-created far-right tyranny. Where once the Democrat-Labour 'left' defended free speech and human rights they now seek to delete them and as I said earlier despite the 'Covid' fascism of the Jackboot Johnson Conservative government in the UK the Labour Party of leader Keir Starmer demanded even more extreme measures. The Labour Party has been very publicly absorbed by Sabbatians after a political and media onslaught against the previous leader, the weak and inept Jeremy Corbyn, over made-up allegations of 'anti-Semitism' both by him and his party. The plan was clear with this 'anti-Semite' propaganda and what was required in response was a swift and decisive 'fuck off' from Corbyn and a statement to expose the Anti-Semitism Industry (Sabbatian) attempt to silence Labour criticism of the Israeli government (Sabbatians) and purge the party of all dissent against the extremes of ultra-Zionism (Sabbatians). Instead Corbyn and his party fell to their knees and appeased the abusers which, by definition, is impossible. Appeasing one demand leads only to a new demand to be appeased until takeover is complete. Like I say – 'fuck off' would have been a much more effective policy and I have used it myself with great effect over the years when Sabbatians are on my case which is most of the time. I consider that fact a great compliment, by the way. The outcome of the Labour Party capitulation is that we now have a Sabbatian-controlled

Conservative Party 'opposed' by a Sabbatian-controlled Labour Party in a one-party Sabbatian state that hurtles towards the extremes of tyranny (the Sabbatian cult agenda). In America the situation is the same. Labour's Keir Starmer spends his days on his knees with his tongue out pointing to Tel Aviv, or I guess now Jerusalem, while Boris Johnson has an 'anti-Semitism czar' in the form of former Labour MP John Mann who keeps Starmer company on his prayer mat.

Sabbatian influence can be seen in Jewish members of the Labour Party who have been ejected for criticism of Israel including those from families that suffered in Nazi Germany. Sabbatians despise real Jewish people and target them even more harshly because it is so much more difficult to dub them 'anti-Semitic' although in their desperation they do try.

CHAPTER THREE

The Pushbacker sting

Until you realize how easy it is for your mind to be manipulated, you remain the puppet of someone else's game
Evita Ochel

I will use the presidencies of Trump and Biden to show how the manipulation of the one-party state plays out behind the illusion of political choice across the world. No two presidencies could – on the face of it – be more different and apparently at odds in terms of direction and policy.

A Renegade Mind sees beyond the obvious and focuses on outcomes and consequences and not image, words and waffle. The Cult embarked on a campaign to divide America between those who blindly support its agenda (the mentality known as 'Woke') and those who are pushing back on where the Cult and its Sabbatians want to go. This presents infinite possibilities for dividing and ruling the population by setting them at war with each other and allows a perceptual ring fence of demonisation to encircle the Pushbackers in a modern version of the Little Big Horn in 1876 when American cavalry led by Lieutenant Colonel George Custer were drawn into a trap, surrounded and killed by Native American tribes defending their land of thousands of years from being seized by the government. In this modern version the roles are reversed and it's those defending themselves from the Sabbatian government who are surrounded and the government that's seeking to destroy them. This trap was set years ago and to explain how we must return to 2016

and the emergence of Donald Trump as a candidate to be President of the United States. He set out to overcome the best part of 20 other candidates in the Republican Party before and during the primaries and was not considered by many in those early stages to have a prayer of living in the White House. The Republican Party was said to have great reservations about Trump and yet somehow he won the nomination. When you know how American politics works – politics in general – there is no way that Trump could have become the party's candidate unless the Sabbatian-controlled 'Neocons' that run the Republican Party wanted that to happen. We saw the proof in emails and documents made public by WikiLeaks that the Democratic Party hierarchy, or Democons, systematically undermined the campaign of Bernie Sanders to make sure that Sabbatian gofer Hillary Clinton won the nomination to be their presidential candidate. If the Democons could do that then the Neocons in the Republican Party could have derailed Trump in the same way. But they didn't and at that stage I began to conclude that Trump could well be the one chosen to be president. If that was the case the 'why' was pretty clear to see – the goal of dividing America between Cult agenda-supporting Wokers and Pushbackers who gravitated to Trump because he was telling them what they wanted to hear. His constituency of support had been increasingly ignored and voiceless for decades and profoundly through the eight years of Sabbatian puppet Barack Obama. Now here was someone speaking their language of pulling back from the incessant globalisation of political and economic power, the exporting of American jobs to China and elsewhere by 'American' (Sabbatian) corporations, the deletion of free speech, and the mass immigration policies that had further devastated job opportunities for the urban working class of all races and the once American heartlands of the Midwest.

Beware the forked tongue

Those people collectively sighed with relief that at last a political leader was apparently on their side, but another trait of the Renegade Mind is that you look even harder at people telling you

what you want to hear than those who are telling you otherwise. Obviously as I said earlier people wish what they want to hear to be true and genuine and they are much more likely to believe that than someone saying what they don't want to here and don't want to be true. Sales people are taught to be skilled in eliciting by calculated questioning what their customers want to hear and repeating that back to them as their own opinion to get their targets to like and trust them. Assets of the Cult are also sales people in the sense of selling perception. To read Cult manipulation you have to play the long and expanded game and not fall for the Vaudeville show of party politics. Both American parties are vehicles for the Cult and they exploit them in different ways depending on what the agenda requires at that moment. Trump and the Republicans were used to be the focus of dividing America and isolating Pushbackers to open the way for a Biden presidency to become the most extreme in American history by advancing the full-blown Woke (Cult) agenda with the aim of destroying and silencing Pushbackers now labelled Nazi Trump supporters and white supremacists.

Sabbatians wanted Trump in office for the reasons described by ultra-Zionist Saul Alinsky (1909-1972) who was promoting the Woke philosophy through 'community organising' long before anyone had heard of it. In those days it still went by its traditional name of Marxism. The reason for the manipulated Trump phenomenon was laid out in Alinsky's 1971 book, *Rules for Radicals*, which was his blueprint for overthrowing democratic and other regimes and replacing them with Sabbatian Marxism. Not surprisingly his to-do list was evident in the Sabbatian French and Russian 'Revolutions' and that in China which will become very relevant in the next chapter about the 'Covid' hoax. Among Alinsky's followers have been the deeply corrupt Barack Obama, House Speaker Nancy Pelosi and Hillary Clinton who described him as a 'hero'. All three are Sabbatian stooges with Pelosi personifying the arrogant corrupt idiocy that so widely fronts up for the Cult inner core. Predictably as a Sabbatian advocate of the 'light-bringer' Alinsky features Lucifer on the dedication page of his book as the original radical who gained

his own kingdom ('Earth' as we shall see). One of Alinsky's golden radical rules was to pick an individual and focus all attention, hatred and blame on them and not to target faceless bureaucracies and corporations. *Rules for Radicals* is really a Sabbatian handbook with its contents repeatedly employed all over the world for centuries and why wouldn't Sabbatians bring to power their designer-villain to be used as the individual on which all attention, hatred and blame was bestowed? This is what they did and the only question for me is how much Trump knew that and how much he was manipulated. A bit of both, I suspect. This was Alinsky's Trump technique from a man who died in 1972. The technique has spanned history:

> Pick the target, freeze it, personalize it, polarize it. Don't try to attack abstract corporations or bureaucracies. Identify a responsible individual. Ignore attempts to shift or spread the blame.

From the moment Trump came to illusory power everything was about him. It wasn't about Republican policy or opinion, but all about Trump. Everything he did was presented in negative, derogatory and abusive terms by the Sabbatian-dominated media led by Cult operations such as CNN, MSNBC, *The New York Times* and the Jeff Bezos-owned *Washington Post* – 'Pick the target, freeze it, personalize it, polarize it.' Trump was turned into a demon to be vilified by those who hated him and a demi-god loved by those who worshipped him. This, in turn, had his supporters, too, presented as equally demonic in preparation for the punchline later down the line when Biden was about to take office. It was here's a Trump, there's a Trump, everywhere a Trump, Trump. Virtually every news story or happening was filtered through the lens of 'The Donald'. You loved him or hated him and which one you chose was said to define you as Satan's spawn or a paragon of virtue. Even supporting some Trump policies or statements and not others was enough for an assault on your character. No shades of grey were or are allowed. Everything is black and white (literally and figuratively). A Californian I knew had her head utterly scrambled by her hatred for Trump while telling people they should love each other. She was so totally consumed by

Trump Derangement Syndrome as it became to be known that this glaring contradiction would never have occurred to her. By definition anyone who criticised Trump or praised his opponents was a hero and this lady described Joe Biden as 'a kind, honest gentleman' when he's a provable liar, mega-crook and vicious piece of work to boot. Sabbatians had indeed divided America using Trump as the fall-guy and all along the clock was ticking on the consequences for his supporters.

In hock to his masters

Trump gave Sabbatians via Israel almost everything they wanted in his four years. Ask and you shall receive was the dynamic between himself and Benjamin Netanyahu orchestrated by Trump's ultra-Zionist son-in-law Jared Kushner, his ultra-Zionist Ambassador to Israel, David Friedman, and ultra-Zionist 'Israel adviser', Jason Greenblatt. The last two were central to the running and protecting from collapse of his business empire, the Trump Organisation, and colossal business failures made him forever beholding to Sabbatian networks that bailed him out. By the start of the 1990s Trump owed $4 billion to banks that he couldn't pay and almost $1billion of that was down to him personally and not his companies. This mega-disaster was the result of building two new casinos in Atlantic City and buying the enormous Taj Mahal operation which led to crippling debt payments. He had borrowed fantastic sums from 72 banks with major Sabbatian connections and although the scale of debt should have had him living in a tent alongside the highway they never foreclosed. A plan was devised to lift Trump from the mire by BT Securities Corporation and Rothschild Inc. and the case was handled by Wilber Ross who had worked for the Rothschilds for 27 years. Ross would be named US Commerce Secretary after Trump's election. Another crucial figure in saving Trump was ultra-Zionist 'investor' Carl Icahn who bought the Taj Mahal casino. Icahn was made special economic adviser on financial regulation in the Trump administration. He didn't stay long but still managed to find time to make a tidy sum of a reported $31.3 million when he sold his

holdings affected by the price of steel three days before Trump imposed a 235 percent tariff on steel imports. What amazing bits of luck these people have. Trump and Sabbatian operatives have long had a close association and his mentor and legal adviser from the early 1970s until 1986 was the dark and genetically corrupt ultra-Zionist Roy Cohn who was chief counsel to Senator Joseph McCarthy's 'communist' witch-hunt in the 1950s. *Esquire* magazine published an article about Cohn with the headline 'Don't mess with Roy Cohn'. He was described as the most feared lawyer in New York and 'a ruthless master of dirty tricks ... [with] ... more than one Mafia Don on speed dial'. Cohn's influence, contacts, support and protection made Trump a front man for Sabbatians in New York with their connections to one of Cohn's many criminal employers, the 'Russian' Sabbatian Mafia. Israel-centric media mogul Rupert Murdoch was introduced to Trump by Cohn and they started a long friendship. Cohn died in 1986 weeks after being disbarred for unethical conduct by the Appellate Division of the New York State Supreme Court. The wheels of justice do indeed run slow given the length of Cohn's crooked career.

QAnon-sense

We are asked to believe that Donald Trump with his fundamental connections to Sabbatian networks and operatives has been leading the fight to stop the Sabbatian agenda for the fascistic control of America and the world. Sure he has. A man entrapped during his years in the White House by Sabbatian operatives and whose biggest financial donor was casino billionaire Sheldon Adelson who was Sabbatian to his DNA?? Oh, do come on. Trump has been used to divide America and isolate Pushbackers on the Cult agenda under the heading of 'Trump supporters', 'insurrectionists' and 'white supremacists'. The US Intelligence/Mossad Psyop or psychological operation known as QAnon emerged during the Trump years as a central pillar in the Sabbatian campaign to lead Pushbackers into the trap set by those that wished to destroy them. I knew from the start that QAnon was a scam because I had seen the same scenario many

times before over 30 years under different names and I had written about one in particular in the books. 'Not again' was my reaction when QAnon came to the fore. The same script is pulled out every few years and a new name added to the letterhead. The story always takes the same form: 'Insiders' or 'the good guys' in the government-intelligence-military 'Deep State' apparatus were going to instigate mass arrests of the 'bad guys' which would include the Rockefellers, Rothschilds, Barack Obama, Hillary Clinton, George Soros, etc., etc. Dates are given for when the 'good guys' are going to move in, but the dates pass without incident and new dates are given which pass without incident. The central message to Pushbackers in each case is that they don't have to do anything because there is 'a plan' and it is all going to be sorted by the 'good guys' on the inside. 'Trust the plan' was a QAnon mantra when the only plan was to misdirect Pushbackers into putting their trust in a Psyop they believed to be real. Beware, beware, those who tell you what you want to hear and always check it out. Right up to Biden's inauguration QAnon was still claiming that 'the Storm' was coming and Trump would stay on as president when Biden and his cronies were arrested and jailed. It was never going to happen and of course it didn't, but what did happen as a result provided that punchline to the Sabbatian Trump/QAnon Psyop.

On January 6th, 2021, a very big crowd of Trump supporters gathered in the National Mall in Washington DC down from the Capitol Building to protest at what they believed to be widespread corruption and vote fraud that stopped Trump being re-elected for a second term as president in November, 2020. I say as someone that does not support Trump or Biden that the evidence is clear that major vote-fixing went on to favour Biden, a man with cognitive problems so advanced he can often hardly string a sentence together without reading the words written for him on the Teleprompter. Glaring ballot discrepancies included serious questions about electronic voting machines that make vote rigging a comparative cinch and hundreds of thousands of paper votes that suddenly appeared during already advanced vote counts and virtually all of

them for Biden. Early Trump leads in crucial swing states suddenly began to close and disappear. The pandemic hoax was used as the excuse to issue almost limitless numbers of mail-in ballots with no checks to establish that the recipients were still alive or lived at that address. They were sent to streams of people who had not even asked for them. Private organisations were employed to gather these ballots and who knows what they did with them before they turned up at the counts. The American election system has been manipulated over decades to become a sick joke with more holes than a Swiss cheese for the express purpose of dictating the results. Then there was the criminal manipulation of information by Sabbatian tech giants like Facebook, Twitter and Google-owned YouTube which deleted pro-Trump, anti-Biden accounts and posts while everything in support of Biden was left alone. Sabbatians wanted Biden to win because after the dividing of America it was time for full-on Woke and every aspect of the Cult agenda to be unleashed.

Hunter gatherer

Extreme Silicon Valley bias included blocking information by the *New York Post* exposing a Biden scandal that should have ended his bid for president in the final weeks of the campaign. Hunter Biden, his monumentally corrupt son, is reported to have sent a laptop to be repaired at a local store and failed to return for it. Time passed until the laptop became the property of the store for non-payment of the bill. When the owner saw what was on the hard drive he gave a copy to the FBI who did nothing even though it confirmed widespread corruption in which the Joe Biden family were using his political position, especially when he was vice president to Obama, to make multiple millions in countries around the world and most notably Ukraine and China. Hunter Biden's one-time business partner Tony Bobulinski went public when the story broke in the *New York Post* to confirm the corruption he saw and that Joe Biden not only knew what was going on he also profited from the spoils. Millions were handed over by a Chinese company with close

connections – like all major businesses in China – to the Chinese communist party of President Xi Jinping. Joe Biden even boasted at a meeting of the Cult's World Economic Forum that as vice president he had ordered the government of Ukraine to fire a prosecutor. What he didn't mention was that the same man just happened to be investigating an energy company which was part of Hunter Biden's corrupt portfolio. The company was paying him big bucks for no other reason than the influence his father had. Overnight Biden's presidential campaign should have been over given that he had lied publicly about not knowing what his son was doing. Instead almost the entire Sabbatian-owned mainstream media and Sabbatian-owned Silicon Valley suppressed circulation of the story. This alone went a mighty way to rigging the election of 2020. Cult assets like Mark Zuckerberg at Facebook also spent hundreds of millions to be used in support of Biden and vote 'administration'.

The Cult had used Trump as the focus to divide America and was now desperate to bring in moronic, pliable, corrupt Biden to complete the double-whammy. No way were they going to let little things like the will of the people thwart their plan. Silicon Valley widely censored claims that the election was rigged because it *was* rigged. For the same reason anyone claiming it was rigged was denounced as a 'white supremacist' including the pathetically few Republican politicians willing to say so. Right across the media where the claim was mentioned it was described as a 'false claim' even though these excuses for 'journalists' would have done no research into the subject whatsoever. Trump won seven million more votes than any sitting president had ever achieved while somehow a cognitively-challenged soon to be 78-year-old who was hidden away from the public for most of the campaign managed to win more votes than any presidential candidate in history. It makes no sense. You only had to see election rallies for both candidates to witness the enthusiasm for Trump and the apathy for Biden. Tens of thousands would attend Trump events while Biden was speaking in empty car parks with often only television crews attending and framing their shots to hide the fact that no one was there. It was pathetic to see

footage come to light of Biden standing at a podium making speeches only to TV crews and party fixers while reading the words written for him on massive Teleprompter screens. So, yes, those protestors on January 6th had a point about election rigging, but some were about to walk into a trap laid for them in Washington by the Cult Deep State and its QAnon Psyop. This was the Capitol Hill riot ludicrously dubbed an 'insurrection'.

The spider and the fly

Renegade Minds know there are not two 'sides' in politics, only one side, the Cult, working through all 'sides'. It's a stage show, a puppet show, to direct the perceptions of the population into focusing on diversions like parties and candidates while missing the puppeteers with their hands holding all the strings. The Capitol Hill 'insurrection' brings us back to the Little Big Horn. Having created two distinct opposing groupings – Woke and Pushbackers – the trap was about to be sprung. Pushbackers were to be encircled and isolated by associating them all in the public mind with Trump and then labelling Trump as some sort of Confederate leader. I knew immediately that the Capitol riot was a set-up because of two things. One was how easy the rioters got into the building with virtually no credible resistance and secondly I could see – as with the 'Covid' hoax in the West at the start of 2020 – how the Cult could exploit the situation to move its agenda forward with great speed. My experience of Cult techniques and activities over more than 30 years has showed me that while they do exploit situations they haven't themselves created this never happens with events of fundamental agenda significance. Every time major events giving cultists the excuse to rapidly advance their plan you find they are manipulated into being for the specific reason of providing that excuse – Problem-Reaction-Solution. Only a tiny minority of the huge crowd of Washington protestors sought to gain entry to the Capitol by smashing windows and breaching doors. That didn't matter. The whole crowd and all Pushbackers, even if they did not support Trump, were going to be lumped together as dangerous

insurrectionists and conspiracy theorists. The latter term came into widespread use through a CIA memo in the 1960s aimed at discrediting those questioning the nonsensical official story of the Kennedy assassination and it subsequently became widely employed by the media. It's still being used by inept 'journalists' with no idea of its origin to discredit anyone questioning anything that authority claims to be true. When you are perpetrating a conspiracy you need to discredit the very word itself even though the dictionary definition of conspiracy is merely 'the activity of secretly planning with other people to do something bad or illegal' and 'a general agreement to keep silent about a subject for the purpose of keeping it secret'. On that basis there are conspiracies almost wherever you look. For obvious reasons the Cult and its lapdog media have to claim there are no conspiracies even though the word appears in state laws as with conspiracy to defraud, to murder, and to corrupt public morals.

Agent provocateurs are widely used by the Cult Deep State to manipulate genuine people into acting in ways that suit the desired outcome. By genuine in this case I mean protestors genuinely supporting Trump and claims that the election was stolen. In among them, however, were agents of the state wearing the garb of Trump supporters and QAnon to pump-prime the Capital riot which some genuine Trump supporters naively fell for. I described the situation as 'Come into my parlour said the spider to the fly'. Leaflets appeared through the Woke paramilitary arm Antifa, the anti-fascist fascists, calling on supporters to turn up in Washington looking like Trump supporters even though they hated him. Some of those arrested for breaching the Capitol Building were sourced to Antifa and its stable mate Black Lives Matter. Both organisations are funded by Cult billionaires and corporations. One man charged for the riot was according to his lawyer a former FBI agent who had held top secret security clearance for 40 years. Attorney Thomas Plofchan said of his client, 66-year-old Thomas Edward Caldwell:

He has held a Top Secret Security Clearance since 1979 and has undergone multiple Special Background Investigations in support of his clearances. After retiring from the Navy, he

worked as a section chief for the Federal Bureau of Investigation from 2009-2010 as a GS-12 [mid-level employee].

He also formed and operated a consulting firm performing work, often classified, for U.S government customers including the US. Drug Enforcement Agency, Department of Housing and Urban Development, the US Coast Guard, and the US Army Personnel Command.

A judge later released Caldwell pending trial in the absence of evidence about a conspiracy or that he tried to force his way into the building. *The New York Post* reported a 'law enforcement source' as saying that 'at least two known Antifa members were spotted' on camera among Trump supporters during the riot while one of the rioters arrested was John Earle Sullivan, a seriously extreme Black Lives Matter Trump-hater from Utah who was previously arrested and charged in July, 2020, over a BLM-Antifa riot in which drivers were threatened and one was shot. Sullivan is the founder of Utah-based Insurgence USA which is an affiliate of the Cult-created-and-funded Black Lives Matter movement. Footage appeared and was then deleted by Twitter of Trump supporters calling out Antifa infiltrators and a group was filmed changing into pro-Trump clothing before the riot. Security at the building was *pathetic* – as planned. Colonel Leroy Fletcher Prouty, a man with long experience in covert operations working with the US security apparatus, once described the tell-tale sign to identify who is involved in an assassination. He said:

No one has to direct an assassination – it happens. The active role is played secretly by permitting it to happen. This is the greatest single clue. Who has the power to call off or reduce the usual security precautions?

This principle applies to many other situations and certainly to the Capitol riot of January 6th, 2021.

The sting

With such a big and potentially angry crowd known to be gathering near the Capitol the security apparatus would have had a major police detail to defend the building with National Guard troops on

standby given the strength of feeling among people arriving from all over America encouraged by the QAnon Psyop and statements by Donald Trump. Instead Capitol Police 'security' was flimsy, weak, and easily breached. The same number of officers was deployed as on a regular day and that is a blatant red flag. They were not staffed or equipped for a possible riot that had been an obvious possibility in the circumstances. No protective and effective fencing worth the name was put in place and there were no contingency plans. The whole thing was basically a case of standing aside and waving people in. Once inside police mostly backed off apart from one Capitol police officer who ridiculously shot dead unarmed Air Force veteran protestor Ashli Babbitt without a warning as she climbed through a broken window. The 'investigation' refused to name or charge the officer after what must surely be considered a murder in the circumstances. They just lifted a carpet and swept. The story was endlessly repeated about five people dying in the 'armed insurrection' when there was no report of rioters using weapons. Apart from Babbitt the other four died from a heart attack, strokes and apparently a drug overdose. Capitol police officer Brian Sicknick was reported to have died after being bludgeoned with a fire extinguisher when he was alive after the riot was over and died later of what the Washington Medical Examiner's Office said was a stroke. Sicknick had no external injuries. The lies were delivered like rapid fire. There was a narrative to build with incessant repetition of the lie until the lie became the accepted 'everybody knows that' truth. The 'Big Lie' technique of Nazi Propaganda Minister Joseph Goebbels is constantly used by the Cult which was behind the Nazis and is today behind the 'Covid' and 'climate change' hoaxes. Goebbels said:

> If you tell a lie big enough and keep repeating it, people will eventually come to believe it. The lie can be maintained only for such time as the State can shield the people from the political, economic and/or military consequences of the lie. It thus becomes vitally important for the State to use all of its powers to repress dissent, for the truth is the mortal enemy of the lie, and thus by extension, the truth is the greatest enemy of the State.

Most protestors had a free run of the Capitol Building. This allowed pictures to be taken of rioters in iconic parts of the building including the Senate chamber which could be used as propaganda images against all Pushbackers. One Congresswoman described the scene as 'the worst kind of non-security anybody could ever imagine'. Well, the first part was true, but someone obviously did imagine it and made sure it happened. Some photographs most widely circulated featured people wearing QAnon symbols and now the Psyop would be used to dub all QAnon followers with the ubiquitous fit-all label of 'white supremacist' and 'insurrectionists'. When a Muslim extremist called Noah Green drove his car at two police officers at the Capitol Building killing one in April, 2021, there was no such political and media hysteria. They were just disappointed he wasn't white.

The witch-hunt

Government prosecutor Michael Sherwin, an aggressive, dark-eyed, professional Rottweiler led the 'investigation' and to call it over the top would be to understate reality a thousand fold. Hundreds were tracked down and arrested for the crime of having the wrong political views and people were jailed who had done nothing more than walk in the building, committed no violence or damage to property, took a few pictures and left. They were labelled a 'threat to the Republic' while Biden sat in the White House signing executive orders written for him that were dismantling 'the Republic'. Even when judges ruled that a mother and son should not be in jail the government kept them there. Some of those arrested have been badly beaten by prison guards in Washington and lawyers for one man said he suffered a fractured skull and was made blind in one eye. Meanwhile a woman is shot dead for no reason by a Capitol Police officer and we are not allowed to know who he is never mind what has happened to him although that will be *nothing*. The Cult's QAnon/Trump sting to identify and isolate Pushbackers and then target them on the road to crushing and deleting them was a resounding success. You would have thought the Russians had

invaded the building at gunpoint and lined up senators for a firing squad to see the political and media reaction. Congresswoman Alexandria Ocasio-Cortez is a child in a woman's body, a terrible-twos, me, me, me, Woker narcissist of such proportions that words have no meaning. She said she thought she was going to die when 'insurrectionists' banged on her office door. It turned out she wasn't even in the Capitol Building when the riot was happening and the 'banging' was a Capitol Police officer. She referred to herself as a 'survivor' which is an insult to all those true survivors of violent and sexual abuse while she lives her pampered and privileged life talking drivel for a living. Her Woke colleague and fellow mega-narcissist Rashida Tlaib broke down describing the devastating effect on her, too, of *not being* in the building when the rioters were there. Ocasio-Cortez and Tlaib are members of a fully-Woke group of Congresswomen known as 'The Squad' along with Ilhan Omar and Ayanna Pressley. The Squad from what I can see can be identified by its vehement anti-white racism, anti-white men agenda, and, as always in these cases, the absence of brain cells on active duty.

The usual suspects were on the riot case immediately in the form of Democrat ultra-Zionist senators and operatives Chuck Schumer and Adam Schiff demanding that Trump be impeached for 'his part in the insurrection'. The same pair of prats had led the failed impeachment of Trump over the invented 'Russia collusion' nonsense which claimed Russia had helped Trump win the 2016 election. I didn't realise that Tel Aviv had been relocated just outside Moscow. I must find an up-to-date map. The Russia hoax was a Sabbatian operation to keep Trump occupied and impotent and to stop any rapport with Russia which the Cult wants to retain as a perceptual enemy to be pulled out at will. Puppet Biden began attacking Russia when he came to office as the Cult seeks more upheaval, division and war across the world. A two-year stage show 'Russia collusion inquiry' headed by the not-very-bright former 9/11 FBI chief Robert Mueller, with support from 19 lawyers, 40 FBI agents plus intelligence analysts, forensic accountants and other

staff, devoured tens of millions of dollars and found no evidence of Russia collusion which a ten-year-old could have told them on day one. Now the same moronic Schumer and Schiff wanted a second impeachment of Trump over the Capitol 'insurrection' (riot) which the arrested development of Schumer called another 'Pearl Harbor' while others compared it with 9/11 in which 3,000 died and, in the case of CNN, with the Rwandan genocide in the 1990s in which an estimated 500,000 to 600,000 were murdered, between 250, 000 and 500,000 women were raped, and populations of whole towns were hacked to death with machetes. To make those comparisons purely for Cult political reasons is beyond insulting to those that suffered and lost their lives and confirms yet again the callous inhumanity that we are dealing with. Schumer is a monumental idiot and so is Schiff, but they serve the Cult agenda and do whatever they're told so they get looked after. Talking of idiots – another inane man who spanned the Russia and Capitol impeachment attempts was Senator Eric Swalwell who had the nerve to accuse Trump of collusion with the Russians while sleeping with a Chinese spy called Christine Fang or 'Fang Fang' which is straight out of a Bond film no doubt starring Klaus Schwab as the bloke living on a secret island and controlling laser weapons positioned in space and pointing at world capitals. Fang Fang plays the part of Bond's infiltrator girlfriend which I'm sure she would enjoy rather more than sharing a bed with the brainless Swalwell, lying back and thinking of China. The FBI eventually warned Swalwell about Fang Fang which gave her time to escape back to the Chinese dictatorship. How very thoughtful of them. The second Trump impeachment also failed and hardly surprising when an impeachment is supposed to remove a sitting president and by the time it happened Trump was no longer president. These people are running your country America, well, officially anyway. Terrifying isn't it?

Outcomes tell the story - always

The outcome of all this – and it's the *outcome* on which Renegade Minds focus, not the words – was that a vicious, hysterical and

obviously pre-planned assault was launched on Pushbackers to censor, silence and discredit them and even targeted their right to earn a living. They have since been condemned as 'domestic terrorists' that need to be treated like Al-Qaeda and Islamic State. 'Domestic terrorists' is a label the Cult has been trying to make stick since the period of the Oklahoma bombing in 1995 which was blamed on 'far-right domestic terrorists'. If you read *The Trigger* you will see that the bombing was clearly a Problem-Reaction-Solution carried out by the Deep State during a Bill Clinton administration so corrupt that no dictionary definition of the term would even nearly suffice. Nearly 30, 000 troops were deployed from all over America to the empty streets of Washington for Biden's inauguration. Ten thousand of them stayed on with the pretext of protecting the capital from insurrectionists when it was more psychological programming to normalise the use of the military in domestic law enforcement in support of the Cult plan for a police-military state. Biden's fascist administration began a purge of 'wrong-thinkers' in the military which means anyone that is not on board with Woke. The Capitol Building was surrounded by a fence with razor wire and the Land of the Free was further symbolically and literally dismantled. The circle was completed with the installation of Biden and the exploitation of the QAnon Psyop.

America had never been so divided since the civil war of the 19th century, Pushbackers were isolated and dubbed terrorists and now, as was always going to happen, the Cult immediately set about deleting what little was left of freedom and transforming American society through a swish of the hand of the most controlled 'president' in American history leading (officially at least) the most extreme regime since the country was declared an independent state on July 4th, 1776. Biden issued undebated, dictatorial executive orders almost by the hour in his opening days in office across the whole spectrum of the Cult wish-list including diluting controls on the border with Mexico allowing thousands of migrants to illegally enter the United States to transform the demographics of America and import an election-changing number of perceived Democrat

voters. Then there were Biden deportation amnesties for the already illegally resident (estimated to be as high as 20 or even 30 million). A bill before Congress awarded American citizenship to anyone who could prove they had worked in agriculture for just 180 days in the previous two years as 'Big Ag' secured its slave labour long-term. There were the plans to add new states to the union such as Puerto Rico and making Washington DC a state. They are all parts of a plan to ensure that the Cult-owned Woke Democrats would be permanently in power.

Border – what border?

I have exposed in detail in other books how mass immigration into the United States and Europe is the work of Cult networks fuelled by the tens of billions spent to this and other ends by George Soros and his global Open Society (open borders) Foundations. The impact can be seen in America alone where the population has increased by *100 million* in little more than 30 years mostly through immigration. I wrote in *The Answer* that the plan was to have so many people crossing the southern border that the numbers become unstoppable and we are now there under Cult-owned Biden. El Salvador in Central America puts the scale of what is happening into context. A third of the population now lives in the United States, much of it illegally, and many more are on the way. The methodology is to crush Central and South American countries economically and spread violence through machete-wielding psychopathic gangs like MS-13 based in El Salvador and now operating in many American cities. Biden-imposed lax security at the southern border means that it is all but open. He said before his 'election' that he wanted to see a surge towards the border if he became president and that was the green light for people to do just that after election day to create the human disaster that followed for both America and the migrants. When that surge came the imbecilic Alexandria Ocasio-Cortez said it wasn't a 'surge' because they are 'children, not insurgents' and the term 'surge' (used by Biden) was a claim of 'white supremacists'.

This disingenuous lady may one day enter the realm of the most basic intelligence, but it won't be any time soon.

Sabbatians and the Cult are in the process of destroying America by importing violent people and gangs in among the genuine to terrorise American cities and by overwhelming services that cannot cope with the sheer volume of new arrivals. Something similar is happening in Europe as Western society in general is targeted for demographic and cultural transformation and upheaval. The plan demands violence and crime to create an environment of intimidation, fear and division and Soros has been funding the election of district attorneys across America who then stop prosecuting many crimes, reduce sentences for violent crimes and free as many violent criminals as they can. Sabbatians are creating the chaos from which order – their order – can respond in a classic Problem-Reaction-Solution. A Freemasonic moto says 'Ordo Ab Chao' (Order out of Chaos) and this is why the Cult is constantly creating chaos to impose a new 'order'. Here you have the reason the Cult is constantly creating chaos. The 'Covid' hoax can be seen with those entering the United States by plane being forced to take a 'Covid' test while migrants flooding through southern border processing facilities do not. Nothing is put in the way of mass migration and if that means ignoring the government's own 'Covid' rules then so be it. They know it's all bullshit anyway. Any pushback on this is denounced as 'racist' by Wokers and Sabbatian fronts like the ultra-Zionist Anti-Defamation League headed by the appalling Jonathan Greenblatt which at the same time argues that Israel should not give citizenship and voting rights to more Palestinian Arabs or the 'Jewish population' (in truth the Sabbatian network) will lose control of the country.

Society-changing numbers

Biden's masters have declared that countries like El Salvador are so dangerous that their people must be allowed into the United States for humanitarian reasons when there are fewer murders in large parts of many Central American countries than in US cities like

Baltimore. That is not to say Central America cannot be a dangerous place and Cult-controlled American governments have been making it so since way back, along with the dismantling of economies, in a long-term plan to drive people north into the United States. Parts of Central America are very dangerous, but in other areas the story is being greatly exaggerated to justify relaxing immigration criteria. Migrants are being offered free healthcare and education in the United States as another incentive to head for the border and there is no requirement to be financially independent before you can enter to prevent the resources of America being drained. You can't blame migrants for seeking what they believe will be a better life, but they are being played by the Cult for dark and nefarious ends. The numbers since Biden took office are huge. In February, 2021, more than 100,000 people were known to have tried to enter the US illegally through the southern border (it was 34,000 in the same month in 2020) and in March it was 170,000 – a 418 percent increase on March, 2020. These numbers are only known people, not the ones who get in unseen. The true figure for migrants illegally crossing the border in a single month was estimated by one congressman at 250,000 and that number will only rise under Biden's current policy. Gangs of murdering drug-running thugs that control the Mexican side of the border demand money – thousands of dollars – to let migrants cross the Rio Grande into America. At the same time gun battles are breaking out on the border several times a week between rival Mexican drug gangs (which now operate globally) who are equipped with sophisticated military-grade weapons, grenades and armoured vehicles. While the Capitol Building was being 'protected' from a non-existent 'threat' by thousands of troops, and others were still deployed at the time in the Cult Neocon war in Afghanistan, the southern border of America was left to its fate. This is not incompetence, it is cold calculation.

By March, 2021, there were 17,000 unaccompanied children held at border facilities and many of them are ensnared by people traffickers for paedophile rings and raped on their journey north to America. This is not conjecture – this is fact. Many of those designated

children are in reality teenage boys or older. Meanwhile Wokers posture their self-purity for encouraging poor and tragic people to come to America and face this nightmare both on the journey and at the border with the disgusting figure of House Speaker Nancy Pelosi giving disingenuous speeches about caring for migrants. The woman's evil. Wokers condemned Trump for having children in cages at the border (so did Obama, *Shhhh*), but now they are sleeping on the floor without access to a shower with one border facility 729 percent over capacity. The Biden insanity even proposed flying migrants from the southern border to the northern border with Canada for 'processing'. The whole shambles is being overseen by ultra-Zionist Secretary of Homeland Security, the moronic liar Alejandro Mayorkas, who banned news cameras at border facilities to stop Americans seeing what was happening. Mayorkas said there was not a ban on news crews; it was just that they were not allowed to film. Alongside him at Homeland Security is another ultra-Zionist Cass Sunstein appointed by Biden to oversee new immigration laws. Sunstein despises conspiracy researchers to the point where he suggests they should be banned or *taxed* for having such views. The man is not bonkers or anything. He's perfectly well-adjusted, but adjusted to what is the question. Criticise what is happening and you are a 'white supremacist' when earlier non-white immigrants also oppose the numbers which effect their lives and opportunities. Black people in poor areas are particularly damaged by uncontrolled immigration and the increased competition for work opportunities with those who will work for less. They are also losing voting power as Hispanics become more dominant in former black areas. It's a downward spiral for them while the billionaires behind the policy drone on about how much they care about black people and 'racism'. None of this is about compassion for migrants or black people – that's just wind and air. Migrants are instead being mercilessly exploited to transform America while the countries they leave are losing their future and the same is true in Europe. Mass immigration may now be the work of Woke Democrats, but it can be traced back to the 1986 Immigration Reform and Control Act (it

wasn't) signed into law by Republican hero President Ronald Reagan which gave amnesty to millions living in the United States illegally and other incentives for people to head for the southern border. Here we have the one-party state at work again.

Save me syndrome

Almost every aspect of what I have been exposing as the Cult agenda was on display in even the first days of 'Biden' with silencing of Pushbackers at the forefront of everything. A Renegade Mind will view the Trump years and QAnon in a very different light to their supporters and advocates as the dots are connected. The QAnon/Trump Psyop has given the Cult all it was looking for. We may not know how much, or little, that Trump realised he was being used, but that's a side issue. This pincer movement produced the desired outcome of dividing America and having Pushbackers isolated. To turn this around we have to look at new routes to empowerment which do not include handing our power to other people and groups through what I will call the 'Save Me Syndrome' – 'I want someone else to do it so that I don't have to'. We have seen this at work throughout human history and the QAnon/Trump Psyop is only the latest incarnation alongside all the others. Religion is an obvious expression of this when people look to a 'god' or priest to save them or tell them how to be saved and then there are 'save me' politicians like Trump. Politics is a diversion and not a 'saviour'. It is a means to block positive change, not make it possible.

Save Me Syndrome always comes with the same repeating theme of handing your power to whom or what you believe will save you while your real 'saviour' stares back from the mirror every morning. Renegade Minds are constantly vigilant in this regard and always asking the question 'What can I do?' rather than 'What can someone else do for me?' Gandhi was right when he said: 'You must be the change you want to see in the world.' We are indeed the people we have been waiting for. We are presented with a constant raft of reasons to concede that power to others and forget where the real power is. Humanity has the numbers and the Cult does not. It has to

use diversion and division to target the unstoppable power that comes from unity. Religions, governments, politicians, corporations, media, QAnon, are all different manifestations of this power-diversion and dilution. Refusing to give your power to governments and instead handing it to Trump and QAnon is not to take a new direction, but merely to recycle the old one with new names on the posters. I will explore this phenomenon as we proceed and how to break the cycles and recycles that got us here through the mists of repeating perception and so repeating history.

For now we shall turn to the most potent example in the entire human story of the consequences that follow when you give your power away. I am talking, of course, of the 'Covid' hoax.

CHAPTER FOUR

'Covid': Calculated catastrophe

Facts are threatening to those invested in fraud
DaShanne Stokes

We can easily unravel the real reason for the 'Covid pandemic' hoax by employing the Renegade Mind methodology that I have outlined this far. We'll start by comparing the long-planned Cult outcome with the 'Covid pandemic' outcome. Know the outcome and you'll see the journey.

I have highlighted the plan for the Hunger Games Society which has been in my books for so many years with the very few controlling the very many through ongoing dependency. To create this dependency it is essential to destroy independent livelihoods, businesses and employment to make the population reliant on the state (the Cult) for even the basics of life through a guaranteed pittance income. While independence of income remained these Cult ambitions would be thwarted. With this knowledge it was easy to see where the 'pandemic' hoax was going once talk of 'lockdowns' began and the closing of all but perceived 'essential' businesses to 'save' us from an alleged 'deadly virus'. Cult corporations like Amazon and Walmart were naturally considered 'essential' while mom and pop shops and stores had their doors closed by fascist decree. As a result with every new lockdown and new regulation more small and medium, even large businesses not owned by the Cult, went to the wall while Cult giants and their frontmen and women grew financially fatter by the second. Mom and pop were

denied an income and the right to earn a living and the wealth of people like Jeff Bezos (Amazon), Mark Zuckerberg (Facebook) and Sergei Brin and Larry Page (Google/Alphabet) have reached record levels. The Cult was increasing its own power through further dramatic concentrations of wealth while the competition was being destroyed and brought into a state of dependency. Lockdowns have been instigated to secure that very end and were never anything to do with health. My brother Paul spent 45 years building up a bus repair business, but lockdowns meant buses were running at a fraction of normal levels for months on end. Similar stories can told in their hundreds of millions worldwide. Efforts of a lifetime coldly destroyed by Cult multi-billionaires and their lackeys in government and law enforcement who continued to earn their living from the taxation of the people while denying the right of the same people to earn theirs. How different it would have been if those making and enforcing these decisions had to face the same financial hardships of those they affected, but they never do.

Gates of Hell

Behind it all in the full knowledge of what he is doing and why is the psychopathic figure of Cult operative Bill Gates. His puppet Tedros at the World Health Organization declared 'Covid' a pandemic in March, 2020. The WHO had changed the definition of a 'pandemic' in 2009 just a month before declaring the 'swine flu pandemic' which would not have been so under the previous definition. The same applies to 'Covid'. The definition had included… 'an infection by an infectious agent, occurring simultaneously in different countries, with a significant mortality rate relative to the proportion of the population infected'. The new definition removed the need for 'significant mortality'. The 'pandemic' has been fraudulent even down to the definition, but Gates demanded economy-destroying lockdowns, school closures, social distancing, mandatory masks, a 'vaccination' for every man, woman and child on the planet and severe consequences and restrictions for those that refused. Who gave him this power? The

Cult did which he serves like a little boy in short trousers doing what his daddy tells him. He and his psychopathic missus even smiled when they said that much worse was to come (what they knew was planned to come). Gates responded in the matter-of-fact way of all psychopaths to a question about the effect on the world economy of what he was doing:

> Well, it won't go to zero but it will shrink. Global GDP is probably going to take the biggest hit ever [Gates was smiling as he said this] … in my lifetime this will be the greatest economic hit. But you don't have a choice. People act as if you have a choice. People don't feel like going to the stadium when they might get infected … People are deeply affected by seeing these stats, by knowing they could be part of the transmission chain, old people, their parents and grandparents, could be affected by this, and so you don't get to say ignore what is going on here.
>
> There will be the ability to open up, particularly in rich countries, if things are done well over the next few months, but for the world at large normalcy only returns when we have largely vaccinated the entire population.

The man has no compassion or empathy. How could he when he's a psychopath like all Cult players? My own view is that even beyond that he is very seriously mentally ill. Look in his eyes and you can see this along with his crazy flailing arms. You don't do what he has done to the world population since the start of 2020 unless you are mentally ill and at the most extreme end of psychopathic. You especially don't do it when to you know, as we shall see, that cases and deaths from 'Covid' are fakery and a product of monumental figure massaging. 'These stats' that Gates referred to are based on a 'test' that's not testing for the 'virus' as he has known all along. He made his fortune with big Cult support as an infamously ruthless software salesman and now buys global control of 'health' (death) policy without the population he affects having any say. It's a breathtaking outrage. Gates talked about people being deeply affected by fear of 'Covid' when that was because of *him* and his global network lying to them minute-by-minute supported by a lying media that he seriously influences and funds to the tune of hundreds of millions. He's handed big sums to media operations including the BBC, NBC, Al Jazeera, Univision, *PBS NewsHour*,

ProPublica, National Journal, The Guardian, The Financial Times, The Atlantic, Texas Tribune, USA Today publisher Gannett, *Washington Monthly, Le Monde,* Center for Investigative Reporting, Pulitzer Center on Crisis Reporting, National Press Foundation, International Center for Journalists, Solutions Journalism Network, the Poynter Institute for Media Studies, and many more. Gates is everywhere in the 'Covid' hoax and the man must go to prison – or a mental facility – for the rest of his life and his money distributed to those he has taken such enormous psychopathic pleasure in crushing.

The Muscle

The Hunger Games global structure demands a police-military state – a fusion of the two into one force – which viciously imposes the will of the Cult on the population and protects the Cult from public rebellion. In that regard, too, the 'Covid' hoax just keeps on giving. Often unlawful, ridiculous and contradictory 'Covid' rules and regulations have been policed across the world by moronic automatons and psychopaths made faceless by face-nappy masks and acting like the Nazi SS and fascist blackshirts and brownshirts of Hitler and Mussolini. The smallest departure from the rules decreed by the psychos in government and their clueless gofers were jumped upon by the face-nappy fascists. Brutality against public protestors soon became commonplace even on girls, women and old people as the brave men with the batons – the Face-Nappies as I call them – broke up peaceful protests and handed out fines like confetti to people who couldn't earn a living let alone pay hundreds of pounds for what was once an accepted human right. Robot Face-Nappies of Nottingham police in the English East Midlands fined one group £11,000 for attending a child's birthday party. For decades I charted the transformation of law enforcement as genuine, decent officers were replaced with psychopaths and the brain dead who would happily and brutally do whatever their masters told them. Now they were let loose on the public and I would emphasise the point that none of this just happened. The step-by-step change in the dynamic between police and public was orchestrated from the shadows by

those who knew where this was all going and the same with the perceptual reframing of those in all levels of authority and official administration through 'training courses' by organisations such as Common Purpose which was created in the late 1980s and given a massive boost in Blair era Britain until it became a global phenomenon. Supposed public 'servants' began to view the population as the enemy and the same was true of the police. This was the start of the explosion of behaviour manipulation organisations and networks preparing for the all-war on the human psyche unleashed with the dawn of 2020. I will go into more detail about this later in the book because it is a core part of what is happening.

Police desecrated beauty spots to deter people gathering and arrested women for walking in the countryside alone 'too far' from their homes. We had arrogant, clueless sergeants in the Isle of Wight police where I live posting on Facebook what they insisted the population must do or else. A schoolmaster sergeant called Radford looked young enough for me to ask if his mother knew he was out, but he was posting what he *expected* people to do while a Sergeant Wilkinson boasted about fining lads for meeting in a McDonald's car park where they went to get a lockdown takeaway. Wilkinson added that he had even cancelled their order. What a pair of prats these people are and yet they have increasingly become the norm among Jackboot Johnson's Yellowshirts once known as the British police. This was the theme all over the world with police savagery common during lockdown protests in the United States, the Netherlands, and the fascist state of Victoria in Australia under its tyrannical and again moronic premier Daniel Andrews. Amazing how tyrannical and moronic tend to work as a team and the same combination could be seen across America as arrogant, narcissistic Woke governors and mayors such as Gavin Newsom (California), Andrew Cuomo (New York), Gretchen Whitmer (Michigan), Lori Lightfoot (Chicago) and Eric Garcetti (Los Angeles) did their Nazi and Stalin impressions with the full support of the compliant brutality of their enforcers in uniform as they arrested small business owners defying

fascist shutdown orders and took them to jail in ankle shackles and handcuffs. This happened to bistro owner Marlena Pavlos-Hackney in Gretchen Whitmer's fascist state of Michigan when police arrived to enforce an order by a state-owned judge for 'putting the community at risk' at a time when other states like Texas were dropping restrictions and migrants were pouring across the southern border without any 'Covid' questions at all. I'm sure there are many officers appalled by what they are ordered to do, but not nearly enough of them. If they were truly appalled they would not do it. As the months passed every opportunity was taken to have the military involved to make their presence on the streets ever more familiar and 'normal' for the longer-term goal of police-military fusion.

Another crucial element to the Hunger Games enforcement network has been encouraging the public to report neighbours and others for 'breaking the lockdown rules'. The group faced with £11,000 in fines at the child's birthday party would have been dobbed-in by a neighbour with a brain the size of a pea. The technique was most famously employed by the Stasi secret police in communist East Germany who had public informants placed throughout the population. A police chief in the UK says his force doesn't need to carry out 'Covid' patrols when they are flooded with so many calls from the public reporting other people for visiting the beach. Dorset police chief James Vaughan said people were so enthusiastic about snitching on their fellow humans they were now operating as an auxiliary arm of the police: 'We are still getting around 400 reports a week from the public, so we will respond to reports …We won't need to be doing hotspot patrols because people are very quick to pick the phone up and tell us.' Vaughan didn't say that this is a pillar of all tyrannies of whatever complexion and the means to hugely extend the reach of enforcement while spreading distrust among the people and making them wary of doing anything that might get them reported. Those narcissistic Isle of Wight sergeants Radford and Wilkinson never fail to add a link to their Facebook posts where the public can inform on their fellow slaves.

Neither would be self-aware enough to realise they were imitating the Stasi which they might well never have heard of. Government psychologists that I will expose later laid out a policy to turn communities against each other in the same way.

A coincidence? Yep, and I can knit fog

I knew from the start of the alleged pandemic that this was a Cult operation. It presented limitless potential to rapidly advance the Cult agenda and exploit manipulated fear to demand that every man, woman and child on the planet was 'vaccinated' in a process never used on humans before which infuses self-replicating *synthetic* material into human cells. Remember the plan to transform the human body from a biological to a synthetic biological state. I'll deal with the 'vaccine' (that's not actually a vaccine) when I focus on the genetic agenda. Enough to say here that mass global 'vaccination' justified by this 'new virus' set alarms ringing after 30 years of tracking these people and their methods. The 'Covid' hoax officially beginning in China was also a big red flag for reasons I will be explaining. The agenda potential was so enormous that I could dismiss any idea that the 'virus' appeared naturally. Major happenings with major agenda implications never occur without Cult involvement in making them happen. My questions were twofold in early 2020 as the media began its campaign to induce global fear and hysteria: Was this alleged infectious agent released on purpose by the Cult or did it even exist at all? I then did what I always do in these situations. I sat, observed and waited to see where the evidence and information would take me. By March and early April synchronicity was strongly – and ever more so since then – pointing me in the direction of *there is no 'virus'*. I went public on that with derision even from swathes of the alternative media that voiced a scenario that the Chinese government released the 'virus' in league with Deep State elements in the United States from a top-level bio-lab in Wuhan where the 'virus' is said to have first appeared. I looked at that possibility, but I didn't buy it for several reasons. Deaths from the 'virus' did not in any way match what they

would have been with a 'deadly bioweapon' and it is much more effective if you sell the *illusion* of an infectious agent rather than having a real one unless you can control through injection who has it and who doesn't. Otherwise you lose control of events. A made-up 'virus' gives you a blank sheet of paper on which you can make it do whatever you like and have any symptoms or mutant 'variants' you choose to add while a real infectious agent would limit you to what it actually does. A phantom disease allows you to have endless ludicrous 'studies' on the 'Covid' dollar to widen the perceived impact by inventing ever more 'at risk' groups including one study which said those who walk slowly may be almost four times more likely to die from the 'virus'. People are in psychiatric wards for less.

A real 'deadly bioweapon' can take out people in the hierarchy that are not part of the Cult, but essential to its operation. Obviously they don't want that. Releasing a real disease means you immediately lose control of it. Releasing an illusory one means you don't. Again it's vital that people are extra careful when dealing with what they want to hear. A bioweapon unleashed from a Chinese laboratory in collusion with the American Deep State may fit a conspiracy narrative, but is it true? Would it not be far more effective to use the excuse of a 'virus' to justify the real bioweapon – the 'vaccine'? That way your disease agent does not have to be transmitted and arrives directly through a syringe. I saw a French virologist Luc Montagnier quoted in the alternative media as saying he had discovered that the alleged 'new' severe acute respiratory syndrome coronavirus , or SARS-CoV-2, was made artificially and included elements of the human immunodeficiency 'virus' (HIV) and a parasite that causes malaria. SARS-CoV-2 is alleged to trigger an alleged illness called Covid-19. I remembered Montagnier's name from my research years before into claims that an HIV 'retrovirus' causes AIDs – claims that were demolished by Berkeley virologist Peter Duesberg who showed that no one had ever proved that HIV causes acquired immunodeficiency syndrome or AIDS. Claims that become accepted as fact, publicly and medically, with no proof whatsoever are an ever-recurring story that profoundly applies to

'Covid'. Nevertheless, despite the lack of proof, Montagnier's team at the Pasteur Institute in Paris had a long dispute with American researcher Robert Gallo over which of them discovered and isolated the HIV 'virus' and with *no evidence* found it to cause AIDS. You will see later that there is also no evidence that any 'virus' causes any disease or that there is even such a thing as a 'virus' in the way it is said to exist. The claim to have 'isolated' the HIV 'virus' will be presented in its real context as we come to the shocking story – and it is a story – of SARS-CoV-2 and so will Montagnier's assertion that he identified the full SARS-CoV-2 genome.

Hoax in the making

We can pick up the 'Covid' story in 2010 and the publication by the Rockefeller Foundation of a document called 'Scenarios for the Future of Technology and International Development'. The inner circle of the Rockefeller family has been serving the Cult since John D. Rockefeller (1839-1937) made his fortune with Standard Oil. It is less well known that the same Rockefeller – the Bill Gates of his day – was responsible for establishing what is now referred to as 'Big Pharma', the global network of pharmaceutical companies that make outrageous profits dispensing scalpel and drug 'medicine' and are obsessed with pumping vaccines in ever-increasing number into as many human arms and backsides as possible. John D. Rockefeller was the driving force behind the creation of the 'education' system in the United States and elsewhere specifically designed to program the perceptions of generations thereafter. The Rockefeller family donated exceptionally valuable land in New York for the United Nations building and were central in establishing the World Health Organization in 1948 as an agency of the UN which was created from the start as a Trojan horse and stalking horse for world government. Now enter Bill Gates. His family and the Rockefellers have long been extremely close and I have seen genealogy which claims that if you go back far enough the two families fuse into the same bloodline. Gates has said that the Bill and Melinda Gates Foundation was inspired by the Rockefeller Foundation and why not

when both are serving the same Cult? Major tax-exempt foundations are overwhelmingly criminal enterprises in which Cult assets fund the Cult agenda in the guise of 'philanthropy' while avoiding tax in the process. Cult operatives can become mega-rich in their role of front men and women for the psychopaths at the inner core and they, too, have to be psychopaths to knowingly serve such evil. Part of the deal is that a big percentage of the wealth gleaned from representing the Cult has to be spent advancing the ambitions of the Cult and hence you have the Rockefeller Foundation, Bill and Melinda Gates Foundation (and *so* many more) and people like George Soros with his global Open Society Foundations spending their billions in pursuit of global Cult control. Gates is a global public face of the Cult with his interventions in world affairs including Big Tech influence; a central role in the 'Covid' and 'vaccine' scam; promotion of the climate change shakedown; manipulation of education; geoengineering of the skies; and his food-control agenda as the biggest owner of farmland in America, his GMO promotion and through other means. As one writer said: 'Gates monopolizes or wields disproportionate influence over the tech industry, global health and vaccines, agriculture and food policy (including biopiracy and fake food), weather modification and other climate technologies, surveillance, education and media.' The almost limitless wealth secured through Microsoft and other not-allowed-to-fail ventures (including vaccines) has been ploughed into a long, long list of Cult projects designed to enslave the entire human race. Gates and the Rockefellers have been working as one unit with the Rockefeller-established World Health Organization leading global 'Covid' policy controlled by Gates through his mouth-piece Tedros. Gates became the WHO's biggest funder when Trump announced that the American government would cease its donations, but Biden immediately said he would restore the money when he took office in January, 2021. The Gates Foundation (the Cult) owns through limitless funding the world health system and the major players across the globe in the 'Covid' hoax.

Okay, with that background we return to that Rockefeller Foundation document of 2010 headed 'Scenarios for the Future of Technology and International Development' and its 'imaginary' epidemic of a virulent and deadly influenza strain which infected 20 percent of the global population and killed eight million in seven months. The Rockefeller scenario was that the epidemic destroyed economies, closed shops, offices and other businesses and led to governments imposing fierce rules and restrictions that included mandatory wearing of face masks and body-temperature checks to enter communal spaces like railway stations and supermarkets. The document predicted that even after the height of the Rockefeller-envisaged epidemic the authoritarian rule would continue to deal with further pandemics, transnational terrorism, environmental crises and rising poverty. Now you may think that the Rockefellers are our modern-day seers or alternatively, and rather more likely, that they well knew what was planned a few years further on. Fascism had to be imposed, you see, to 'protect citizens from risk and exposure'. The Rockefeller scenario document said:

During the pandemic, national leaders around the world flexed their authority and imposed airtight rules and restrictions, from the mandatory wearing of face masks to body-temperature checks at the entries to communal spaces like train stations and supermarkets. Even after the pandemic faded, this more authoritarian control and oversight of citizens and their activities stuck and even intensified. In order to protect themselves from the spread of increasingly global problems – from pandemics and transnational terrorism to environmental crises and rising poverty – leaders around the world took a firmer grip on power.

At first, the notion of a more controlled world gained wide acceptance and approval. Citizens willingly gave up some of their sovereignty – and their privacy – to more paternalistic states in exchange for greater safety and stability. Citizens were more tolerant, and even eager, for top-down direction and oversight, and national leaders had more latitude to impose order in the ways they saw fit.

In developed countries, this heightened oversight took many forms: biometric IDs for all citizens, for example, and tighter regulation of key industries whose stability was deemed vital to national interests. In many developed countries, enforced cooperation with a suite of new regulations and agreements slowly but steadily restored both order and, importantly, economic growth.

There we have the prophetic Rockefellers in 2010 and three years later came their paper for the Global Health Summit in Beijing, China, when government representatives, the private sector, international organisations and groups met to discuss the next 100 years of 'global health'. The Rockefeller Foundation-funded paper was called 'Dreaming the Future of Health for the Next 100 Years and more prophecy ensued as it described a dystopian future: 'The abundance of data, digitally tracking and linking people may mean the 'death of privacy' and may replace physical interaction with transient, virtual connection, generating isolation and raising questions of how values are shaped in virtual networks.' Next in the 'Covid' hoax preparation sequence came a 'table top' simulation in 2018 for another 'imaginary' pandemic of a disease called Clade X which was said to kill 900 million people. The exercise was organised by the Gates-funded Johns Hopkins University's Center for Health Security in the United States and this is the very same university that has been compiling the disgustingly and systematically erroneous global figures for 'Covid' cases and deaths. Similar Johns Hopkins health crisis scenarios have included the Dark Winter exercise in 2001 and Atlantic Storm in 2005.

Nostradamus 201

For sheer predictive genius look no further prophecy-watchers than the Bill Gates-funded Event 201 held only six weeks before the 'coronavirus pandemic' is supposed to have broken out in China and Event 201 was based on a scenario of a global 'coronavirus pandemic'. Melinda Gates, the great man's missus, told the BBC that he had 'prepared for years' for a coronavirus pandemic which told us what we already knew. Nostradamugates had predicted in a TED talk in 2015 that a pandemic was coming that would kill a lot of people and demolish the world economy. My god, the man is a machine – possibly even literally. Now here he was only weeks before the real thing funding just such a simulated scenario and involving his friends and associates at Johns Hopkins, the World Economic Forum Cult-front of Klaus Schwab, the United Nations,

Johnson & Johnson, major banks, and officials from China and the Centers for Disease Control in the United States. What synchronicity – Johns Hopkins would go on to compile the fraudulent 'Covid' figures, the World Economic Forum and Schwab would push the 'Great Reset' in response to 'Covid', the Centers for Disease Control would be at the forefront of 'Covid' policy in the United States, Johnson & Johnson would produce a 'Covid vaccine', and everything would officially start just weeks later in China. Spooky, eh? They were even accurate in creating a simulation of a 'virus' pandemic because the 'real thing' would also be a simulation. Event 201 was not an exercise preparing for something that might happen; it was a rehearsal for what those in control knew was *going* to happen and very shortly. Hours of this simulation were posted on the Internet and the various themes and responses mirrored what would soon be imposed to transform human society. News stories were inserted and what they said would be commonplace a few weeks later with still more prophecy perfection. Much discussion focused on the need to deal with misinformation and the 'anti-vax movement' which is exactly what happened when the 'virus' arrived – was said to have arrived – in the West.

Cult-owned social media banned criticism and exposure of the official 'virus' narrative and when I said there *was* no 'virus' in early April, 2020, I was banned by one platform after another including YouTube, Facebook and later Twitter. The mainstream broadcast media in Britain was in effect banned from interviewing me by the Tony-Blair-created government broadcasting censor Ofcom headed by career government bureaucrat Melanie Dawes who was appointed just as the 'virus' hoax was about to play out in January, 2020. At the same time the Ickonic media platform was using Vimeo, another ultra-Zionist-owned operation, while our own player was being created and they deleted in an instant hundreds of videos, documentaries, series and shows to confirm their unbelievable vindictiveness. We had copies, of course, and they had to be restored one by one when our player was ready. These people have no class. Sabbatian Facebook promised free advertisements for the Gates-

controlled World Health Organization narrative while deleting 'false claims and conspiracy theories' to stop 'misinformation' about the alleged coronavirus. All these responses could be seen just a short while earlier in the scenarios of Event 201. Extreme censorship was absolutely crucial for the Cult because the official story was so ridiculous and unsupportable by the evidence that it could never survive open debate and the free-flow of information and opinion. If you can't win a debate then don't have one is the Cult's approach throughout history. Facebook's little boy front man – front boy – Mark Zuckerberg equated 'credible and accurate information' with official sources and exposing their lies with 'misinformation'.

Silencing those that can see

The censorship dynamic of Event 201 is now the norm with an army of narrative-supporting 'fact-checker' organisations whose entire reason for being is to tell the public that official narratives are true and those exposing them are lying. One of the most appalling of these 'fact-checkers' is called NewsGuard founded by ultra-Zionist Americans Gordon Crovitz and Steven Brill. Crovitz is a former publisher of *The Wall Street Journal*, former Executive Vice President of Dow Jones, a member of the Council on Foreign Relations (CFR), and on the board of the American Association of Rhodes Scholars. The CFR and Rhodes Scholarships, named after Rothschild agent Cecil Rhodes who plundered the gold and diamonds of South Africa for his masters and the Cult, have featured widely in my books. NewsGuard don't seem to like me for some reason – I really can't think why – and they have done all they can to have me censored and discredited which is, to quote an old British politician, like being savaged by a dead sheep. They are, however, like all in the censorship network, very well connected and funded by organisations themselves funded by, or connected to, Bill Gates. As you would expect with anything associated with Gates NewsGuard has an offshoot called HealthGuard which 'fights online health care hoaxes'. How very kind. Somehow the NewsGuard European Managing Director Anna-Sophie Harling, a remarkably young-

looking woman with no broadcasting experience and little hands-on work in journalism, has somehow secured a position on the 'Content Board' of UK government broadcast censor Ofcom. An executive of an organisation seeking to discredit dissidents of the official narratives is making decisions for the government broadcast 'regulator' about content?? Another appalling 'fact-checker' is Full Fact funded by George Soros and global censors Google and Facebook.

It's amazing how many activists in the 'fact-checking', 'anti-hate', arena turn up in government-related positions – people like UK Labour Party activist Imran Ahmed who heads the Center for Countering Digital Hate founded by people like Morgan McSweeney, now chief of staff to the Labour Party's hapless and useless 'leader' Keir Starmer. Digital Hate – which is what it really is – uses the American spelling of Center to betray its connection to a transatlantic network of similar organisations which in 2020 shapeshifted from attacking people for 'hate' to attacking them for questioning the 'Covid' hoax and the dangers of the 'Covid vaccine'. It's just a coincidence, you understand. This is one of Imran Ahmed's hysterical statements: 'I would go beyond calling anti-vaxxers conspiracy theorists to say they are an extremist group that pose a national security risk.' No one could ever accuse this prat of understatement and he's including in that those parents who are now against vaccines after their children were damaged for life or killed by them. He's such a nice man. Ahmed does the rounds of the Woke media getting soft-ball questions from spineless 'journalists' who never ask what right he has to campaign to destroy the freedom of speech of others while he demands it for himself. There also seems to be an overrepresentation in Ofcom of people connected to the narrative-worshipping BBC. This incredible global network of narrative-support was super-vital when the 'Covid' hoax was played in the light of the mega-whopper lies that have to be defended from the spotlight cast by the most basic intelligence.

Setting the scene

The Cult plays the long game and proceeds step-by-step ensuring that everything is in place before major cards are played and they don't come any bigger than the 'Covid' hoax. The psychopaths can't handle events where the outcome isn't certain and as little as possible – preferably nothing – is left to chance. Politicians, government and medical officials who would follow direction were brought to illusory power in advance by the Cult web whether on the national stage or others like state governors and mayors of America. For decades the dynamic between officialdom, law enforcement and the public was changed from one of service to one of control and dictatorship. Behaviour manipulation networks established within government were waiting to impose the coming 'Covid' rules and regulations specifically designed to subdue and rewire the psyche of the people in the guise of protecting health. These included in the UK the Behavioural Insights Team part-owned by the British government Cabinet Office; the Scientific Pandemic Insights Group on Behaviours (SPI-B); and a whole web of intelligence and military groups seeking to direct the conversation on social media and control the narrative. Among them are the cyberwarfare (on the people) 77th Brigade of the British military which is also coordinated through the Cabinet Office as civilian and military leadership continues to combine in what they call the Fusion Doctrine. The 77th Brigade is a British equivalent of the infamous Israeli (Sabbatian) military cyberwarfare and Internet manipulation operation Unit 8200 which I expose at length in *The Trigger*. Also carefully in place were the medical and science advisers to government – many on the payroll past or present of Bill Gates – and a whole alternative structure of unelected government stood by to take control when elected parliaments were effectively closed down once the 'Covid' card was slammed on the table. The structure I have described here and so much more was installed in every major country through the Cult networks. The top-down control hierarchy looks like this: The Cult – Cult-owned Gates – the World Health Organization and Tedros – Gates-funded or controlled chief medical officers and science 'advisers' (dictators) in each country –

political 'leaders'– law enforcement – The People. Through this simple global communication and enforcement structure the policy of the Cult could be imposed on virtually the entire human population so long as they acquiesced to the fascism. With everything in place it was time for the button to be pressed in late 2019/early 2020.

These were the prime goals the Cult had to secure for its will to prevail:

1) Locking down economies, closing all but designated 'essential' businesses (Cult-owned corporations were 'essential'), and putting the population under house arrest was an imperative to destroy independent income and employment and ensure dependency on the Cult-controlled state in the Hunger Games Society. Lockdowns had to be established as the global blueprint from the start to respond to the 'virus' and followed by pretty much the entire world.

2) The global population had to be terrified into believing in a deadly 'virus' that didn't actually exist so they would unquestioningly obey authority in the belief that authority must know how best to protect them and their families. Software salesman Gates would suddenly morph into the world's health expert and be promoted as such by the Cult-owned media.

3) A method of testing that wasn't testing for the 'virus', but was only claimed to be, had to be in place to provide the illusion of 'cases' and subsequent 'deaths' that had a very different cause to the 'Covid-19' that would be scribbled on the death certificate.

4) Because there was no 'virus' and the great majority testing positive with a test not testing for the 'virus' would have no symptoms of anything the lie had to be sold that people without symptoms (without the 'virus') could still pass it on to others. This was crucial to justify for the first time quarantining – house arresting – healthy people. Without this the economy-destroying lockdown of *everybody* could not have been credibly sold.

5) The 'saviour' had to be seen as a vaccine which beyond evil drug companies were working like angels of mercy to develop as quickly as possible, with all corners cut, to save the day. The public must absolutely not know that the 'vaccine' had nothing to do with a 'virus' or that the contents were ready and waiting with a very different motive long before the 'Covid' card was even lifted from the pack.

I said in March, 2020, that the 'vaccine' would have been created way ahead of the 'Covid' hoax which justified its use and the following December an article in the New York *Intelligencer* magazine said the Moderna 'vaccine' had been 'designed' by

January, 2020. This was 'before China had even acknowledged that the disease could be transmitted from human to human, more than a week before the first confirmed coronavirus case in the United States'. The article said that by the time the first American death was announced a month later 'the vaccine had already been manufactured and shipped to the National Institutes of Health for the beginning of its Phase I clinical trial'. The 'vaccine' was actually 'designed' long before that although even with this timescale you would expect the article to ask how on earth it could have been done that quickly. Instead it asked why the 'vaccine' had not been rolled out then and not months later. Journalism in the mainstream is truly dead. I am going to detail in the next chapter why the 'virus' has never existed and how a hoax on that scale was possible, but first the foundation on which the Big Lie of 'Covid' was built.

The test that doesn't test

Fraudulent 'testing' is the bottom line of the whole 'Covid' hoax and was the means by which a 'virus' that did not exist *appeared* to exist. They could only achieve this magic trick by using a test not testing for the 'virus'. To use a test that *was* testing for the 'virus' would mean that every test would come back negative given there was no 'virus'. They chose to exploit something called the RT-PCR test invented by American biochemist Kary Mullis in the 1980s who said publicly that his PCR test … *cannot detect infectious disease*. Yes, the 'test' used worldwide to detect infectious 'Covid' to produce all the illusory 'cases' and 'deaths' compiled by Johns Hopkins and others *cannot detect infectious disease*. This fact came from the mouth of the man who invented PCR and was awarded the Nobel Prize in Chemistry in 1993 for doing so. Sadly, and incredibly conveniently for the Cult, Mullis died in August, 2019, at the age of 74 just before his test would be fraudulently used to unleash fascism on the world. He was said to have died from pneumonia which was an irony in itself. A few months later he would have had 'Covid-19' on his death certificate. I say the timing of his death was convenient because had he lived Mullis, a brilliant, honest and decent man, would have been

vociferously speaking out against the use of his test to detect 'Covid' when it was never designed, or able, to do that. I know that to be true given that Mullis made the same point when his test was used to 'detect' – not detect – HIV. He had been seriously critical of the Gallo/Montagnier claim to have isolated the HIV 'virus' and shown it to cause AIDS for which Mullis said there was no evidence. AIDS is actually not a disease but a series of diseases from which people die all the time. When they die from those *same diseases* after a positive 'test' for HIV then AIDS goes on their death certificate. I think I've heard that before somewhere. Countries instigated a policy with 'Covid' that anyone who tested positive with a test not testing for the 'virus' and died of any other cause within 28 days and even longer 'Covid-19' had to go on the death certificate. Cases have come from the test that can't test for infectious disease and the deaths are those who have died of *anything* after testing positive with a test not testing for the 'virus'. I'll have much more later about the death certificate scandal.

Mullis was deeply dismissive of the now US 'Covid' star Anthony Fauci who he said was a liar who didn't know anything about anything – 'and I would say that to his face – nothing.' He said of Fauci: 'The man thinks he can take a blood sample, put it in an electron microscope and if it's got a virus in there you'll know it – he doesn't understand electron microscopy and he doesn't understand medicine and shouldn't be in a position like he's in.' That position, terrifyingly, has made him the decider of 'Covid' fascism policy on behalf of the Cult in his role as director since 1984 of the National Institute of Allergy and Infectious Diseases (NIAID) while his record of being wrong is laughable; but being wrong, so long as it's the *right kind* of wrong, is why the Cult loves him. He'll say anything the Cult tells him to say. Fauci was made Chief Medical Adviser to the President immediately Biden took office. Biden was installed in the White House by Cult manipulation and one of his first decisions was to elevate Fauci to a position of even more control. This is a coincidence? Yes, and I identify as a flamenco dancer called Lola. How does such an incompetent criminal like Fauci remain in that

pivotal position in American health since *the 1980s*? When you serve the Cult it looks after you until you are surplus to requirements. Kary Mullis said prophetically of Fauci and his like: 'Those guys have an agenda and it's not an agenda we would like them to have … they make their own rules, they change them when they want to, and Tony Fauci does not mind going on television in front of the people who pay his salary and lie directly into the camera.' Fauci has done that almost daily since the 'Covid' hoax began. Lying is in Fauci's DNA. To make the situation crystal clear about the PCR test this is a direct quote from its inventor Kary Mullis:

> It [the PCR test] doesn't tell you that you're sick and doesn't tell you that the thing you ended up with was really going to hurt you ...'

Ask yourself why governments and medical systems the world over have been using this very test to decide who is 'infected' with the SARS-CoV-2 'virus' and the alleged disease it allegedly causes, 'Covid-19'. The answer to that question will tell you what has been going on. By the way, here's a little show-stopper – the 'new' SARS-CoV-2 'virus' was 'identified' as such right from the start using … *the PCR test not testing for the 'virus'*. If you are new to this and find that shocking then stick around. I have hardly started yet. Even worse, other 'tests', like the 'Lateral Flow Device' (LFD), are considered so useless that they have to be *confirmed* by the PCR test! Leaked emails written by Ben Dyson, adviser to UK 'Health' Secretary Matt Hancock, said they were 'dangerously unreliable'. Dyson, executive director of strategy at the Department of Health, wrote: 'As of today, someone who gets a positive LFD result in (say) London has at best a 25 per cent chance of it being a true positive, but if it is a self-reported test potentially as low as 10 per cent (on an optimistic assumption about specificity) or as low as 2 per cent (on a more pessimistic assumption).' These are the 'tests' that schoolchildren and the public are being urged to have twice a week or more and have to isolate if they get a positive. Each fake positive goes in the statistics as a 'case' no matter how ludicrously inaccurate and the

'cases' drive lockdown, masks and the pressure to 'vaccinate'. The government said in response to the email leak that the 'tests' were accurate which confirmed yet again what shocking bloody liars they are. The real false positive rate is *100 percent* as we'll see. In another 'you couldn't make it up' the UK government agreed to pay £2.8 billion to California's Innova Medical Group to supply the irrelevant lateral flow tests. The company's primary test-making centre is in China. Innova Medical Group, established in March, 2020, is owned by Pasaca Capital Inc, chaired by Chinese-American millionaire Charles Huang who was born in Wuhan.

How it works – and how it doesn't

The RT-PCR test, known by its full title of Polymerase chain reaction, is used across the world to make millions, even billions, of copies of a DNA/RNA genetic information sample. The process is called 'amplification' and means that a tiny sample of genetic material is amplified to bring out the detailed content. I stress that it is not testing for an infectious disease. It is simply amplifying a sample of genetic material. In the words of Kary Mullis: 'PCR is … just a process that's used to make a whole lot of something out of something.' To emphasise the point companies that make the PCR tests circulated around the world to 'test' for 'Covid' warn on the box that it can't be used to detect 'Covid' or infectious disease and is for research purposes only. It's okay, rest for a minute and you'll be fine. This is the test that produces the 'cases' and 'deaths' that have been used to destroy human society. All those global and national medical and scientific 'experts' demanding this destruction to 'save us' *KNOW* that the test is not testing for the 'virus' and the cases and deaths they claim to be real are an almost unimaginable fraud. Every one of them and so many others including politicians and psychopaths like Gates and Tedros must be brought before Nuremburg-type trials and jailed for the rest of their lives. The more the genetic sample is amplified by PCR the more elements of that material become sensitive to the test and by that I don't mean sensitive for a 'virus' but for elements of the genetic material which

is *naturally* in the body or relates to remnants of old conditions of various kinds lying dormant and causing no disease. Once the amplification of the PCR reaches a certain level *everyone* will test positive. So much of the material has been made sensitive to the test that everyone will have some part of it in their body. Even lying criminals like Fauci have said that once PCR amplifications pass 35 cycles everything will be a false positive that cannot be trusted for the reasons I have described. I say, like many proper doctors and scientists, that 100 percent of the 'positives' are false, but let's just go with Fauci for a moment.

He says that any amplification over 35 cycles will produce false positives and yet the US Centers for Disease Control (CDC) and Food and Drug Administration (FDA) have recommended up to *40 cycles* and the National Health Service (NHS) in Britain admitted in an internal document for staff that it was using *45 cycles* of amplification. A long list of other countries has been doing the same and at least one 'testing' laboratory has been using *50 cycles*. Have you ever heard a doctor, medical 'expert' or the media ask what level of amplification has been used to claim a 'positive'. The 'test' comes back 'positive' and so you have the 'virus', end of story. Now we can see how the government in Tanzania could send off samples from a goat and a pawpaw fruit under human names and both came back positive for 'Covid-19'. Tanzania president John Magufuli mocked the 'Covid' hysteria, the PCR test and masks and refused to import the DNA-manipulating 'vaccine'. The Cult hated him and an article sponsored by the Bill Gates Foundation appeared in the London *Guardian* in February, 2021, headed 'It's time for Africa to rein in Tanzania's anti-vaxxer president'. Well, 'reined in' he shortly was. Magufuli appeared in good health, but then, in March, 2021, he was dead at 61 from 'heart failure'. He was replaced by Samia Hassan Suhulu who is connected to Klaus Schwab's World Economic Forum and she immediately reversed Magufuli's 'Covid' policy. A sample of cola tested positive for 'Covid' with the PCR test in Germany while American actress and singer-songwriter Erykah Badu tested positive in one nostril and negative in the other. Footballer Ronaldo called

the PCR test 'bullshit' after testing positive three times and being forced to quarantine and miss matches when there was nothing wrong with him. The mantra from Tedros at the World Health Organization and national governments (same thing) has been test, test, test. They know that the more tests they can generate the more fake 'cases' they have which go on to become 'deaths' in ways I am coming to. The UK government has its Operation Moonshot planned to test multiple millions every day in workplaces and schools with free tests for everyone to use twice a week at home in line with the Cult plan from the start to make testing part of life. A government advertisement for an 'Interim Head of Asymptomatic Testing Communication' said the job included responsibility for delivering a 'communications strategy' (propaganda) 'to support the expansion of asymptomatic testing that *'normalises testing as part of everyday life'*. More tests means more fake 'cases', 'deaths' and fascism. I have heard of, and from, many people who booked a test, couldn't turn up, and yet got a positive result through the post for a test they'd never even had. The whole thing is crazy, but for the Cult there's method in the madness. Controlling and manipulating the level of amplification of the test means the authorities can control whenever they want the number of apparent 'cases' and 'deaths'. If they want to justify more fascist lockdown and destruction of livelihoods they keep the amplification high. If they want to give the illusion that lockdowns and the 'vaccine' are working then they lower the amplification and 'cases' and 'deaths' will appear to fall. In January, 2021, the Cult-owned World Health Organization suddenly warned laboratories about over-amplification of the test and to lower the threshold. Suddenly headlines began appearing such as: 'Why ARE "Covid" cases plummeting?' This was just when the vaccine rollout was underway and I had predicted months before they would make cases appear to fall through amplification tampering when the 'vaccine' came. These people are so predictable.

Cow vaccines?

The question must be asked of what is on the test swabs being poked far up the nose of the population to the base of the brain? A nasal swab punctured one woman's brain and caused it to leak fluid. Most of these procedures are being done by people with little training or medical knowledge. Dr Lorraine Day, former orthopaedic trauma surgeon and Chief of Orthopaedic Surgery at San Francisco General Hospital, says the tests are really a *'vaccine'*. Cows have long been vaccinated this way. She points out that masks have to cover the nose and the mouth where it is claimed the 'virus' exists in saliva. Why then don't they take saliva from the mouth as they do with a DNA test instead of pushing a long swab up the nose towards the brain? The ethmoid bone separates the nasal cavity from the brain and within that bone is the cribriform plate. Dr Day says that when the swab is pushed up against this plate and twisted the procedure is 'depositing things back there'. She claims that among these 'things' are nanoparticles that can enter the brain. Researchers have noted that a team at the Gates-funded Johns Hopkins have designed tiny, star-shaped micro-devices that can latch onto intestinal mucosa and release drugs into the body. Mucosa is the thin skin that covers the inside surface of parts of the body such as *the nose* and mouth and produces mucus to protect them. The Johns Hopkins micro-devices are called 'theragrippers' and were 'inspired' by a parasitic worm that digs its sharp teeth into a host's intestines. Nasal swabs are also coated in the sterilisation agent ethylene oxide. The US National Cancer Institute posts this explanation on its website:

> At room temperature, ethylene oxide is a flammable colorless gas with a sweet odor. It is used primarily to produce other chemicals, including antifreeze. In smaller amounts, ethylene oxide is used as a pesticide and a sterilizing agent. The ability of ethylene oxide to damage DNA makes it an effective sterilizing agent but also accounts for its cancer-causing activity.

The Institute mentions lymphoma and leukaemia as cancers most frequently reported to be associated with occupational exposure to ethylene oxide along with stomach and breast cancers. How does anyone think this is going to work out with the constant testing

regime being inflicted on adults and children at home and at school that will accumulate in the body anything that's on the swab?

Doctors know best

It is vital for people to realise that 'hero' doctors 'know' only what the Big Pharma-dominated medical authorities tell them to 'know' and if they refuse to 'know' what they are told to 'know' they are out the door. They are mostly not physicians or healers, but repeaters of the official narrative – or else. I have seen alleged professional doctors on British television make shocking statements that we are supposed to take seriously. One called 'Dr' Amir Khan, who is actually telling patients how to respond to illness, said that men could take the birth pill to 'help slow down the effects of Covid-19'. In March, 2021, another ridiculous 'Covid study' by an American doctor proposed injecting men with the female sex hormone progesterone as a 'Covid' treatment. British doctor Nighat Arif told the BBC that face coverings were now going to be part of ongoing normal. Yes, the vaccine protects you, she said (evidence?) … but the way to deal with viruses in the community was always going to come down to hand washing, face covering and keeping a physical distance. That's not what we were told before the 'vaccine' was circulating. Arif said she couldn't imagine ever again going on the underground or in a lift without a mask. I was just thanking my good luck that she was not my doctor when she said – in March, 2021 – that if 'we are *behaving* and we are doing all the right things' she thought we could 'have our nearest and dearest around us at home … around *Christmas* and N*ew Year!* Her patronising delivery was the usual school teacher talking to six-year-olds as she repeated every government talking point and probably believed them all. If we have learned anything from the 'Covid' experience surely it must be that humanity's perception of doctors needs a fundamental rethink. NHS 'doctor' Sara Kayat told her television audience that the 'Covid vaccine' would '100 percent prevent hospitalisation and death'. Not even Big Pharma claimed that. We have to stop taking 'experts' at their word without question when so many of them are

clueless and only repeating the party line on which their careers depend. That is not to say there are not brilliants doctors – there are and I have spoken to many of them since all this began – but you won't see them in the mainstream media or quoted by the psychopaths and yes-people in government.

Remember the name – Christian Drosten

German virologist Christian Drosten, Director of Charité Institute of Virology in Berlin, became a national star after the pandemic hoax began. He was feted on television and advised the German government on 'Covid' policy. Most importantly to the wider world Drosten led a group that produced the 'Covid' testing protocol for the PCR test. What a remarkable feat given the PCR cannot test for infectious disease and even more so when you think that Drosten said that his method of testing for SARS-CoV-2 was developed 'without having virus material available'. *He developed a test for a 'virus' that he didn't have and had never seen*. Let that sink in as you survey the global devastation that came from what he did. The whole catastrophe of Drosten's 'test' was based on the alleged genetic sequence published by Chinese scientists on the Internet. We will see in the next chapter that this alleged 'genetic sequence' has never been produced by China or anyone and cannot be when there *is no* SARS-CoV-2. Drosten, however, doesn't seem to let little details like that get in the way. He was the lead author with Victor Corman from the same Charité Hospital of the paper 'Detection of 2019 novel coronavirus (2019-nCoV) by real-time PCR' published in a magazine called *Eurosurveillance*. This became known as the Corman-Drosten paper. In November, 2020, with human society devastated by the effects of the Corman-Drosten test baloney, the protocol was publicly challenged by 22 international scientists and independent researchers from Europe, the United States, and Japan. Among them were senior molecular geneticists, biochemists, immunologists, and microbiologists. They produced a document headed 'External peer review of the RTPCR test to detect SARS-Cov-2 Reveals 10 Major Flaws At The Molecular and Methodological Level: Consequences

For False-Positive Results'. The flaws in the Corman-Drosten test included the following:

- The test is non-specific because of erroneous design
- Results are enormously variable
- The test is unable to discriminate between the whole 'virus' and viral fragments
- It doesn't have positive or negative controls
- The test lacks a standard operating procedure
- It is unsupported by proper peer view

The scientists said the PCR 'Covid' testing protocol was not founded on science and they demanded the Corman-Drosten paper be retracted by *Eurosurveillance*. They said all present and previous Covid deaths, cases, and 'infection rates' should be subject to a massive retroactive inquiry. Lockdowns and travel restrictions should be reviewed and relaxed and those diagnosed through PCR to have 'Covid-19' should not be forced to isolate. Dr Kevin Corbett, a health researcher and nurse educator with a long academic career producing a stream of peer-reviewed publications at many UK universities, made the same point about the PCR test debacle. He said of the scientists' conclusions: 'Every scientific rationale for the development of that test has been totally destroyed by this paper. It's like Hiroshima/Nagasaki to the Covid test.' He said that China hadn't given them an isolated 'virus' when Drosten developed the test. Instead they had developed the test from *a sequence in a gene bank*.' Put another way … *they made it up*! The scientists were supported in this contention by a Portuguese appeals court which ruled in November, 2020, that PCR tests are unreliable and it is unlawful to quarantine people based solely on a PCR test. The point about China not providing an isolated virus must be true when the 'virus' has never been isolated to this day and the consequences of that will become clear. Drosten and company produced this useless 'protocol' right on cue in January, 2020, just as the 'virus' was said to

be moving westward and it somehow managed to successfully pass a peer-review in 24 hours. In other words there was no peer-review for a test that would be used to decide who had 'Covid' and who didn't across the world. The Cult-created, Gates-controlled World Health Organization immediately recommended all its nearly 200 member countries to use the Drosten PCR protocol to detect 'cases' and 'deaths'. The sting was underway and it continues to this day.

So who is this Christian Drosten that produced the means through which death, destruction and economic catastrophe would be justified? His education background, including his doctoral thesis, would appear to be somewhat shrouded in mystery and his track record is dire as with another essential player in the 'Covid' hoax, the Gates-funded Professor Neil Ferguson at the Gates-funded Imperial College in London of whom more shortly. Drosten predicted in 2003 that the alleged original SARS 'virus' (SARS-1') was an epidemic that could have serious effects on economies and an effective vaccine would take at least two years to produce. Drosten's answer to every alleged 'outbreak' is a vaccine which you won't be shocked to know. What followed were just 774 official deaths worldwide and none in Germany where there were only nine cases. That is even if you believe there ever was a SARS 'virus' when the evidence is zilch and I will expand on this in the next chapter. Drosten claims to be co-discoverer of 'SARS-1' and developed a test for it in 2003. He was screaming warnings about 'swine flu' in 2009 and how it was a widespread infection far more severe than any dangers from a vaccine could be and people should get vaccinated. It would be helpful for Drosten's vocal chords if he simply recorded the words 'the virus is deadly and you need to get vaccinated' and copies could be handed out whenever the latest made-up threat comes along. Drosten's swine flu epidemic never happened, but Big Pharma didn't mind with governments spending hundreds of millions on vaccines that hardly anyone bothered to use and many who did wished they hadn't. A study in 2010 revealed that the risk of dying from swine flu, or H1N1, was no higher than that of the annual seasonal flu which is what at least most of 'it' really was as in

the case of 'Covid-19'. A media investigation into Drosten asked how with such a record of inaccuracy he could be *the* government adviser on these issues. The answer to that question is the same with Drosten, Ferguson and Fauci – they keep on giving the authorities the 'conclusions' and 'advice' they want to hear. Drosten certainly produced the goods for them in January, 2020, with his PCR protocol garbage and provided the foundation of what German internal medicine specialist Dr Claus Köhnlein, co-author of *Virus Mania*, called the 'test pandemic'. The 22 scientists in the *Eurosurveillance* challenge called out conflicts of interest within the Drosten 'protocol' group and with good reason. Olfert Landt, a regular co-author of Drosten 'studies', owns the biotech company TIB Molbiol Syntheselabor GmbH in Berlin which manufactures and sells the tests that Drosten and his mates come up with. They have done this with SARS, Enterotoxigenic E. coli (ETEC), MERS, Zika 'virus', yellow fever, and now 'Covid'. Landt told the *Berliner Zeitung* newspaper:

> The testing, design and development came from the Charité [Drosten and Corman]. We simply implemented it immediately in the form of a kit. And if we don't have the virus, which originally only existed in Wuhan, we can make a synthetic gene to simulate the genome of the virus. That's what we did very quickly.

This is more confirmation that the Drosten test was designed without access to the 'virus' and only a synthetic simulation which is what SARS-CoV-2 really is – a computer-generated synthetic fiction. It's quite an enterprise they have going here. A Drosten team decides what the test for something should be and Landt's biotech company flogs it to governments and medical systems across the world. His company must have made an absolute fortune since the 'Covid' hoax began. Dr Reiner Fuellmich, a prominent German consumer protection trial lawyer in Germany and California, is on Drosten's case and that of Tedros at the World Health Organization for crimes against humanity with a class-action lawsuit being prepared in the United States and other legal action in Germany.

Why China?

Scamming the world with a 'virus' that doesn't exist would seem impossible on the face of it, but not if you have control of the relatively few people that make policy decisions and the great majority of the global media. Remember it's not about changing 'real' reality it's about controlling *perception* of reality. You don't have to make something happen you only have make people *believe* that it's happening. Renegade Minds understand this and are therefore much harder to swindle. 'Covid-19' is not a 'real' 'virus'. It's a mind virus, like a computer virus, which has infected the minds, not the bodies, of billions. It all started, publically at least, in China and that alone is of central significance. The Cult was behind the revolution led by its asset Mao Zedong, or Chairman Mao, which established the People's Republic of China on October 1st, 1949. It should have been called The Cult's Republic of China, but the name had to reflect the recurring illusion that vicious dictatorships are run by and for the people (see all the 'Democratic Republics' controlled by tyrants). In the same way we have the 'Biden' Democratic Republic of America officially ruled by a puppet tyrant (at least temporarily) on behalf of Cult tyrants. The creation of Mao's merciless communist/fascist dictatorship was part of a frenzy of activity by the Cult at the conclusion of World War Two which, like the First World War, it had instigated through its assets in Germany, Britain, France, the United States and elsewhere. Israel was formed in 1948; the Soviet Union expanded its 'Iron Curtain' control, influence and military power with the Warsaw Pact communist alliance in 1955; the United Nations was formed in 1945 as a Cult precursor to world government; and a long list of world bodies would be established including the World Health Organization (1948), World Trade Organization (1948 under another name until 1995), International Monetary Fund (1945) and World Bank (1944). Human society was redrawn and hugely centralised in the global Problem-Reaction-Solution that was World War Two. All these changes were significant. Israel would become the headquarters of the Sabbatians

and the revolution in China would prepare the ground and control system for the events of 2019/2020.

Renegade Minds know there are no borders except for public consumption. The Cult is a seamless, borderless global entity and to understand the game we need to put aside labels like borders, nations, countries, communism, fascism and democracy. These delude the population into believing that countries are ruled within their borders by a government of whatever shade when these are mere agencies of a global power. America's illusion of democracy and China's communism/fascism are subsidiaries – vehicles – for the same agenda. We may hear about conflict and competition between America and China and on the lower levels that will be true; but at the Cult level they are branches of the same company in the way of the McDonald's example I gave earlier. I have tracked in the books over the years support by US governments of both parties for Chinese Communist Party infiltration of American society through allowing the sale of land, even military facilities, and the acquisition of American business and university influence. All this is underpinned by the infamous stealing of intellectual property and technological know-how. Cult-owned Silicon Valley corporations waive their fraudulent 'morality' to do business with human-rights-free China; Cult-controlled Disney has become China's PR department; and China in effect owns 'American' sports such as basketball which depends for much of its income on Chinese audiences. As a result any sports player, coach or official speaking out against China's horrific human rights record is immediately condemned or fired by the China-worshipping National Basketball Association. One of the first acts of China-controlled Biden was to issue an executive order telling federal agencies to stop making references to the 'virus' by the 'geographic location of its origin'. Long-time Congressman Jerry Nadler warned that criticising China, America's biggest rival, leads to hate crimes against Asian people in the United States. So shut up you bigot. China is fast closing in on Israel as a country that must not be criticised which is apt, really, given that Sabbatians control them both. The two countries have

developed close economic, military, technological and strategic ties which include involvement in China's 'Silk Road' transport and economic initiative to connect China with Europe. Israel was the first country in the Middle East to recognise the establishment of Mao's tyranny in 1950 months after it was established.

Project Wuhan – the 'Covid' Psyop

I emphasise again that the Cult plays the long game and what is happening to the world today is the result of centuries of calculated manipulation following a script to take control step-by-step of every aspect of human society. I will discuss later the common force behind all this that has spanned those centuries and thousands of years if the truth be told. Instigating the Mao revolution in China in 1949 with a 2020 'pandemic' in mind is not only how they work – the 71 years between them is really quite short by the Cult's standards of manipulation preparation. The reason for the Cult's Chinese revolution was to create a fiercely-controlled environment within which an extreme structure for human control could be incubated to eventually be unleashed across the world. We have seen this happen since the 'pandemic' emerged from China with the Chinese control-structure founded on AI technology and tyrannical enforcement sweep across the West. Until the moment when the Cult went for broke in the West and put its fascism on public display Western governments had to pay some lip-service to freedom and democracy to not alert too many people to the tyranny-in-the-making. Freedoms were more subtly eroded and power centralised with covert government structures put in place waiting for the arrival of 2020 when that smokescreen of 'freedom' could be dispensed with. The West was not able to move towards tyranny before 2020 anything like as fast as China which was created as a tyranny and had no limits on how fast it could construct the Cult's blueprint for global control. When the time came to impose that structure on the world it was the same Cult-owned Chinese communist/fascist government that provided the excuse – the 'Covid pandemic'. It was absolutely crucial to the Cult plan for the Chinese response to the 'pandemic' –

draconian lockdowns of the entire population – to become the blueprint that Western countries would follow to destroy the livelihoods and freedom of their people. This is why the Cult-owned, Gates-owned, WHO Director-General Tedros said early on:

> The Chinese government is to be congratulated for the extraordinary measures it has taken to contain the outbreak. China is actually setting a new standard for outbreak response and it is not an exaggeration.

Forbes magazine said of China: '… those measures protected untold millions from getting the disease'. The Rockefeller Foundation 'epidemic scenario' document in 2010 said 'prophetically':

> However, a few countries did fare better – China in particular. The Chinese government's quick imposition and enforcement of mandatory quarantine for all citizens, as well as its instant and near-hermetic sealing off of all borders, saved millions of lives, stopping the spread of the virus far earlier than in other countries and enabling a swifter post-pandemic recovery.

Once again – *spooky*.

The first official story was the 'bat theory' or rather the bat diversion. The source of the 'virus outbreak' we were told was a ''wet market' in Wuhan where bats and other animals are bought and eaten in horrifically unhygienic conditions. Then another story emerged through the alternative media that the 'virus' had been released on purpose or by accident from a BSL-4 (biosafety level 4) laboratory in Wuhan not far from the wet market. The lab was reported to create and work with lethal concoctions and bioweapons. Biosafety level 4 is the highest in the World Health Organization system of safety and containment. Renegade Minds are aware of what I call designer manipulation. The ideal for the Cult is for people to buy its prime narrative which in the opening salvoes of the 'pandemic' was the wet market story. It knows, however, that there is now a considerable worldwide alternative media of researchers sceptical of anything governments say and they are often given a version of events in a form they can perceive as credible while misdirecting them from the real truth. In this case let them

think that the conspiracy involved is a 'bioweapon virus' released from the Wuhan lab to keep them from the real conspiracy – *there is no 'virus'*. The WHO's current position on the source of the outbreak at the time of writing appears to be: 'We haven't got a clue, mate.' This is a good position to maintain mystery and bewilderment. The inner circle will know where the 'virus' came from – *nowhere*. The bottom line was to ensure the public believed there *was* a 'virus' and it didn't much matter if they thought it was natural or had been released from a lab. The belief that there was a 'deadly virus' was all that was needed to trigger global panic and fear. The population was terrified into handing their power to authority and doing what they were told. They had to or they were 'all gonna die'.

In March, 2020, information began to come my way from real doctors and scientists and my own additional research which had my intuition screaming: 'Yes, that's it! *There is no virus*.' The 'bioweapon' was not the 'virus'; it was the *'vaccine'* already being talked about that would be the bioweapon. My conclusion was further enhanced by happenings in Wuhan. The 'virus' was said to be sweeping the city and news footage circulated of people collapsing in the street (which they've never done in the West with the same 'virus'). The Chinese government was building 'new hospitals' in a matter of ten days to 'cope with demand' such was the virulent nature of the 'virus'. Yet in what seemed like no time the 'new hospitals' closed – even if they even opened – and China declared itself 'virus-free'. It was back to business as usual. This was more propaganda to promote the Chinese draconian lockdowns in the West as the way to 'beat the virus'. Trouble was that we subsequently had lockdown after lockdown, but never business as usual. As the people of the West and most of the rest of the world were caught in an ever-worsening spiral of lockdown, social distancing, masks, isolated old people, families forced apart, and livelihood destruction, it was party-time in Wuhan. Pictures emerged of thousands of people enjoying pool parties and concerts. It made no sense until you realised there never was a 'virus' and the

whole thing was a Cult set-up to transform human society out of one its major global strongholds – China.

How is it possible to deceive virtually the entire world population into believing there is a deadly virus when there is not even a 'virus' let alone a deadly one? It's nothing like as difficult as you would think and that's clearly true because it happened.

Postscript: See end of book Postscript for more on the 'Wuhan lab virus release' story which the authorities and media were pushing heavily in the summer of 2021 to divert attention from the truth that the 'Covid virus' is pure invention.

CHAPTER FIVE

There *is no* 'virus'

You can fool some of the people all of the time, and all of the people some of the time, but you cannot fool all of the people all of the time
Abraham Lincoln

The greatest form of mind control is repetition. The more you repeat the same mantra of alleged 'facts' the more will accept them to be true. It becomes an 'everyone knows that, mate'. If you can also censor any other version or alternative to your alleged 'facts' you are pretty much home and cooking.

By the start of 2020 the Cult owned the global mainstream media almost in its entirety to spew out its 'Covid' propaganda and ignore or discredit any other information and view. Cult-owned social media platforms in Cult-owned Silicon Valley were poised and ready to unleash a campaign of ferocious censorship to obliterate all but the official narrative. To complete the circle many demands for censorship by Silicon Valley were led by the mainstream media as 'journalists' became full-out enforcers for the Cult both as propagandists and censors. Part of this has been the influx of young people straight out of university who have become 'journalists' in significant positions. They have no experience and a headful of programmed perceptions from their years at school and university at a time when today's young are the most perceptually-targeted generations in known human history given the insidious impact of technology. They enter the media perceptually prepared and ready to repeat the narratives of the system that programmed them to

repeat its narratives. The BBC has a truly pathetic 'specialist disinformation reporter' called Marianna Spring who fits this bill perfectly. She is clueless about the world, how it works and what is really going on. Her role is to discredit anyone doing the job that a proper journalist would do and system-serving hacks like Spring wouldn't dare to do or even see the need to do. They are too busy licking the arse of authority which can never be wrong and, in the case of the BBC propaganda programme, *Panorama*, contacting payments systems such as PayPal to have a donations page taken down for a film company making documentaries questioning vaccines. Even the BBC soap opera *EastEnders* included a disgracefully biased scene in which an inarticulate white working class woman was made to look foolish for questioning the 'vaccine' while a well-spoken black man and Asian woman promoted the government narrative. It ticked every BBC box and the fact that the black and minority community was resisting the 'vaccine' had nothing to do with the way the scene was written. The BBC has become a disgusting tyrannical propaganda and censorship operation that should be defunded and disbanded and a free media take its place with a brief to stop censorship instead of demanding it. A BBC 'interview' with Gates goes something like: 'Mr Gates, sir, if I can call you sir, would you like to tell our audience why you are such a great man, a wonderful humanitarian philanthropist, and why you should absolutely be allowed as a software salesman to decide health policy for approaching eight billion people? Thank you, sir, please sir.' Propaganda programming has been incessant and merciless and when all you hear is the same story from the media, repeated by those around you who have only heard the same story, is it any wonder that people on a grand scale believe absolute mendacious garbage to be true? You are about to see, too, why this level of information control is necessary when the official 'Covid' narrative is so nonsensical and unsupportable by the evidence.

Structure of Deceit

The pyramid structure through which the 'Covid' hoax has been manifested is very simple and has to be to work. As few people as possible have to be involved with full knowledge of what they are doing – and why – or the real story would get out. At the top of the pyramid are the inner core of the Cult which controls Bill Gates who, in turn, controls the World Health Organization through his pivotal funding and his puppet Director-General mouthpiece, Tedros. Before he was appointed Tedros was chair of the Gates-founded Global Fund to 'fight against AIDS, tuberculosis and malaria', a board member of the Gates-funded 'vaccine alliance' GAVI, and on the board of another Gates-funded organisation. Gates owns him and picked him for a specific reason – Tedros is a crook and worse. 'Dr' Tedros (he's not a medical doctor, the first WHO chief not to be) was a member of the tyrannical Marxist government of Ethiopia for decades with all its human rights abuses. He has faced allegations of corruption and misappropriation of funds and was exposed three times for covering up cholera epidemics while Ethiopia's health minister. Tedros appointed the mass-murdering genocidal Zimbabwe dictator Robert Mugabe as a WHO goodwill ambassador for public health which, as with Tedros, is like appointing a psychopath to run a peace and love campaign. The move was so ridiculous that he had to drop Mugabe in the face of widespread condemnation. American economist David Steinman, a Nobel peace prize nominee, lodged a complaint with the International Criminal Court in The Hague over alleged genocide by Tedros when he was Ethiopia's foreign minister. Steinman says Tedros was a 'crucial decision maker' who directed the actions of Ethiopia's security forces from 2013 to 2015 and one of three officials in charge when those security services embarked on the 'killing' and 'torturing' of Ethiopians. You can see where Tedros is coming from and it's sobering to think that he has been the vehicle for Gates and the Cult to direct the global response to 'Covid'. Think about that. A psychopathic Cult dictates to psychopath Gates who dictates to psychopath Tedros who dictates how countries of the world must respond to a 'Covid virus' never scientifically shown to exist. At the same time psychopathic Cult-owned Silicon Valley information

giants like Google, YouTube, Facebook and Twitter announced very early on that they would give the Cult/Gates/Tedros/WHO version of the narrative free advertising and censor those who challenged their intelligence-insulting, mendacious story.

The next layer in the global 'medical' structure below the Cult, Gates and Tedros are the chief medical officers and science 'advisers' in each of the WHO member countries which means virtually all of them. Medical officers and arbiters of science (they're not) then take the WHO policy and recommended responses and impose them on their country's population while the political 'leaders' say they are deciding policy (they're clearly not) by 'following the science' on the advice of the 'experts' – the same medical officers and science 'advisers' (dictators). In this way with the rarest of exceptions the entire world followed the same policy of lockdown, people distancing, masks and 'vaccines' dictated by the psychopathic Cult, psychopathic Gates and psychopathic Tedros who we are supposed to believe give a damn about the health of the world population they are seeking to enslave. That, amazingly, is all there is to it in terms of crucial decision-making. Medical staff in each country then follow like sheep the dictates of the shepherds at the top of the national medical hierarchies – chief medical officers and science 'advisers' who themselves follow like sheep the shepherds of the World Health Organization and the Cult. Shepherds at the national level often have major funding and other connections to Gates and his Bill and Melinda Gates Foundation which carefully hands out money like confetti at a wedding to control the entire global medical system from the WHO down.

Follow the money

Christopher Whitty, Chief Medical Adviser to the UK Government at the centre of 'virus' policy, a senior adviser to the government's Scientific Advisory Group for Emergencies (SAGE), and Executive Board member of the World Health Organization, was gifted a grant of $40 million by the Bill and Melinda Gates Foundation for malaria research in Africa. The BBC described the unelected Whitty as 'the

official who will probably have the greatest impact on our everyday lives of any individual policymaker in modern times' and so it turned out. What Gates and Tedros have said Whitty has done like his equivalents around the world. Patrick Vallance, co-chair of SAGE and the government's Chief Scientific Adviser, is a former executive of Big Pharma giant GlaxoSmithKline with its fundamental financial and business connections to Bill Gates. In September, 2020, it was revealed that Vallance owned a deferred bonus of shares in GlaxoSmithKline worth £600,000 while the company was 'developing' a 'Covid vaccine'. Move along now – nothing to see here – what could possibly be wrong with that? Imperial College in London, a major player in 'Covid' policy in Britain and elsewhere with its 'Covid-19' Response Team, is funded by Gates and has big connections to China while the now infamous Professor Neil Ferguson, the useless 'computer modeller' at Imperial College is also funded by Gates. Ferguson delivered the dramatically inaccurate excuse for the first lockdowns (much more in the next chapter). The Institute for Health Metrics and Evaluation (IHME) in the United States, another source of outrageously false 'Covid' computer models to justify lockdowns, is bankrolled by Gates who is a vehement promotor of lockdowns. America's version of Whitty and Vallance, the again now infamous Anthony Fauci, has connections to 'Covid vaccine' maker Moderna as does Bill Gates through funding from the Bill and Melinda Gates Foundation. Fauci is director of the National Institute of Allergy and Infectious Diseases (NIAID), a major recipient of Gates money, and they are very close. Deborah Birx who was appointed White House Coronavirus Response Coordinator in February, 2020, is yet another with ties to Gates. Everywhere you look at the different elements around the world behind the coordination and decision making of the 'Covid' hoax there is Bill Gates and his money. They include the World Health Organization; Centers for Disease Control (CDC) in the United States; National Institutes of Health (NIH) of Anthony Fauci; Imperial College and Neil Ferguson; the London School of Hygiene where Chris Whitty worked; Regulatory agencies like the UK Medicines & Healthcare products Regulatory Agency (MHRA)

which gave emergency approval for 'Covid vaccines'; Wellcome Trust; GAVI, the Vaccine Alliance; the Coalition for Epidemic Preparedness Innovations (CEPI); Johns Hopkins University which has compiled the false 'Covid' figures; and the World Economic Forum. A Nationalfile.com article said:

> Gates has a lot of pull in the medical world, he has a multi-million dollar relationship with Dr. Fauci, and Fauci originally took the Gates line supporting vaccines and casting doubt on [the drug hydroxychloroquine]. Coronavirus response team member Dr. Deborah Birx, appointed by former president Obama to serve as United States Global AIDS Coordinator, also sits on the board of a group that has received billions from Gates' foundation, and Birx reportedly used a disputed Bill Gates-funded model for the White House's Coronavirus effort. Gates is a big proponent for a population lockdown scenario for the Coronavirus outbreak.

Another funder of Moderna is the Defense Advanced Research Projects Agency (DARPA), the technology-development arm of the Pentagon and one of the most sinister organisations on earth. DARPA had a major role with the CIA covert technology-funding operation In-Q-Tel in the development of Google and social media which is now at the centre of global censorship. Fauci and Gates are extremely close and openly admit to talking regularly about 'Covid' policy, but then why wouldn't Gates have a seat at every national 'Covid' table after his Foundation committed $1.75 billion to the 'fight against Covid-19'. When passed through our Orwellian Translation Unit this means that he has bought and paid for the Cult-driven 'Covid' response worldwide. Research the major 'Covid' response personnel in your own country and you will find the same Gates funding and other connections again and again. Medical and science chiefs following World Health Organization 'policy' sit atop a medical hierarchy in their country of administrators, doctors and nursing staff. These 'subordinates' are told they must work and behave in accordance with the policy delivered from the 'top' of the national 'health' pyramid which is largely the policy delivered by the WHO which is the policy delivered by Gates and the Cult. The whole 'Covid' narrative has been imposed on medical staff by a climate of fear although great numbers don't even need that to comply. They do so through breathtaking levels of ignorance and

include doctors who go through life simply repeating what Big Pharma and their hierarchical masters tell them to say and believe. No wonder Big Pharma 'medicine' is one of the biggest killers on Planet Earth.

The same top-down system of intimidation operates with regard to the Cult Big Pharma cartel which also dictates policy through national and global medical systems in this way. The Cult and Big Pharma agendas are the same because the former controls and owns the latter. 'Health' administrators, doctors, and nursing staff are told to support and parrot the dictated policy or they will face consequences which can include being fired. How sad it's been to see medical staff meekly repeating and imposing Cult policy without question and most of those who can see through the deceit are only willing to speak anonymously off the record. They know what will happen if their identity is known. This has left the courageous few to expose the lies about the 'virus', face masks, overwhelmed hospitals that aren't, and the dangers of the 'vaccine' that isn't a vaccine. When these medical professionals and scientists, some renowned in their field, have taken to the Internet to expose the truth their articles, comments and videos have been deleted by Cult-owned Facebook, Twitter and YouTube. What a real head-shaker to see YouTube videos with leading world scientists and highly qualified medical specialists with an added link underneath to the notorious Cult propaganda website *Wikipedia* to find the 'facts' about the same subject.

HIV – the 'Covid' trial-run

I'll give you an example of the consequences for health and truth that come from censorship and unquestioning belief in official narratives. The story was told by PCR inventor Kary Mullis in his book *Dancing Naked in the Mind Field*. He said that in 1984 he accepted as just another scientific fact that Luc Montagnier of France's Pasteur Institute and Robert Gallo of America's National Institutes of Health had independently discovered that a 'retrovirus' dubbed HIV (human immunodeficiency virus) caused AIDS. They

were, after all, Mullis writes, specialists in retroviruses. This is how the medical and science pyramids work. Something is announced or *assumed* and then becomes an everybody-knows-that purely through repetition of the assumption as if it is fact. Complete crap becomes accepted truth with no supporting evidence and only repetition of the crap. This is how a 'virus' that doesn't exist became the 'virus' that changed the world. The HIV-AIDS fairy story became a multi-billion pound industry and the media poured out propaganda terrifying the world about the deadly HIV 'virus' that caused the lethal AIDS. By then Mullis was working at a lab in Santa Monica, California, to detect retroviruses with his PCR test in blood donations received by the Red Cross. In doing so he asked a virologist where he could find a reference for HIV being the cause of AIDS. 'You don't need a reference,' the virologist said … *'Everybody knows it.'* Mullis said he wanted to quote a reference in the report he was doing and he said he felt a little funny about not knowing the source of such an important discovery when everyone else seemed to. The virologist suggested he cite a report by the Centers for Disease Control and Prevention (CDC) on morbidity and mortality. Mullis read the report, but it only said that an organism had been identified and did not say how. The report did not identify the original scientific work. Physicians, however, *assumed* (key recurring theme) that if the CDC was convinced that HIV caused AIDS then proof must exist. Mullis continues:

I did computer searches. Neither Montagnier, Gallo, nor anyone else had published papers describing experiments which led to the conclusion that HIV probably caused AIDS. I read the papers in Science for which they had become well known as AIDS doctors, but all they had said there was that they had found evidence of a past infection by something which was probably HIV in some AIDS patients.

They found antibodies. Antibodies to viruses had always been considered evidence of past disease, not present disease. Antibodies signaled that the virus had been defeated. The patient had saved himself. There was no indication in these papers that this virus caused a disease. They didn't show that everybody with the antibodies had the disease. In fact they found some healthy people with antibodies.

Mullis asked why their work had been published if Montagnier and Gallo hadn't really found this evidence, and why had they been fighting so hard to get credit for the discovery? He says he was hesitant to write 'HIV is the probable cause of AIDS' until he found published evidence to support that. 'Tens of thousands of scientists and researchers were spending billions of dollars a year doing research based on this idea,' Mullis writes. 'The reason had to be there somewhere; otherwise these people would not have allowed their research to settle into one narrow channel of investigation.' He said he lectured about PCR at numerous meetings where people were always talking about HIV and he asked them how they knew that HIV was the cause of AIDS:

> Everyone said something. Everyone had the answer at home, in the office, in some drawer. They all knew, and they would send me the papers as soon as they got back. But I never got any papers. Nobody ever sent me the news about how AIDS was caused by HIV.

Eventually Mullis was able to ask Montagnier himself about the reference proof when he lectured in San Diego at the grand opening of the University of California AIDS Research Center. Mullis says this was the last time he would ask his question without showing anger. Montagnier said he should reference the CDC report. 'I read it', Mullis said, and it didn't answer the question. 'If Montagnier didn't know the answer who the hell did?' Then one night Mullis was driving when an interview came on National Public Radio with Peter Duesberg, a prominent virologist at Berkeley and a California Scientist of the Year. Mullis says he finally understood why he could not find references that connected HIV to AIDS – *there weren't any!* No one had ever proved that HIV causes AIDS even though it had spawned a multi-billion pound global industry and the media was repeating this as fact every day in their articles and broadcasts terrifying the shit out of people about AIDS and giving the impression that a positive test for HIV (see 'Covid') was a death sentence. Duesberg was a threat to the AIDS gravy train and the agenda that underpinned it. He was therefore abused and castigated after he told the Proceedings of the National Academy of Sciences

there was no good evidence implicating the new 'virus'. Editors rejected his manuscripts and his research funds were deleted. Mullis points out that the CDC has defined AIDS as one of more than 30 diseases *if accompanied* by a positive result on a test that detects antibodies to HIV; but those same diseases are not defined as AIDS cases when antibodies are not detected:

> If an HIV-positive woman develops uterine cancer, for example, she is considered to have AIDS. If she is not HIV positive, she simply has uterine cancer. An HIV-positive man with tuberculosis has AIDS; if he tests negative he simply has tuberculosis. If he lives in Kenya or Colombia, where the test for HIV antibodies is too expensive, he is simply presumed to have the antibodies and therefore AIDS, and therefore he can be treated in the World Health Organization's clinic. It's the only medical help available in some places. And it's free, because the countries that support WHO are worried about AIDS.

Mullis accuses the CDC of continually adding new diseases (see ever more 'Covid symptoms') to the grand AIDS definition and of virtually doctoring the books to make it appear as if the disease continued to spread. He cites how in 1993 the CDC enormously broadened its AIDS definition and county health authorities were delighted because they received $2,500 per year from the Federal government for every reported AIDS case. Ladies and gentlemen, I have just described, via Kary Mullis, the 'Covid pandemic' of 2020 and beyond. Every element is the same and it's been pulled off in the same way by the same networks.

The 'Covid virus' exists? Okay – prove it. Er … still waiting

What Kary Mullis described with regard to 'HIV' has been repeated with 'Covid'. A claim is made that a new, or 'novel', infection has been found and the entire medical system of the world repeats that as fact exactly as they did with HIV and AIDS. No one in the mainstream asks rather relevant questions such as 'How do you know?' and 'Where is your proof?' The SARS-Cov-2 'virus' and the 'Covid-19 disease' became an overnight 'everybody-knows-that'. The origin could be debated and mulled over, but what you could not suggest was that 'SARS-Cov-2' didn't exist. That would be

ridiculous. 'Everybody knows' the 'virus' exists. Well, I didn't for one along with American proper doctors like Andrew Kaufman and Tom Cowan and long-time American proper journalist Jon Rappaport. We dared to pursue the obvious and simple question: 'Where's the evidence?' The overwhelming majority in medicine, journalism and the general public did not think to ask that. After all, *everyone knew* there was a new 'virus'. Everyone was saying so and I heard it on the BBC. Some would eventually argue that the 'deadly virus' was nothing like as deadly as claimed, but few would venture into the realms of its very existence. Had they done so they would have found that the evidence for that claim had gone AWOL as with HIV causes AIDS. In fact, not even that. For something to go AWOL it has to exist in the first place and scientific proof for a 'SARS-Cov-2' can be filed under nothing, nowhere and zilch.

Dr Andrew Kaufman is a board-certified forensic psychiatrist in New York State, a Doctor of Medicine and former Assistant Professor and Medical Director of Psychiatry at SUNY Upstate Medical University, and Medical Instructor of Hematology and Oncology at the Medical School of South Carolina. He also studied biology at the Massachusetts Institute of Technology (MIT) and trained in Psychiatry at Duke University. Kaufman is retired from allopathic medicine, but remains a consultant and educator on natural healing, I saw a video of his very early on in the 'Covid' hoax in which he questioned claims about the 'virus' in the absence of any supporting evidence and with plenty pointing the other way. I did everything I could to circulate his work which I felt was asking the pivotal questions that needed an answer. I can recommend an excellent pull-together interview he did with the website The Last Vagabond entitled *Dr Andrew Kaufman: Virus Isolation, Terrain Theory and Covid-19* and his website is andrewkaufmanmd.com. Kaufman is not only a forensic psychiatrist; he is forensic in all that he does. He always reads original scientific papers, experiments and studies instead of second-third-fourth-hand reports about the 'virus' in the media which are repeating the repeated repetition of the narrative. When he did so with the original Chinese 'virus' papers Kaufman

realised that there was no evidence of a 'SARS-Cov-2'. They had never – from the start – shown it to exist and every repeat of this claim worldwide was based on the accepted existence of proof that was nowhere to be found – see Kary Mullis and HIV. Here we go again.

Let's postulate

Kaufman discovered that the Chinese authorities immediately concluded that the cause of an illness that broke out among about 200 initial patients in Wuhan was a 'new virus' when there were no grounds to make that conclusion. The alleged 'virus' was not isolated from other genetic material in their samples and then shown through a system known as Koch's postulates to be the causative agent of the illness. The world was told that the SARS-Cov-2 'virus' caused a disease they called 'Covid-19' which had 'flu-like' symptoms and could lead to respiratory problems and pneumonia. If it wasn't so tragic it would almost be funny. *'Flu-like' symptoms'? Pneumonia? Respiratory disease?* What in *CHINA* and particularly in *Wuhan*, one of the most polluted cities in the world with a resulting epidemic of respiratory disease?? Three hundred thousand people get pneumonia in China every year and there are nearly a billion cases worldwide of 'flu-like symptoms'. These have a whole range of causes – including pollution in Wuhan – but no other possibility was credibly considered in late 2019 when the world was told there was a new and deadly 'virus'. The global prevalence of pneumonia and 'flu-like systems' gave the Cult networks unlimited potential to re-diagnose these other causes as the mythical 'Covid-19' and that is what they did from the very start. Kaufman revealed how Chinese medical and science authorities (all subordinates to the Cult-owned communist government) took genetic material from the lungs of only a few of the first patients. The material contained their own cells, bacteria, fungi and other microorganisms living in their bodies. The only way you could prove the existence of the 'virus' and its responsibility for the alleged 'Covid-19' was to isolate the virus from all the other material – a process also known as 'purification' – and

then follow the postulates sequence developed in the late 19th century by German physician and bacteriologist Robert Koch which became the 'gold standard' for connecting an alleged causation agent to a disease:

1. The microorganism (bacteria, fungus, virus, etc.) must be present in every case of the disease and all patients must have the same symptoms. It must also *not be present in healthy individuals*.

2. The microorganism must be isolated from the host with the disease. If the microorganism is a bacteria or fungus it must be grown in a pure culture. If it is a virus, it must be purified (i.e. containing no other material except the virus particles) from a clinical sample.

3. The specific disease, with all of its characteristics, must be reproduced when the infectious agent (the purified virus or a pure culture of bacteria or fungi) is inoculated into a healthy, susceptible host.

4. The microorganism must be recoverable from the experimentally infected host as in step 2.

Not one of these criteria has been met in the case of 'SARS-Cov-2' and 'Covid-19'. Not ONE. *EVER*. Robert Koch refers to bacteria and not viruses. What are called 'viral particles' are so minute (hence masks are useless by any definition) that they could only be seen after the invention of the electron microscope in the 1930s and can still only be observed through that means. American bacteriologist and virologist Thomas Milton Rivers, the so-called 'Father of Modern Virology' who was very significantly director of the Rockefeller Institute for Medical Research in the 1930s, developed a less stringent version of Koch's postulates to identify 'virus' causation known as 'Rivers criteria'. 'Covid' did not pass that process either. Some even doubt whether any 'virus' can be isolated from other particles containing genetic material in the Koch method. Freedom of Information requests in many countries asking for scientific proof that the 'Covid virus' has been purified and isolated and shown to exist have all come back with a 'we don't have that' and when this happened with a request to the UK Department of Health they added this comment:

However, outside of the scope of the [Freedom of Information Act] and on a discretionary basis, the following information has been advised to us, which may be of interest. Most infectious diseases are caused by viruses, bacteria or fungi. Some bacteria or fungi have the capacity to grow on their own in isolation, for example in colonies on a petri dish. Viruses are different in that they are what we call 'obligate pathogens' – that is, they cannot survive or reproduce without infecting a host ...

… For some diseases, it is possible to establish causation between a microorganism and a disease by isolating the pathogen from a patient, growing it in pure culture and reintroducing it to a healthy organism. These are known as 'Koch's postulates' and were developed in 1882. However, as our understanding of disease and different disease-causing agents has advanced, these are no longer the method for determining causation [Andrew Kaufman asks why in that case are there two published articles falsely claiming to satisfy Koch's postulates].

It has long been known that viral diseases cannot be identified in this way as viruses cannot be grown in 'pure culture'. When a patient is tested for a viral illness, this is normally done by looking for the presence of antigens, or viral genetic code in a host with molecular biology techniques [Kaufman asks how you could know the origin of these chemicals without having a pure culture for comparison].

For the record 'antigens' are defined so:

Invading microorganisms have antigens on their surface that the human body can recognise as being foreign – meaning not belonging to it. When the body recognises a foreign antigen, lymphocytes (white blood cells) produce antibodies, which are complementary in shape to the antigen.

Notwithstanding that this is open to question in relation to 'SARS-Cov-2' the presence of 'antibodies' can have many causes and they are found in people that are perfectly well. Kary Mullis said: 'Antibodies … had always been considered evidence of past disease, not present disease.'

'Covid' really is a *computer* 'virus'

Where the UK Department of Health statement says 'viruses' are now 'diagnosed' through a 'viral genetic code in a host with molecular biology techniques', they mean … *the PCR test* which its inventor said cannot test for infectious disease. They have no credible method of connecting a 'virus' to a disease and we will see that there is no scientific proof that any 'virus' causes any disease or there is any such thing as a 'virus' in the way that it is described. Tenacious Canadian researcher Christine Massey and her team made

some 40 Freedom of Information requests to national public health agencies in different countries asking for proof that SARS-CoV-2 has been isolated and not one of them could supply that information. Massey said of her request in Canada: 'Freedom of Information reveals Public Health Agency of Canada has no record of 'SARS-COV-2' isolation performed by anyone, anywhere, ever.' If you accept the comment from the UK Department of Health it's because they can't isolate a 'virus'. Even so many 'science' papers claimed to have isolated the 'Covid virus' until they were questioned and had to admit they hadn't. A reply from the Robert Koch Institute in Germany was typical: 'I am not aware of a paper which purified isolated SARS-CoV-2.' So what the hell was Christian Drosten and his gang using to design the 'Covid' testing protocol that has produced all the illusory Covid' cases and 'Covid' deaths when the head of the Chinese version of the CDC admitted there was a problem right from the start in that the 'virus' had never been isolated/purified? Breathe deeply: What they are calling 'Covid' is actually created by a *computer program* i.e. *they made it up* – er, that's it. They took lung fluid, with many sources of genetic material, from one single person alleged to be infected with Covid-19 by a PCR test which they *claimed*, without clear evidence, contained a 'virus'. They used several computer programs to create a model of a theoretical virus genome sequence from more than fifty-six million small sequences of RNA, each of an unknown source, assembling them like a puzzle with no known solution. The computer filled in the gaps with sequences from bits in the gene bank to make it look like a bat SARS-like coronavirus! A wave of the magic wand and poof, an in silico (computer-generated) genome, a scientific fantasy, was created. UK health researcher Dr Kevin Corbett made the same point with this analogy:

> … It's like giving you a few bones and saying that's your fish. It could be any fish. Not even a skeleton. Here's a few fragments of bones. That's your fish … It's all from gene bank and the bits of the virus sequence that weren't there they made up.
>
> They synthetically created them to fill in the blanks. That's what genetics is; it's a code. So it's ABBBCCDDD and you're missing some what you think is EEE so you put it in. It's all

synthetic. You just manufacture the bits that are missing. This is the end result of the geneticization of virology. This is basically a computer virus.

Further confirmation came in an email exchange between British citizen journalist Frances Leader and the government's Medicines & Healthcare Products Regulatory Agency (the Gates-funded MHRA) which gave emergency permission for untested 'Covid vaccines' to be used. The agency admitted that the 'vaccine' is not based on an isolated 'virus', but comes from a *computer-generated model*. Frances Leader was naturally banned from Cult-owned fascist Twitter for making this exchange public. The process of creating computer-generated alleged 'viruses' is called 'in silico' or 'in silicon' – computer chips – and the term 'in silico' is believed to originate with biological experiments using only a computer in 1989. 'Vaccines' involved with 'Covid' are also produced 'in silico' or by computer not a natural process. If the original 'virus' is nothing more than a made-up computer model how can there be 'new variants' of something that never existed in the first place? They are not new 'variants'; they are new *computer models* only minutely different to the original program and designed to further terrify the population into having the 'vaccine' and submitting to fascism. You want a 'new variant'? Click, click, enter – there you go. Tell the medical profession that you have discovered a 'South African variant', 'UK variants' or a 'Brazilian variant' and in the usual HIV-causes-AIDS manner they will unquestioningly repeat it with no evidence whatsoever to support these claims. They will go on television and warn about the dangers of 'new variants' while doing nothing more than repeating what they have been told to be true and knowing that any deviation from that would be career suicide. Big-time insiders will know it's a hoax, but much of the medical community is clueless about the way they are being played and themselves play the public without even being aware they are doing so. What an interesting 'coincidence' that AstraZeneca and Oxford University were conducting 'Covid vaccine trials' in the three countries – the UK, South Africa and Brazil – where the first three 'variants' were claimed to have 'broken out'.

Here's your 'virus' – it's a unicorn

Dr Andrew Kaufman presented a brilliant analysis describing how the 'virus' was imagined into fake existence when he dissected an article published by *Nature* and written by 19 authors detailing *alleged* 'sequencing of a complete viral genome' of the 'new SARS-CoV-2 virus'. This computer-modelled *in silico* genome was used as a template for all subsequent genome sequencing experiments that resulted in the so-called variants which he said now number more than *6,000*. The fake genome was constructed from more than 56 million individual short strands of RNA. Those little pieces were assembled into longer pieces by finding areas of overlapping sequences. The computer programs created over two million possible combinations from which the authors simply chose the longest one. They then compared this to a 'bat virus' and the computer 'alignment' rearranged the sequence and filled in the gaps! They called this computer-generated abomination the 'complete genome'. Dr Tom Cowan, a fellow medical author and collaborator with Kaufman, said such computer-generation constitutes scientific fraud and he makes this superb analogy:

> Here is an equivalency: A group of researchers claim to have found a unicorn because they found a piece of a hoof, a hair from a tail, and a snippet of a horn. They then add that information into a computer and program it to re-create the unicorn, and they then claim this computer re-creation is the real unicorn. Of course, they had never actually seen a unicorn so could not possibly have examined its genetic makeup to compare their samples with the actual unicorn's hair, hooves and horn.
>
> The researchers claim they decided which is the real genome of SARS-CoV-2 by 'consensus', sort of like a vote. Again, different computer programs will come up with different versions of the imaginary 'unicorn', so they come together as a group and decide which is the real imaginary unicorn.

This is how the 'virus' that has transformed the world was brought into fraudulent 'existence'. Extraordinary, yes, but as the Nazis said the bigger the lie the more will believe it. Cowan, however, wasn't finished and he went on to identify what he called the real blockbuster in the paper. He quotes this section from a paper written

by virologists and published by the CDC and then explains what it means:

> Therefore, we examined the capacity of SARS-CoV-2 to infect and replicate in several common primate and human cell lines, including human adenocarcinoma cells (A549), human liver cells (HUH 7.0), and human embryonic kidney cells (HEK-293T). In addition to Vero E6 and Vero CCL81 cells. ... Each cell line was inoculated at high multiplicity of infection and examined 24h post-infection.
>
> No CPE was observed in any of the cell lines except in Vero cells, which grew to greater than 10 to the 7th power at 24 h post-infection. In contrast, HUH 7.0 and 293T showed only modest viral replication, and A549 cells were incompatible with SARS CoV-2 infection.

Cowan explains that when virologists attempt to prove infection they have three possible 'hosts' or models on which they can test. The first was humans. Exposure to humans was generally not done for ethical reasons and has never been done with SARS-CoV-2 or any coronavirus. The second possible host was animals. Cowan said that forgetting for a moment that they never actually use purified virus when exposing animals they do use solutions that they *claim* contain the virus. Exposure to animals has been done with SARS-CoV-2 in an experiment involving mice and this is what they found: *None of the wild (normal) mice got sick*. In a group of genetically-modified mice, a statistically insignificant number lost weight and had slightly bristled fur, but they experienced nothing like the illness called 'Covid-19'. Cowan said the third method – the one they mostly rely on – is to inoculate solutions they *say* contain the virus onto a variety of tissue cultures. This process had never been shown to kill tissue *unless* the sample material was starved of nutrients and poisoned as *part of the process*. Yes, incredibly, in tissue experiments designed to show the 'virus' is responsible for killing the tissue they starve the tissue of nutrients and add toxic drugs including antibiotics and they do not have control studies to see if it's the starvation and poisoning that is degrading the tissue rather than the 'virus' they allege to be in there somewhere. You want me to pinch you? Yep, I understand. Tom Cowan said this about the whole nonsensical farce as he explains what that quote from the CDC paper really means:

The shocking thing about the above quote is that using their own methods, the virologists found that solutions containing SARS-CoV-2 – even in high amounts – were NOT, I repeat NOT, infective to any of the three human tissue cultures they tested. In plain English, this means they proved, on their terms, that this 'new coronavirus' is not infectious to human beings. It is ONLY infective to monkey kidney cells, and only then when you add two potent drugs (gentamicin and amphotericin), known to be toxic to kidneys, to the mix.

My friends, read this again and again. These virologists, published by the CDC, performed a clear proof, on their terms, showing that the SARS-CoV-2 virus is harmless to human beings. That is the only possible conclusion, but, unfortunately, this result is not even mentioned in their conclusion. They simply say they can provide virus stocks cultured only on monkey Vero cells, thanks for coming.

Cowan concluded: 'If people really understood how this "science" was done, I would hope they would storm the gates and demand honesty, transparency and truth.' Dr Michael Yeadon, former Vice President and Chief Scientific Adviser at drug giant Pfizer has been a vocal critic of the 'Covid vaccine' and its potential for multiple harm. He said in an interview in April, 2021, that 'not one [vaccine] has the virus. He was asked why vaccines normally using a 'dead' version of a disease to activate the immune system were not used for 'Covid' and instead we had the synthetic methods of the 'mRNA Covid vaccine'. Yeadon said that to do the former 'you'd have to have some of [the virus] wouldn't you?' He added: 'No-one's got any – seriously.' Yeadon said that surely they couldn't have fooled the whole world for a year without having a virus, 'but oddly enough ask around – no one's got it'. He didn't know why with all the 'great labs' around the world that the virus had not been isolated – 'Maybe they've been too busy running bad PCR tests and vaccines that people don't need.' What is today called 'science' is not 'science' at all. Science is no longer what is, but whatever people can be manipulated to *believe* that it is. Real science has been hijacked by the Cult to dispense and produce the 'expert scientists' and contentions that suit the agenda of the Cult. How big-time this has happened with the 'Covid' hoax which is entirely based on fake science delivered by fake 'scientists' and fake 'doctors'. The human-caused climate change hoax is also entirely based on fake science delivered by fake 'scientists' and fake 'climate experts'. In both cases real

scientists, climate experts and doctors have their views suppressed and deleted by the Cult-owned science establishment, media and Silicon Valley. This is the 'science' that politicians claim to be 'following' and a common denominator of 'Covid' and climate are Cult psychopaths Bill Gates and his mate Klaus Schwab at the Gates-funded World Economic Forum. But, don't worry, it's all just a coincidence and absolutely nothing to worry about. *Zzzzzzzz*.

What is a 'virus' REALLY?

Dr Tom Cowan is one of many contesting the very existence of viruses let alone that they cause disease. This is understandable when there is no scientific evidence for a disease-causing 'virus'. German virologist Dr Stefan Lanka won a landmark case in 2017 in the German Supreme Court over his contention that there is no such thing as a measles virus. He had offered a big prize for anyone who could prove there is and Lanka won his case when someone sought to claim the money. There is currently a prize of more than 225,000 euros on offer from an Isolate Truth Fund for anyone who can prove the isolation of SARS-CoV-2 and its genetic substance. Lanka wrote in an article headed 'The Misconception Called Virus' that scientists think a 'virus' is causing tissue to become diseased and degraded when in fact it is the *processes they are using* which do that – not a 'virus'. Lanka has done an important job in making this point clear as Cowan did in his analysis of the CDC paper. Lanka says that all claims about viruses as disease-causing pathogens are wrong and based on 'easily recognisable, understandable and verifiable misinterpretations.' Scientists believed they were working with 'viruses' in their laboratories when they were really working with 'typical particles of specific dying tissues or cells …' Lanka said that the tissue decaying process claimed to be caused by a 'virus' still happens when no alleged 'virus' is involved. It's the *process* that does the damage and not a 'virus'. The genetic sample is deprived of nutrients, removed from its energy supply through removal from the body and then doused in toxic antibiotics to remove any bacteria. He confirms again that establishment scientists do not (pinch me)

conduct control experiments to see if this is the case and if they did they would see the claims that 'viruses' are doing the damage is nonsense. He adds that during the measles 'virus' court case he commissioned an independent laboratory to perform just such a control experiment and the result was that the tissues and cells died in the exact same way as with alleged 'infected' material. This is supported by a gathering number of scientists, doctors and researchers who reject what is called 'germ theory' or the belief in the body being infected by contagious sources emitted by other people. Researchers Dawn Lester and David Parker take the same stance in their highly-detailed and sourced book *What Really Makes You Ill – Why everything you thought you knew about disease is wrong* which was recommended to me by a number of medical professionals genuinely seeking the truth. Lester and Parker say there is no provable scientific evidence to show that a 'virus' can be transmitted between people or people and animals or animals and people:

> The definition also claims that viruses are the cause of many diseases, as if this has been definitively proven. But this is not the case; there is no original scientific evidence that definitively demonstrates that any virus is the cause of any disease. The burden of proof for any theory lies with those who proposed it; but none of the existing documents provides 'proof' that supports the claim that 'viruses' are pathogens.

Dr Tom Cowan employs one of his clever analogies to describe the process by which a 'virus' is named as the culprit for a disease when what is called a 'virus' is only material released by cells detoxing themselves from infiltration by chemical or radiation poisoning. The tidal wave of technologically-generated radiation in the 'smart' modern world plus all the toxic food and drink are causing this to happen more than ever. Deluded 'scientists' misread this as a gathering impact of what they wrongly label 'viruses'.

Paper can infect houses

Cowan said in an article for davidicke.com – with his tongue only mildly in his cheek – that he believed he had made a tremendous

discovery that may revolutionise science. He had discovered that small bits of paper are alive, 'well alive-ish', can 'infect' houses, and then reproduce themselves inside the house. The result was that this explosion of growth in the paper inside the house causes the house to explode, blowing it to smithereens. His evidence for this new theory is that in the past months he had carefully examined many of the houses in his neighbourhood and found almost no scraps of paper on the lawns and surrounds of the house. There was an occasional stray label, but nothing more. Then he would return to these same houses a week or so later and with a few, not all of them, particularly the old and decrepit ones, he found to his shock and surprise they were littered with stray bits of paper. He knew then that the paper had infected these houses, made copies of itself, and blew up the house. A young boy on a bicycle at one of the sites told him he had seen a demolition crew using dynamite to explode the house the previous week, but Cowan dismissed this as the idle thoughts of silly boys because 'I was on to something big'. He was on to how 'scientists' mistake genetic material in the detoxifying process for something they call a 'virus'. Cowan said of his house and paper story:

If this sounds crazy to you, it's because it should. This scenario is obviously nuts. But consider this admittedly embellished, for effect, current viral theory that all scientists, medical doctors and virologists currently believe.

He takes the example of the 'novel SARS-Cov2' virus to prove the point. First they take someone with an undefined illness called 'Covid-19' and don't even attempt to find any virus in their sputum. Never mind the scientists still describe how this 'virus', which they have not located attaches to a cell receptor, injects its genetic material, in 'Covid's' case, RNA, into the cell. The RNA once inserted exploits the cell to reproduce itself and makes 'thousands, nay millions, of copies of itself … Then it emerges victorious to claim its next victim':

If you were to look in the scientific literature for proof, actual scientific proof, that uniform SARS-CoV2 viruses have been properly isolated from the sputum of a sick person, that actual spike proteins could be seen protruding from the virus (which has not been found), you would find that such evidence doesn't exist.

If you go looking in the published scientific literature for actual pictures, proof, that these spike proteins or any viral proteins are ever attached to any receptor embedded in any cell membrane, you would also find that no such evidence exists. If you were to look for a video or documented evidence of the intact virus injecting its genetic material into the body of the cell, reproducing itself and then emerging victorious by budding off the cell membrane, you would find that no such evidence exists.

The closest thing you would find is electron micrograph pictures of cellular particles, possibly attached to cell debris, both of which to be seen were stained by heavy metals, a process that completely distorts their architecture within the living organism. This is like finding bits of paper stuck to the blown-up bricks, thereby proving the paper emerged by taking pieces of the bricks on its way out.

The Enders baloney

Cowan describes the 'Covid' story as being just as make-believe as his paper story and he charts back this fantasy to a Nobel Prize winner called John Enders (1897-1985), an American biomedical scientist who has been dubbed 'The Father of Modern Vaccines'. Enders is claimed to have 'discovered' the process of the viral culture which 'proved' that a 'virus' caused measles. Cowan explains how Enders did this 'by using the EXACT same procedure that has been followed by every virologist to find and characterize every new virus since 1954'. Enders took throat swabs from children with measles and immersed them in 2ml of milk. Penicillin (100u/ml) and the antibiotic streptomycin (50,g/ml) were added and the whole mix was centrifuged – rotated at high speed to separate large cellular debris from small particles and molecules as with milk and cream, for example. Cowan says that if the aim is to find little particles of genetic material ('viruses') in the snot from children with measles it would seem that the last thing you would do is mix the snot with other material – milk –that also has genetic material. 'How are you ever going to know whether whatever you found came from the snot or the milk?' He points out that streptomycin is a 'nephrotoxic' or poisonous-to-the-kidney drug. You will see the relevance of that

shortly. Cowan says that it gets worse, much worse, when Enders describes the culture medium upon which the virus 'grows': 'The culture medium consisted of bovine amniotic fluid (90%), beef embryo extract (5%), horse serum (5%), antibiotics and phenol red as an indicator of cell metabolism.' Cowan asks incredulously: 'Did he just say that the culture medium also contained fluids and tissues that are themselves rich sources of genetic material?' The genetic cocktail, or 'medium', is inoculated onto tissue and cells from rhesus monkey *kidney* tissue. This is where the importance of streptomycin comes in and currently-used antimicrobials and other drugs that are *poisonous to kidneys* and used in ALL modern viral cultures (e.g. gentamicin, streptomycin, and amphotericin). Cowan asks: 'How are you ever going to know from this witch's brew where any genetic material comes from as we now have five different sources of rich genetic material in our mix?' Remember, he says, that all genetic material, whether from monkey kidney tissues, bovine serum, milk, etc., is made from the exact same components. The same central question returns: 'How are you possibly going to know that it was the virus that killed the kidney tissue and not the toxic antibiotic and starvation rations on which you are growing the tissue?' John Enders answered the question himself – *you can't*:

A second agent was obtained from an uninoculated culture of monkey kidney cells. The cytopathic changes [death of the cells] it induced in the unstained preparations could not be distinguished with confidence from the viruses isolated from measles.

The death of the cells ('cytopathic changes') happened in exactly the same manner, whether they inoculated the kidney tissue with the measles snot or not, Cowan says. 'This is evidence that the destruction of the tissue, the very proof of viral causation of illness, was not caused by anything in the snot because they saw the same destructive effect when the snot was not even used … the cytopathic, i.e., cell-killing, changes come from the process of the culture itself, not from any virus in any snot, period.' Enders quotes in his 1957 paper a virologist called Ruckle as reporting similar findings 'and in addition has isolated an agent from monkey kidney tissue that is so

far indistinguishable from human measles virus'. In other words, Cowan says, these particles called 'measles viruses' are simply and clearly breakdown products of the starved and poisoned tissue. For measles 'virus' see all 'viruses' including the so-called 'Covid virus'. Enders, the 'Father of Modern Vaccines', also said:

There is a potential risk in employing cultures of primate cells for the production of vaccines composed of attenuated virus, since the presence of other agents possibly latent in primate tissues cannot be definitely excluded by any known method.

Cowan further quotes from a paper published in the journal *Viruses* in May, 2020, while the 'Covid pandemic' was well underway in the media if not in reality. 'EVs' here refers to particles of genetic debris from our own tissues, such as exosomes of which more in a moment: 'The remarkable resemblance between EVs and viruses has caused quite a few problems in the studies focused on the analysis of EVs released during viral infections.' Later the paper adds that to date a reliable method that can actually guarantee a complete separation (of EVs from viruses) DOES NOT EXIST. This was published at a time when a fairy tale 'virus' was claimed in total certainty to be causing a fairy tale 'viral disease' called 'Covid-19' – a fairy tale that was already well on the way to transforming human society in the image that the Cult has worked to achieve for so long. Cowan concludes his article:

To summarize, there is no scientific evidence that pathogenic viruses exist. What we think of as 'viruses' are simply the normal breakdown products of dead and dying tissues and cells. When we are well, we make fewer of these particles; when we are starved, poisoned, suffocated by wearing masks, or afraid, we make more.

There is no engineered virus circulating and making people sick. People in laboratories all over the world are making genetically modified products to make people sick. These are called vaccines. There is no virome, no 'ecosystem' of viruses, viruses are not 8%, 50% or 100 % of our genetic material. These are all simply erroneous ideas based on the misconception called a virus.

What is 'Covid'? Load of bollocks

The background described here by Cowan and Lanka was emphasised in the first video presentation that I saw by Dr Andrew Kaufman when he asked whether the 'Covid virus' was in truth a natural defence mechanism of the body called 'exosomes'. These are released by cells when in states of toxicity – see the same themes returning over and over. They are released ever more profusely as chemical and radiation toxicity increases and think of the potential effect therefore of 5G alone as its destructive frequencies infest the human energetic information field with a gathering pace (5G went online in Wuhan in 2019 as the 'virus' emerged). I'll have more about this later. Exosomes transmit a warning to the rest of the body that 'Houston, we have a problem'. Kaufman presented images of exosomes and compared them with 'Covid' under an electron microscope and the similarity was remarkable. They both attach to the same cell receptors (*claimed* in the case of 'Covid'), contain the same genetic material in the form of RNA or ribonucleic acid, and both are found in 'viral cell cultures' with damaged or dying cells. James Hildreth MD, President and Chief Executive Officer of the Meharry Medical College at Johns Hopkins, said: 'The virus is fully an exosome in every sense of the word.' Kaufman's conclusion was that there is no 'virus': 'This entire pandemic is a completely manufactured crisis … there is no evidence of anyone dying from [this] illness.' Dr Tom Cowan and Sally Fallon Morell, authors of *The Contagion Myth*, published a statement with Dr Kaufman in February, 2021, explaining why the 'virus' does not exist and you can read it that in full in the Appendix.

'Virus' theory can be traced to the 'cell theory' in 1858 of German physician Rudolf Virchow (1821-1920) who contended that disease originates from a single cell infiltrated by a 'virus'. Dr Stefan Lanka said that findings and insights with respect to the structure, function and central importance of tissues in the creation of life, which were already known in 1858, comprehensively refute the cell theory. Virchow ignored them. We have seen the part later played by John Enders in the 1950s and Lanka notes that infection theories were only established as a global dogma through the policies and

eugenics of the Third Reich in Nazi Germany (creation of the same Sabbatian cult behind the 'Covid' hoax). Lanka said: 'Before 1933, scientists dared to contradict this theory; after 1933, these critical scientists were silenced'. Dr Tom Cowan's view is that ill-heath is caused by too much of something, too little of something, or toxification from chemicals and radiation – not contagion. We must also highlight as a major source of the 'virus' theology a man still called the 'Father of Modern Virology' – Thomas Milton Rivers (1888-1962). There is no way given the Cult's long game policy that it was a coincidence for the 'Father of Modern Virology' to be director of the Rockefeller Institute for Medical Research from 1937 to 1956 when he is credited with making the Rockefeller Institute a leader in 'viral research'. Cult Rockefellers were the force behind the creation of Big Pharma 'medicine', established the World Health Organisation in 1948, and have long and close associations with the Gates family that now runs the WHO during the pandemic hoax through mega-rich Cult gofer and psychopath Bill Gates.

Only a Renegade Mind can see through all this bullshit by asking the questions that need to be answered, not taking 'no' or prevarication for an answer, and certainly not hiding from the truth in fear of speaking it. Renegade Minds have always changed the world for the better and they will change this one no matter how bleak it may currently appear to be.

CHAPTER SIX

Sequence of deceit

If you tell the truth, you don't have to remember anything
Mark Twain

Against the background that I have laid out this far the sequence that took us from an invented 'virus' in Cult-owned China in late 2019 to the fascist transformation of human society can be seen and understood in a whole new context.

We were told that a deadly disease had broken out in Wuhan and the world media began its campaign (coordinated by behavioural psychologists as we shall see) to terrify the population into unquestioning compliance. We were shown images of Chinese people collapsing in the street which never happened in the West with what was supposed to be the same condition. In the earliest days when alleged cases and deaths were few the fear register was hysterical in many areas of the media and this would expand into the common media narrative across the world. The real story was rather different, but we were never told that. The Chinese government, one of the Cult's biggest centres of global operation, said they had discovered a new illness with flu-like and pneumonia-type symptoms in a city with such toxic air that it is overwhelmed with flu-like symptoms, pneumonia and respiratory disease. Chinese scientists said it was a new – 'novel' – coronavirus which they called Sars-Cov-2 and that it caused a disease they labelled 'Covid-19'. There was no evidence for this and the 'virus' has never to this day been isolated, purified and its genetic code established from that. It

was from the beginning a computer-generated fiction. Stories of Chinese whistleblowers saying the number of deaths was being supressed or that the 'new disease' was related to the Wuhan bio-lab misdirected mainstream and alternative media into cul-de-sacs to obscure the real truth – there was no 'virus'.

Chinese scientists took genetic material from the lung fluid of just a few people and said they had found a 'new' disease when this material had a wide range of content. There was no evidence for a 'virus' for the very reasons explained in the last two chapters. The 'virus' has never been shown to (a) exist and (b) cause any disease. People were diagnosed on symptoms that are so widespread in Wuhan and polluted China and with a PCR test that can't detect infectious disease. On this farce the whole global scam was sold to the rest of the world which would also diagnose respiratory disease as 'Covid-19' from symptoms alone or with a PCR test not testing for a 'virus'. Flu miraculously disappeared *worldwide* in 2020 and into 2021 as it was redesignated 'Covid-19'. It was really the same old flu with its 'flu-like' symptoms attributed to 'flu-like' 'Covid-19'. At the same time with very few exceptions the Chinese response of draconian lockdown and fascism was the chosen weapon to respond across the West as recommended by the Cult-owned Tedros at the Cult-owned World Health Organization run by the Cult-owned Gates. All was going according to plan. Chinese scientists – everything in China is controlled by the Cult-owned government – compared their contaminated RNA lung-fluid material with other RNA sequences and said it appeared to be just under 80 percent identical to the SARS-CoV-1 'virus' claimed to be the cause of the SARS (severe acute respiratory syndrome) 'outbreak' in 2003. They decreed that because of this the 'new virus' had to be related and they called it SARS-CoV-2. There are some serious problems with this assumption and *assumption* was all it was. Most 'factual' science turns out to be assumptions repeated into everyone-knows-that. A match of under 80-percent is meaningless. Dr Kaufman makes the point that there's a *96 percent* genetic correlation between humans and chimpanzees, but 'no one would say our genetic material is part

of the chimpanzee family'. Yet the Chinese authorities were claiming that a much lower percentage, less than 80 percent, proved the existence of a new 'coronavirus'. For goodness sake human DNA is 60 percent similar to a *banana*.

You are feeling sleepy

The entire 'Covid' hoax is a global Psyop, a psychological operation to program the human mind into believing and fearing a complete fantasy. A crucial aspect of this was what *appeared* to happen in Italy. It was all very well streaming out daily images of an alleged catastrophe in Wuhan, but to the Western mind it was still on the other side of the world in a very different culture and setting. A reaction of 'this could happen to me and my family' was still nothing like as intense enough for the mind-doctors. The Cult needed a Western example to push people over that edge and it chose Italy, one of its major global locations going back to the Roman Empire. An Italian 'Covid' crisis was manufactured in a particular area called Lombardy which just happens to be notorious for its toxic air and therefore respiratory disease. Wuhan, China, déjà vu. An hysterical media told horror stories of Italians dying from 'Covid' in their droves and how Lombardy hospitals were being overrun by a tidal wave of desperately ill people needing treatment after being struck down by the 'deadly virus'. Here was the psychological turning point the Cult had planned. Wow, if this is happening in Italy, the Western mind concluded, this indeed could happen to me and my family. Another point is that Italian authorities responded by following the Chinese blueprint so vehemently recommended by the Cult-owned World Health Organization. They imposed fascistic lockdowns on the whole country viciously policed with the help of surveillance drones sweeping through the streets seeking out anyone who escaped from mass house arrest. Livelihoods were destroyed and psychology unravelled in the way we have witnessed since in all lockdown countries. Crucial to the plan was that Italy responded in this way to set the precedent of suspending freedom and imposing fascism in a 'Western liberal democracy'. I emphasised in an

animated video explanation on davidicke.com posted in the summer of 2020 how important it was to the Cult to expand the Chinese lockdown model across the West. Without this, and the bare-faced lie that non-symptomatic people could still transmit a 'disease' they didn't have, there was no way locking down the whole population, sick and not sick, could be pulled off. At just the right time and with no evidence Cult operatives and gofers claimed that people without symptoms could pass on the 'disease'. In the name of protecting the 'vulnerable' like elderly people, who lockdowns would kill by the tens of thousands, we had for the first time healthy people told to isolate as well as the sick. The great majority of people who tested positive had no symptoms because there was nothing wrong with them. It was just a trick made possible by a test not testing for the 'virus'.

Months after my animated video the Gates-funded Professor Neil Ferguson at the Gates-funded Imperial College confirmed that I was right. He didn't say it in those terms, naturally, but he did say it. Ferguson will enter the story shortly for his outrageously crazy 'computer models' that led to Britain, the United States and many other countries following the Chinese and now Italian methods of response. Put another way, following the Cult script. Ferguson said that SAGE, the UK government's scientific advisory group which has controlled 'Covid' policy from the start, wanted to follow the Chinese lockdown model (while they all continued to work and be paid), but they wondered if they could possibly, in Ferguson's words, 'get away with it in Europe'. 'Get away with it'? Who the hell do these moronic, arrogant people think they are? This appalling man Ferguson said that once Italy went into national lockdown they realised they, too, could mimic China:

> It's a communist one-party state, we said. We couldn't get away with it in Europe, we thought … and then Italy did it. And we realised we could. Behind this garbage from Ferguson is a simple fact: Doing the same as China in every country was the plan from the start and Ferguson's 'models' would play a central role in achieving that. It's just a coincidence, of course, and absolutely nothing to worry your little head about.

Oops, sorry, our mistake

Once the Italian segment of the Psyop had done the job it was designed to do a very different story emerged. Italian authorities revealed that 99 percent of those who had 'died from Covid-19' in Italy had one, two, three, or more 'co-morbidities' or illnesses and health problems that could have ended their life. The US Centers for Disease Control and Prevention (CDC) published a figure of 94 percent for Americans dying of 'Covid' while having other serious medical conditions – on average two to three (some five or six) other potential causes of death. In terms of death from an unproven 'virus' I say it is 100 percent. The other one percent in Italy and six percent in the US would presumably have died from 'Covid's' flu-like symptoms with a range of other possible causes in conjunction with a test not testing for the 'virus'. Fox News reported that even more startling figures had emerged in one US county in which 410 of 422 deaths attributed to 'Covid-19' had other potentially deadly health conditions. The Italian National Health Institute said later that the average age of people dying with a 'Covid-19' diagnosis in Italy was about 81. Ninety percent were over 70 with ten percent over 90. In terms of other reasons to die some 80 percent had two or more chronic diseases with half having three or more including cardiovascular problems, diabetes, respiratory problems and cancer. Why is the phantom 'Covid-19' said to kill overwhelmingly old people and hardly affect the young? Old people continually die of many causes and especially respiratory disease which you can re-diagnose 'Covid-19' while young people die in tiny numbers by comparison and rarely of respiratory disease. Old people 'die of Covid' because they die of other things that can be redesignated 'Covid' and it really is that simple.

Flu has flown

The blueprint was in place. Get your illusory 'cases' from a test not testing for the 'virus' and redesignate other causes of death as 'Covid-19'. You have an instant 'pandemic' from something that is nothing more than a computer-generated fiction. With near-on a

billion people having 'flu-like' symptoms every year the potential was limitless and we can see why flu quickly and apparently miraculously disappeared *worldwide* by being diagnosed 'Covid-19'. The painfully bloody obvious was explained away by the childlike media in headlines like this in the UK '*Independent*': 'Not a single case of flu detected by Public Health England this year as Covid restrictions suppress virus'. I kid you not. The masking, social distancing and house arrest that did not make the 'Covid virus' disappear somehow did so with the 'flu virus'. Even worse the article, by a bloke called Samuel Lovett, suggested that maybe the masking, sanitising and other 'Covid' measures should continue to keep the flu away. With a ridiculousness that disturbs your breathing (it's 'Covid-19') the said Lovett wrote: 'With widespread social distancing and mask-wearing measures in place throughout the UK, the usual routes of transmission for influenza have been blocked.' He had absolutely no evidence to support that statement, but look at the consequences of him acknowledging the obvious. With flu not disappearing at all and only being relabelled 'Covid-19' he would have to contemplate that 'Covid' was a hoax on a scale that is hard to imagine. You need guts and commitment to truth to even go there and that's clearly something Samuel Lovett does not have in abundance. He would never have got it through the editors anyway.

Tens of thousands die in the United States alone every winter from flu including many with pneumonia complications. CDC figures record *45 million* Americans diagnosed with flu in 2017-2018 of which 61,000 died and some reports claim 80,000. Where was the same hysteria then that we have seen with 'Covid-19'? Some 250,000 Americans are admitted to hospital with pneumonia every year with about 50,000 cases proving fatal. About 65 million suffer respiratory disease every year and three million deaths makes this the third biggest cause of death worldwide. You only have to redesignate a portion of all these people 'Covid-19' and you have an instant global pandemic or the *appearance* of one. Why would doctors do this? They are told to do this and all but a few dare not refuse those who must be obeyed. Doctors in general are not researching their own

knowledge and instead take it direct and unquestioned from the authorities that own them and their careers. The authorities say they must now diagnose these symptoms 'Covid-19' and not flu, or whatever, and they do it. Dark suits say put 'Covid-19' on death certificates no matter what the cause of death and the doctors do it. Renegade Minds don't fall for the illusion that doctors and medical staff are all highly-intelligent, highly-principled, seekers of medical truth. *Some are*, but not the majority. They are repeaters, gofers, and yes sir, no sir, purveyors of what the system demands they purvey. The 'Covid' con is not merely confined to diseases of the lungs. Instructions to doctors to put 'Covid-19' on death certificates for anyone dying of *anything* within 28 days (or much more) of a positive test not testing for the 'virus' opened the floodgates. The term dying *with* 'Covid' and not *of* 'Covid' was coined to cover the truth. Whether it was a *with* or an *of* they were all added to the death numbers attributed to the 'deadly virus' compiled by national governments and globally by the Gates-funded Johns Hopkins operation in the United States that was so involved in those 'pandemic' simulations. Fraudulent deaths were added to the ever-growing list of fraudulent 'cases' from false positives from a false test. No wonder Professor Walter Ricciardi, scientific advisor to the Italian minister of health, said after the Lombardy hysteria had done its job that 'Covid' death rates were due to Italy having the second oldest population in the world and to *how hospitals record deaths*:

> The way in which we code deaths in our country is very generous in the sense that all the people who die in hospitals with the coronavirus are deemed to be dying of the coronavirus. On re-evaluation by the National Institute of Health, only 12 per cent of death certificates have shown a direct causality from coronavirus, while 88 per cent of patients who have died have at least one pre-morbidity – many had two or three.

This is extraordinary enough when you consider the propaganda campaign to use Italy to terrify the world, but how can they even say twelve percent were genuine when the 'virus' has not been shown to exist, its 'code' is a computer program, and diagnosis comes from a test not testing for it? As in China, and soon the world, 'Covid-19' in

Italy was a redesignation of diagnosis. Lies and corruption were to become the real 'pandemic' fuelled by a pathetically-compliant medical system taking its orders from the tiny few at the top of their national hierarchy who answered to the World Health Organization which answers to Gates and the Cult. Doctors were told – ordered – to diagnose a particular set of symptoms 'Covid-19' and put that on the death certificate for any cause of death if the patient had tested positive with a test not testing for the virus or had 'Covid' symptoms like the flu. The United States even introduced big financial incentives to manipulate the figures with hospitals receiving £4,600 from the Medicare system for diagnosing someone with regular pneumonia, $13,000 if they made the diagnosis from the same symptoms 'Covid-19' pneumonia, and $39, 000 if they put a 'Covid' diagnosed patient on a ventilator that would almost certainly kill them. A few – painfully and pathetically few – medical whistleblowers revealed (before Cult-owned YouTube deleted their videos) that they had been instructed to 'let the patient crash' and put them straight on a ventilator instead of going through a series of far less intrusive and dangerous methods as they would have done before the pandemic hoax began and the financial incentives kicked in. We are talking cold-blooded murder given that ventilators are so damaging to respiratory systems they are usually the last step before heaven awaits. Renegade Minds never fall for the belief that people in white coats are all angels of mercy and cannot be full-on psychopaths. I have explained in detail in *The Answer* how what I am describing here played out across the world coordinated by the World Health Organization through the medical hierarchies in almost every country.

Medical scientist calls it

Information about the non-existence of the 'virus' began to emerge for me in late March, 2020, and mushroomed after that. I was sent an email by Sir Julian Rose, a writer, researcher, and organic farming promotor, from a medical scientist friend of his in the United States. Even at that early stage in March the scientist was able to explain

how the 'Covid' hoax was being manipulated. He said there were no reliable tests for a specific 'Covid-19 virus' and nor were there any reliable agencies or media outlets for reporting numbers of actual 'Covid-19' cases. We have seen in the long period since then that he was absolutely right. 'Every action and reaction to Covid-19 is based on totally flawed data and we simply cannot make accurate assessments,' he said. Most people diagnosed with 'Covid-19' were showing nothing more than cold and flu-like symptoms 'because most coronavirus strains *are* nothing more than cold/flu-like symptoms'. We had farcical situations like an 84-year-old German man testing positive for 'Covid-19' and his nursing home ordered to quarantine only for him to be found to have a common cold. The scientist described back then why PCR tests and what he called the 'Mickey Mouse test kits' were useless for what they were claimed to be identifying. 'The idea these kits can isolate a specific virus like Covid-19 is nonsense,' he said. Significantly, he pointed out that 'if you want to create a totally false panic about a totally false pandemic – pick a coronavirus'. This is exactly what the Cult-owned Gates, World Economic Forum and Johns Hopkins University did with their Event 201 'simulation' followed by their real-life simulation called the 'pandemic'. The scientist said that all you had to do was select the sickest of people with respiratory-type diseases in a single location – 'say Wuhan' – and administer PCR tests to them. You can then claim that anyone showing 'viral sequences' similar to a coronavirus 'which will inevitably be quite a few' is suffering from a 'new' disease:

> Since you already selected the sickest flu cases a fairly high proportion of your sample will go on to die. You can then say this 'new' virus has a CFR [case fatality rate] higher than the flu and use this to infuse more concern and do more tests which will of course produce more 'cases', which expands the testing, which produces yet more 'cases' and so on and so on. Before long you have your 'pandemic', and all you have done is use a simple test kit trick to convert the worst flu and pneumonia cases into something new that doesn't ACTUALLY EXIST [my emphasis].

He said that you then 'just run the same scam in other countries' and make sure to keep the fear message running high 'so that people

will feel panicky and less able to think critically'. The only problem to overcome was the fact *there is no* actual new deadly pathogen and only regular sick people. This meant that deaths from the 'new deadly pathogen' were going to be way too low for a real new deadly virus pandemic, but he said this could be overcome in the following ways – all of which would go on to happen:

1. You can claim this is just the beginning and more deaths are imminent [you underpin this with fantasy 'computer projections']. Use this as an excuse to quarantine everyone and then claim the quarantine prevented the expected millions of dead.

2. You can [say that people] 'minimizing' the dangers are irresponsible and bully them into not talking about numbers.

3. You can talk crap about made up numbers hoping to blind people with pseudoscience.

4. You can start testing well people (who, of course, will also likely have shreds of coronavirus [RNA] in them) and thus inflate your 'case figures' with 'asymptomatic carriers' (you will of course have to spin that to sound deadly even though any virologist knows the more symptom-less cases you have the less deadly is your pathogen).

The scientist said that if you take these simple steps 'you can have your own entirely manufactured pandemic up and running in weeks'. His analysis made so early in the hoax was brilliantly prophetic of what would actually unfold. Pulling all the information together in these recent chapters we have this is simple 1, 2, 3, of how you can delude virtually the entire human population into believing in a 'virus' that doesn't exist:

- A 'Covid case' is someone who tests positive with a test not testing for the 'virus'.

- A 'Covid death' is someone who dies of *any cause* within 28 days (or much longer) of testing positive with a test not testing for the 'virus.

- Asymptomatic means there is nothing wrong with you, but they claim you can pass on what you don't have to justify locking

down (quarantining) healthy people in totality.

The foundations of the hoax are that simple. A study involving ten million people in Wuhan, published in November, 2020, demolished the whole lie about those without symptoms passing on the 'virus'. They found '300 asymptomatic cases' and traced their contacts to find that not one of them was detected with the 'virus'. 'Asymptomatic' patients and their contacts were isolated for no less than two weeks and nothing changed. I know it's all crap, but if you are going to claim that those without symptoms can transmit 'the virus' then you must produce evidence for that and they never have. Even World Health Organization official Dr Maria Van Kerkhove, head of the emerging diseases and zoonosis unit, said as early as June, 2020, that she doubted the validity of asymptomatic transmission. She said that 'from the data we have, it still seems to be rare that an asymptomatic person actually transmits onward to a secondary individual' and by 'rare' she meant that she couldn't cite any case of asymptomatic transmission.

The Ferguson factor

The problem for the Cult as it headed into March, 2020, when the script had lockdown due to start, was that despite all the manipulation of the case and death figures they still did not have enough people alleged to have died from 'Covid' to justify mass house arrest. This was overcome in the way the scientist described: 'You can claim this is just the beginning and more deaths are imminent … Use this as an excuse to quarantine everyone and then claim the quarantine prevented the expected millions of dead.' Enter one Professor Neil Ferguson, the Gates-funded 'epidemiologist' at the Gates-funded Imperial College in London. Ferguson is Britain's Christian Drosten in that he has a dire record of predicting health outcomes, but is still called upon to advise government on the next health outcome when another 'crisis' comes along. This may seem to be a strange and ridiculous thing to do. Why would you keep turning for policy guidance to people who have a history of being

monumentally wrong? Ah, but it makes sense from the Cult point of view. These 'experts' keep on producing predictions that suit the Cult agenda for societal transformation and so it was with Neil Ferguson as he revealed his horrific (and clearly insane) computer model predictions that allowed lockdowns to be imposed in Britain, the United States and many other countries. Ferguson does not have even an A-level in biology and would appear to have no formal training in computer modelling, medicine or epidemiology, according to Derek Winton, an MSc in Computational Intelligence. He wrote an article somewhat aghast at what Ferguson did which included taking no account of respiratory disease 'seasonality' which means it is far worse in the winter months. Who would have thought that respiratory disease could be worse in the winter? Well, certainly not Ferguson.

The massively China-connected Imperial College and its bizarre professor provided the excuse for the long-incubated Chinese model of human control to travel westward at lightning speed. Imperial College confirms on its website that it collaborates with the Chinese Research Institute; publishes more than 600 research papers every year with Chinese research institutions; has 225 Chinese staff; 2,600 Chinese students – the biggest international group; 7,000 former students living in China which is the largest group outside the UK; and was selected for a tour by China's President Xi Jinping during his state visit to the UK in 2015. The college takes major donations from China and describes itself as the UK's number one university collaborator with Chinese research institutions. The China communist/fascist government did not appear phased by the woeful predictions of Ferguson and Imperial when during the lockdown that Ferguson induced the college signed a five-year collaboration deal with China tech giant Huawei that will have Huawei's indoor 5G network equipment installed at the college's West London tech campus along with an 'AI cloud platform'. The deal includes Chinese sponsorship of Imperial's Venture Catalyst entrepreneurship competition. Imperial is an example of the enormous influence the Chinese government has within British and North American

universities and research centres – and further afield. Up to 200 academics from more than a dozen UK universities are being investigated on suspicion of 'unintentionally' helping the Chinese government build weapons of mass destruction by 'transferring world-leading research in advanced military technology such as aircraft, missile designs and cyberweapons'. Similar scandals have broken in the United States, but it's all a coincidence. Imperial College serves the agenda in many other ways including the promotion of every aspect of the United Nations Agenda 21/2030 (the Great Reset) and produced computer models to show that human-caused 'climate change' is happening when in the real world it isn't. Imperial College is driving the climate agenda as it drives the 'Covid' agenda (both Cult hoaxes) while Patrick Vallance, the UK government's Chief Scientific Adviser on 'Covid', was named Chief Scientific Adviser to the UN 'climate change' conference known as COP26 hosted by the government in Glasgow, Scotland. 'Covid' and 'climate' are fundamentally connected.

Professor Woeful

From Imperial's bosom came Neil Ferguson still advising government despite his previous disasters and it was announced early on that he and other key people like UK Chief Medical Adviser Chris Whitty had caught the 'virus' as the propaganda story was being sold. Somehow they managed to survive and we had Prime Minister Boris Johnson admitted to hospital with what was said to be a severe version of the 'virus' in this same period. His whole policy and demeanour changed when he returned to Downing Street. It's a small world with these government advisors – especially in their communal connections to Gates – and Ferguson had partnered with Whitty to write a paper called 'Infectious disease: Tough choices to reduce Ebola transmission' which involved another scare-story that didn't happen. Ferguson's 'models' predicted that up to150, 000 could die from 'mad cow disease', or BSE, and its version in sheep if it was transmitted to humans. BSE was not transmitted and instead triggered by an organophosphate pesticide used to treat a pest on

cows. Fewer than 200 deaths followed from the human form. Models by Ferguson and his fellow incompetents led to the unnecessary culling of millions of pigs, cattle and sheep in the foot and mouth outbreak in 2001 which destroyed the lives and livelihoods of farmers and their families who had often spent decades building their herds and flocks. Vast numbers of these animals did not have foot and mouth and had no contact with the infection. Another 'expert' behind the cull was Professor Roy Anderson, a computer modeller at Imperial College specialising in the epidemiology of *human*, not animal, disease. Anderson has served on the Bill and Melinda Gates Grand Challenges in Global Health advisory board and chairs another Gates-funded organisation. Gates is everywhere.

In a precursor to the 'Covid' script Ferguson backed closing schools 'for prolonged periods' over the swine flu 'pandemic' in 2009 and said it would affect a third of the world population if it continued to spread at the speed he claimed to be happening. His mates at Imperial College said much the same and a news report said: 'One of the authors, the epidemiologist and disease modeller Neil Ferguson, who sits on the World Health Organisation's emergency committee for the outbreak, said the virus had "full pandemic potential".' Professor Liam Donaldson, the Chris Whitty of his day as Chief Medical Officer, said the worst case could see 30 percent of the British people infected by swine flu with 65,000 dying. Ferguson and Donaldson were indeed proved correct when at the end of the year the number of deaths attributed to swine flu was 392. The term 'expert' is rather liberally applied unfortunately, not least to complete idiots. Swine flu 'projections' were great for GlaxoSmithKline (GSK) as millions rolled in for its Pandemrix influenza vaccine which led to brain damage with children most affected. The British government (taxpayers) paid out more than £60 million in compensation after GSK was given immunity from prosecution. Yet another 'Covid' déjà vu. Swine flu was supposed to have broken out in Mexico, but Dr Wolfgang Wodarg, a German doctor, former member of parliament and critic of the 'Covid' hoax, observed 'the spread of swine flu' in Mexico City at the time. He

said: 'What we experienced in Mexico City was a very mild flu which did not kill more than usual – which killed even fewer people than usual.' Hyping the fear against all the facts is not unique to 'Covid' and has happened many times before. Ferguson is reported to have over-estimated the projected death toll of bird flu (H5N1) by some three million-fold, but bird flu vaccine makers again made a killing from the scare. This is some of the background to the Neil Ferguson who produced the perfectly-timed computer models in early 2020 predicting that half a million people would die in Britain without draconian lockdown and 2.2 million in the United States. Politicians panicked, people panicked, and lockdowns of alleged short duration were instigated to 'flatten the curve' of cases gleaned from a test not testing for the 'virus'. I said at the time that the public could forget the 'short duration' bit. This was an agenda to destroy the livelihoods of the population and force them into mass control through dependency and there was going to be nothing 'short' about it. American researcher Daniel Horowitz described the consequences of the 'models' spewed out by Gates-funded Ferguson and Imperial College:

> What led our government and the governments of many other countries into panic was a single Imperial College of UK study, funded by global warming activists, that predicted 2.2 million deaths if we didn't lock down the country. In addition, the reported 8-9% death rate in Italy scared us into thinking there was some other mutation of this virus that they got, which might have come here.
>
> Together with the fact that we were finally testing and had the ability to actually report new cases, we thought we were headed for a death spiral. But again … we can't flatten a curve if we don't know when the curve started.

How about it *never* started?

Giving them what they want

An investigation by German news outlet *Welt Am Sonntag (World on Sunday)* revealed how in March, 2020, the German government gathered together 'leading scientists from several research institutes and universities' and 'together, they were to produce a [modelling]

paper that would serve as legitimization for further tough political measures'. The Cult agenda was justified by computer modelling not based on evidence or reality; it was specifically constructed to justify the Cult demand for lockdowns all over the world to destroy the independent livelihoods of the global population. All these modellers and everyone responsible for the 'Covid' hoax have a date with a trial like those in Nuremberg after World War Two when Nazis faced the consequences of their war crimes. These corrupt-beyond-belief 'modellers' wrote the paper according to government instructions and it said that that if lockdown measures were lifted then up to one million Germans would die from 'Covid-19' adding that some would die 'agonizingly at home, gasping for breath' unable to be treated by hospitals that couldn't cope. All lies. No matter – it gave the Cult all that it wanted. What did long-time government 'modeller' Neil Ferguson say? If the UK and the United States didn't lockdown half a million would die in Britain and 2.2 million Americans. Anyone see a theme here? 'Modellers' are such a crucial part of the lockdown strategy that we should look into their background and follow the money. Researcher Rosemary Frei produced an excellent article headlined 'The Modelling-paper Mafiosi'. She highlights a guy called John Edmunds, a British epidemiologist, and professor in the Faculty of Epidemiology and Population Health at the London School of Hygiene & Tropical Medicine. He studied at Imperial College. Edmunds is a member of government 'Covid' advisory bodies which have been dictating policy, the New and Emerging Respiratory Virus Threats Advisory Group (NERVTAG) and the Scientific Advisory Group for Emergencies (SAGE).

Ferguson, another member of NERVTAG and SAGE, led the way with the original 'virus' and Edmunds has followed in the 'variant' stage and especially the so-called UK or Kent variant known as the 'Variant of Concern' (VOC) B.1.1.7. He said in a co-written report for the Centre for Mathematical modelling of Infectious Diseases at the London School of Hygiene and Tropical Medicine, with input from the Centre's 'Covid-19' Working Group, that there was 'a realistic

possibility that VOC B.1.1.7 is associated with an increased risk of death compared to non-VOC viruses'. Fear, fear, fear, get the vaccine, fear, fear, fear, get the vaccine. Rosemary Frei reveals that almost all the paper's authors and members of the modelling centre's 'Covid-19' Working Group receive funding from the Bill and Melinda Gates Foundation and/or the associated Gates-funded Wellcome Trust. The paper was published by e-journal *Medr χiv* which only publishes papers not peer-reviewed and the journal was established by an organisation headed by Facebook's Mark Zuckerberg and his missus. What a small world it is. Frei discovered that Edmunds is on the Scientific Advisory Board of the Coalition for Epidemic Preparedness Innovations (CEPI) which was established by the Bill and Melinda Gates Foundation, Klaus Schwab's Davos World Economic Forum and Big Pharma giant Wellcome. CEPI was 'launched in Davos [in 2017] to develop vaccines to stop future epidemics', according to its website. 'Our mission is to accelerate the development of vaccines against emerging infectious diseases and enable equitable access to these vaccines for people during outbreaks.' What kind people they are. Rosemary Frei reveals that Public Health England (PHE) director Susan Hopkins is an author of her organisation's non-peer-reviewed reports on 'new variants'. Hopkins is a professor of infectious diseases at London's Imperial College which is gifted tens of millions of dollars a year by the Bill and Melinda Gates Foundation. Gates-funded modelling disaster Neil Ferguson also co-authors Public Health England reports and he spoke in December, 2020, about the potential danger of the B.1.1.7. 'UK variant' promoted by Gates-funded modeller John Edmunds. When I come to the 'Covid vaccines' the 'new variants' will be shown for what they are – bollocks.

Connections, connections

All these people and modellers are lockdown-obsessed or, put another way, they demand what the Cult demands. Edmunds said in January, 2021, that to ease lockdowns too soon would be a disaster and they had to 'vaccinate much, much, much more widely than the

elderly'. Rosemary Frei highlights that Edmunds is married to Jeanne Pimenta who is described in a LinkedIn profile as director of epidemiology at GlaxoSmithKline (GSK) and she held shares in the company. Patrick Vallance, co-chair of SAGE and the government's Chief Scientific Adviser, is a former executive of GSK and has a deferred bonus of shares in the company worth £600,000. GSK has serious business connections with Bill Gates and is collaborating with mRNA-'vaccine' company CureVac to make 'vaccines' for the new variants that Edmunds is talking about. GSK is planning a 'Covid vaccine' with drug giant Sanofi. Puppet Prime Minister Boris Johnson announced in the spring of 2021 that up to 60 million vaccine doses were to be made at the GSK facility at Barnard Castle in the English North East. Barnard Castle, with a population of just 6,000, was famously visited in breach of lockdown rules in April, 2020, by Johnson aide Dominic Cummings who said that he drove there 'to test his eyesight' before driving back to London. Cummings would be better advised to test his integrity – not that it would take long. The GSK facility had nothing to do with his visit then although I'm sure Patrick Vallance would have been happy to arrange an introduction and some tea and biscuits. Ruthless psychopath Gates has made yet another fortune from vaccines in collaboration with Big Pharma companies and gushes at the phenomenal profits to be made from vaccines – more than a 20-to-1 return as he told one interviewer. Gates also tweeted in December, 2019, with the foreknowledge of what was coming: 'What's next for our foundation? I'm particularly excited about what the next year could mean for one of the best buys in global health: vaccines.'

Modeller John Edmunds is a big promotor of vaccines as all these people appear to be. He's the dean of the London School of Hygiene & Tropical Medicine's Faculty of Epidemiology and Population Health which is primarily funded by the Bill and Melinda Gates Foundation and the Gates-established and funded GAVI vaccine alliance which is the Gates vehicle to vaccinate the world. The organisation Doctors Without Borders has described GAVI as being 'aimed more at supporting drug-industry desires to promote new

products than at finding the most efficient and sustainable means for fighting the diseases of poverty'. But then that's why the psychopath Gates created it. John Edmunds said in a video that the London School of Hygiene & Tropical Medicine is involved in every aspect of vaccine development including large-scale clinical trials. He contends that mathematical modelling can show that vaccines protect individuals and society. That's on the basis of shit in and shit out, I take it. Edmunds serves on the UK Vaccine Network as does Ferguson and the government's foremost 'Covid' adviser, the grim-faced, dark-eyed Chris Whitty. The Vaccine Network says it works 'to support the government to identify and shortlist targeted investment opportunities for the most promising vaccines and vaccine technologies that will help combat infectious diseases with epidemic potential, and to address structural issues related to the UK's broader vaccine infrastructure'. Ferguson is acting Director of the Imperial College Vaccine Impact Modelling Consortium which has funding from the Bill and Melina Gates Foundation and the Gates-created GAVI 'vaccine alliance'. Anyone wonder why these characters see vaccines as the answer to every problem? Ferguson is wildly enthusiastic in his support for GAVI's campaign to vaccine children en masse in poor countries. You would expect someone like Gates who has constantly talked about the need to reduce the population to want to fund vaccines to keep more people alive. I'm sure that's why he does it. The John Edmunds London School of Hygiene & Tropical Medicine (LSHTM) has a Vaccines Manufacturing Innovation Centre which develops, tests and commercialises vaccines. Rosemary Frei writes:

> The vaccines centre also performs affiliated activities like combating 'vaccine hesitancy'. The latter includes the Vaccine Confidence Project. The project's stated purpose is, among other things, 'to provide analysis and guidance for early response and engagement with the public to ensure sustained confidence in vaccines and immunisation'. The Vaccine Confidence Project's director is LSHTM professor Heidi Larson. For more than a decade she's been researching how to combat vaccine hesitancy.

How the bloody hell can blokes like John Edmunds and Neil Ferguson with those connections and financial ties model 'virus' case

and death projections for the government and especially in a way that gives their paymasters like Gates exactly what they want? It's insane, but this is what you find throughout the world.

'Covid' is not dangerous, oops, wait, yes it is

Only days before Ferguson's nightmare scenario made Jackboot Johnson take Britain into a China-style lockdown to save us from a deadly 'virus' the UK government website gov.uk was reporting something very different to Ferguson on a page of official government guidance for 'high consequence infectious diseases (HCID)'. It said this about 'Covid-19':

> As of 19 March 2020, COVID-19 *is no longer considered to be a high consequence infectious diseases (HCID) in the UK* [my emphasis]. The 4 nations public health HCID group made an interim recommendation in January 2020 to classify COVID-19 as an HCID. This was based on consideration of the UK HCID criteria about the virus and the disease with information available during the early stages of the outbreak.
>
> Now that more is known about COVID-19, the public health bodies in the UK have reviewed the most up to date information about COVID-19 against the UK HCID criteria. They have determined that several features have now changed; in particular, more information is available about mortality rates (low overall), and there is now greater clinical awareness and a specific and sensitive laboratory test, the availability of which continues to increase. The Advisory Committee on Dangerous Pathogens (ACDP) is also of the opinion that COVID-19 should no longer be classified as an HCID.

Soon after the government had been exposed for downgrading the risk they upgraded it again and everyone was back to singing from the same Cult hymn book. Ferguson and his fellow Gates clones indicated that lockdowns and restrictions would have to continue until a Gates-funded vaccine was developed. Gates said the same because Ferguson and his like were repeating the Gates script which is the Cult script. 'Flatten the curve' became an ongoing nightmare of continuing lockdowns with periods in between of severe restrictions in pursuit of destroying independent incomes and had nothing to do with protecting health about which the Cult gives not a shit. Why wouldn't Ferguson be pushing a vaccine 'solution' when he's owned by vaccine-obsessive Gates who makes a fortune from them and

when Ferguson heads the Vaccine Impact Modelling Consortium at Imperial College funded by the Gates Foundation and GAVI, the 'vaccine alliance', created by Gates as his personal vaccine promotion operation? To compound the human catastrophe that Ferguson's 'models' did so much to create he was later exposed for breaking his own lockdown rules by having sexual liaisons with his married girlfriend Antonia Staats at his home while she was living at another location with her husband and children. Staats was a 'climate' activist and senior campaigner at the Soros-funded Avaaz which I wouldn't trust to tell me that grass is green. Ferguson had to resign as a government advisor over this hypocrisy in May, 2020, but after a period of quiet he was back being quoted by the ridiculous media on the need for more lockdowns and a vaccine rollout. Other government-advising 'scientists' from Imperial College' held the fort in his absence and said lockdown could be indefinite until a vaccine was found. The Cult script was being sung by the payrolled choir. I said there was no intention of going back to 'normal' when the 'vaccine' came because the 'vaccine' is part of a very different agenda that I will discuss in Human 2.0. Why would the Cult want to let the world go back to normal when destroying that normal forever was the whole point of what was happening? House arrest, closing businesses and schools through lockdown, (un)social distancing and masks all followed the Ferguson fantasy models. Again as I predicted (these people are so predictable) when the 'vaccine' arrived we were told that house arrest, lockdown, (un)social distancing and masks would still have to continue. I will deal with the masks in the next chapter because they are of fundamental importance.

Where's the 'pandemic'?

Any mildly in-depth assessment of the figures revealed what was really going on. Cult-funded and controlled organisations still have genuine people working within them such is the number involved. So it is with Genevieve Briand, assistant program director of the Applied Economics master's degree program at Johns Hopkins

University. She analysed the impact that 'Covid-19' had on deaths from *all* causes in the United States using official data from the CDC for the period from early February to early September, 2020. She found that allegedly 'Covid' *related*-deaths exceeded those from heart disease which she found strange with heart disease always the biggest cause of fatalities. Her research became even more significant when she noted the sudden decline in 2020 of *all* non-'Covid' deaths: 'This trend is completely contrary to the pattern observed in all previous years … the total decrease in deaths by other causes almost exactly equals the increase in deaths by Covid-19.' This was such a game, set and match in terms of what was happening that Johns Hopkins University deleted the article on the grounds that it 'was being used to support false and dangerous inaccuracies about the impact of the pandemic'. No – because it exposed the scam from official CDC figures and this was confirmed when those figures were published in January, 2021. Here we can see the effect of people dying from heart attacks, cancer, road accidents and gunshot wounds – *anything* – having 'Covid-19' on the death certificate along with those diagnosed from 'symptoms' who had even not tested positive with a test not testing for the 'virus'. I am not kidding with the gunshot wounds, by the way. Brenda Bock, coroner in Grand County, Colorado, revealed that two gunshot victims tested positive for the 'virus' within the previous 30 days and were therefore classified as 'Covid deaths'. Bock said: 'These two people had tested positive for Covid, but that's not what killed them. A gunshot wound is what killed them.' She said she had not even finished her investigation when the state listed the gunshot victims as deaths due to the 'virus'. The death and case figures for 'Covid-19' are an absolute joke and yet they are repeated like parrots by the media, politicians and alleged medical 'experts'. The official Cult narrative is the only show in town.

Genevieve Briand found that deaths from all causes were not exceptional in 2020 compared with previous years and a Spanish magazine published figures that said the same about Spain which was a 'Covid' propaganda hotspot at one point. *Discovery Salud*, a

health and medicine magazine, quoted government figures which showed how 17,000 *fewer* people died in Spain in 2020 than in 2019 and more than 26,000 fewer than in 2018. The age-standardised mortality rate for England and Wales when age distribution is taken into account was significantly lower in 2020 than the 1970s, 80s and 90s, and was only the ninth highest since 2000. Where is the 'pandemic'?

Post mortems and autopsies virtually disappeared for 'Covid' deaths amid claims that 'virus-infected' bodily fluids posed a risk to those carrying out the autopsy. This was rejected by renowned German pathologist and forensic doctor Klaus Püschel who said that he and his staff had by then done 150 autopsies on 'Covid' patients with no problems at all. He said they were needed to know why some 'Covid' patients suffered blood clots and not severe respiratory infections. The 'virus' is, after all, called SARS or 'severe acute respiratory syndrome'. I highlighted in the spring of 2020 this phenomenon and quoted New York intensive care doctor Cameron Kyle-Sidell who posted a soon deleted YouTube video to say that they had been told to prepare to treat an infectious disease called 'Covid-19', but that was not what they were dealing with. Instead he likened the lung condition of the most severely ill patients to what you would expect with cabin depressurisation in a plane at 30,000 feet or someone dropped on the top of Everest without oxygen or acclimatisation. I have never said this is not happening to a small minority of alleged 'Covid' patients – I am saying this is not caused by a phantom 'contagious virus'. Indeed Kyle-Sidell said that 'Covid-19' was not the disease they were told was coming their way. 'We are operating under a medical paradigm that is untrue,' he said, and he believed they were treating the wrong disease: 'These people are being slowly starved of oxygen.' Patients would take off their oxygen masks in a state of fear and stress and while they were blue in the face on the brink of death. They did not look like patients dying of pneumonia. You can see why they don't want autopsies when their virus doesn't exist and there is another condition in some people that they don't wish to be uncovered. I should add here that

the 5G system of millimetre waves was being rapidly introduced around the world in 2020 and even more so now as they fire 5G at the Earth from satellites. At 60 gigahertz within the 5G range that frequency interacts with the oxygen molecule and stops people breathing in sufficient oxygen to be absorbed into the bloodstream. They are installing 5G in schools and hospitals. The world is not mad or anything. 5G can cause major changes to the lungs and blood as I detail in *The Answer* and these consequences are labelled 'Covid-19', the alleged symptoms of which can be caused by 5G and other electromagnetic frequencies as cells respond to radiation poisoning.

The 'Covid death' scam

Dr Scott Jensen, a Minnesota state senator and medical doctor, exposed 'Covid' Medicare payment incentives to hospitals and death certificate manipulation. He said he was sent a seven-page document by the US Department of Health 'coaching' him on how to fill out death certificates which had never happened before. The document said that he didn't need to have a laboratory test for 'Covid-19' to put that on the death certificate and that shocked him when death certificates are supposed to be about facts. Jensen described how doctors had been 'encouraged, if not pressured' to make a diagnosis of 'Covid-19' if they thought it was probable or *'presumed*'. No positive test was necessary – not that this would have mattered anyway. He said doctors were told to diagnose 'Covid' by symptoms when these were the same as colds, allergies, other respiratory problems, and certainly with influenza which 'disappeared' in the 'Covid' era. A common sniffle was enough to get the dreaded verdict. Ontario authorities decreed that a single care home resident with *one* symptom from a long list must lead to the isolation of the entire home. Other courageous doctors like Jensen made the same point about death figure manipulation and how deaths by other causes were falling while 'Covid-19 deaths' were rising at the same rate due to re-diagnosis. Their videos rarely survive long on YouTube with its Cult-supporting algorithms courtesy of CEO Susan Wojcicki and her bosses at Google. Figure-tampering was so glaring

and ubiquitous that even officials were letting it slip or outright saying it. UK chief scientific adviser Patrick Vallance said on one occasion that 'Covid' on the death certificate doesn't mean 'Covid' was the cause of death (so why the hell is it there?) and we had the rare sight of a BBC reporter telling the truth when she said: 'Someone could be successfully treated for Covid, in say April, discharged, and then in June, get run over by a bus and die … That person would still be counted as a Covid death in England.' Yet the BBC and the rest of the world media went on repeating the case and death figures as if they were real. Illinois Public Health Director Dr Ngozi Ezike revealed the deceit while her bosses must have been clenching their buttocks:

> If you were in a hospice and given a few weeks to live and you were then found to have Covid that would be counted as a Covid death. [There might be] a clear alternate cause, but it is still listed as a Covid death. So everyone listed as a Covid death doesn't mean that was the cause of the death, but that they had Covid at the time of death.

Yes, a 'Covid virus' never shown to exist and tested for with a test not testing for the 'virus'. In the first period of the pandemic hoax through the spring of 2020 the process began of designating almost everything a 'Covid' death and this has continued ever since. I sat in a restaurant one night listening to a loud conversation on the next table where a family was discussing in bewilderment how a relative who had no symptoms of 'Covid', and had died of a long-term problem, could have been diagnosed a death by the 'virus'. I could understand their bewilderment. If they read this book they will know why this medical fraud has been perpetrated the world over.

Some media truth shock

The media ignored the evidence of death certificate fraud until eventually one columnist did speak out when she saw it first-hand. Bel Mooney is a long-time national newspaper journalist in Britain currently working for the *Daily Mail*. Her article on February 19th, 2021, carried this headline: 'My dad Ted passed three Covid tests

and died of a chronic illness yet he's officially one of Britain's 120,000 victims of the virus and is far from alone ... so how many more are there?' She told how her 99-year-old father was in a care home with a long-standing chronic obstructive pulmonary disease and vascular dementia. Maybe, but he was still aware enough to tell her from the start that there was no 'virus' and he refused the 'vaccine' for that reason. His death was not unexpected given his chronic health problems and Mooney said she was shocked to find that 'Covid-19' was declared the cause of death on his death certificate. She said this was a 'bizarre and unacceptable untruth' for a man with long-time health problems who had tested negative twice at the home for the 'virus'. I was also shocked by this story although not by what she said. I had been highlighting the death certificate manipulation for ten months. It was the confirmation that a professional full-time journalist only realised this was going on when it affected her directly and neither did she know that whether her dad tested positive or negative was irrelevant with the test not testing for the 'virus'. Where had she been? She said she did not believe in 'conspiracy theories' without knowing I'm sure that this and 'conspiracy theorists' were terms put into widespread circulation by the CIA in the 1960s to discredit those who did not accept the ridiculous official story of the Kennedy assassination. A blanket statement of 'I don't believe in conspiracy theories' is always bizarre. The dictionary definition of the term alone means the world is drowning in conspiracies. What she said was even more daft when her dad had just been affected by the 'Covid' conspiracy. Why else does she think that 'Covid-19' was going on the death certificates of people who died of something else?

To be fair once she saw from personal experience what was happening she didn't mince words. Mooney was called by the care home on the morning of February 9th to be told her father had died in his sleep. When she asked for the official cause of death what came back was 'Covid-19'. Mooney challenged this and was told there had been deaths from Covid on the dementia floor (confirmed by a test not testing for the 'virus') so they considered it 'reasonable

to assume'. 'But doctor,' Mooney rightly protested, 'an assumption isn't a diagnosis.' She said she didn't blame the perfectly decent and sympathetic doctor – 'he was just doing his job'. Sorry, but that's *bullshit*. He wasn't doing his job at all. He was putting a false cause of death on the death certificate and that is a criminal offence for which he should be brought to account and the same with the millions of doctors worldwide who have done the same. They were not doing their job they were following orders and that must not wash at new Nuremberg trials any more than it did at the first ones. Mooney's doctor was 'assuming' (presuming) as he was told to, but 'just following orders' makes no difference to his actions. A doctor's job is to serve the patient and the truth, not follow orders, but that's what they have done all over the world and played a central part in making the 'Covid' hoax possible with all its catastrophic consequences for humanity. Shame on them and they must answer for their actions. Mooney said her disquiet worsened when she registered her father's death by telephone and was told by the registrar there had been very many other cases like hers where 'the deceased' had not tested positive for 'Covid' yet it was recorded as the cause of death. The test may not matter, but those involved at their level *think* it matters and it shows a callous disregard for accurate diagnosis. The pressure to do this is coming from the top of the national 'health' pyramids which in turn obey the World Health Organization which obeys Gates and the Cult. Mooney said the registrar agreed that this must distort the national figures adding that 'the strangest thing is that every winter we record countless deaths from flu, and this winter there have been none. Not one!' She asked if the registrar thought deaths from flu were being misdiagnosed and lumped together with 'Covid' deaths. The answer was a 'puzzled yes'. Mooney said that the funeral director said the same about 'Covid' deaths which had nothing to do with 'Covid'. They had lost count of the number of families upset by this and other funeral companies in different countries have had the same experience. Mooney wrote:

The nightly shroud-waving and shocking close-ups of pain imposed on us by the TV news bewildered and terrified the population into eager compliance with lockdowns. We were invited to 'save the NHS' and to grieve for strangers – the real-life loved ones behind those shocking death counts. Why would the public imagine what I now fear, namely that the way Covid-19 death statistics are compiled might make the numbers seem greater than they are?

Oh, just a little bit – like 100 percent.

Do the maths

Mooney asked why a country would wish to skew its mortality figures by wrongly certifying deaths? What had been going on? Well, if you don't believe in conspiracies you will never find the answer which is that *it's a conspiracy*. She did, however, describe what she had discovered as a 'national scandal'. In reality it's a global scandal and happening everywhere. Pillars of this conspiracy were all put into place before the button was pressed with the Drosten PCR protocol and high amplifications to produce the cases and death certificate changes to secure illusory 'Covid' deaths. Mooney notes that normally two doctors were needed to certify a death, with one having to know the patient, and how the rules were changed in the spring of 2020 to allow one doctor to do this. In the same period 'Covid deaths' were decreed to be all cases where Covid-19 was put on the death certificate even without a positive test or any symptoms. Mooney asked: 'How many of the 30,851 (as of January 15) care home resident deaths with Covid-19 on the certificate (32.4 per cent of all deaths so far) were based on an assumption, like that of my father? And what has that done to our national psyche?'All of them is the answer to the first question and it has devastated and dismantled the national psyche, actually the global psyche, on a colossal scale. In the UK case and death data is compiled by organisations like Public Health England (PHE) and the Office for National Statistics (ONS). Mooney highlights the insane policy of counting a death from any cause as 'Covid-19' if this happens within 28 days of a positive test (with a test not testing for the 'virus') and she points out that ONS statistics reflect deaths 'involving Covid' 'or due to Covid' which meant in practice any

death where 'Covid-19' was mentioned on the death certificate. She described the consequences of this fraud:

> Most people will accept the narrative they are fed, so panicky governments here and in Europe witnessed the harsh measures enacted in totalitarian China and jumped into lockdown. Headlines about Covid deaths tolled like the knell that would bring doomsday to us all. Fear stalked our empty streets. Politicians parroted the frankly ridiculous aim of 'zero Covid' and shut down the economy, while most British people agreed that lockdown was essential and (astonishingly to me, as a patriotic Brit) even wanted more restrictions.
>
> For what? Lies on death certificates? Never mind the grim toll of lives ruined, suicides, schools closed, rising inequality, depression, cancelled hospital treatments, cancer patients in a torture of waiting, poverty, economic devastation, loneliness, families kept apart, and so on. How many lives have been lost as a direct result of lockdown?

She said that we could join in a national chorus of shock and horror at reaching the 120,000 death toll which was surely certain to have been totally skewed all along, but what about the human cost of lockdown justified by these 'death figures'? *The British Medical Journal* had reported a 1,493 percent increase in cases of children taken to Great Ormond Street Hospital with abusive head injuries alone and then there was the effect on families:

> Perhaps the most shocking thing about all this is that families have been kept apart – and obeyed the most irrational, changing rules at the whim of government – because they believed in the statistics. They succumbed to fear, which his generation rejected in that war fought for freedom. Dad (God rest his soul) would be angry. And so am I.

Another theme to watch is that in the winter months when there are more deaths from all causes they focus on 'Covid' deaths and in the summer when the British Lung Foundation says respiratory disease plummets by 80 percent they rage on about 'cases'. Either way fascism on population is always the answer.

Nazi eugenics in the 21st century

Elderly people in care homes have been isolated from their families month after lonely month with no contact with relatives and grandchildren who were banned from seeing them. We were told

that lockdown fascism was to 'protect the vulnerable' like elderly people. At the same time Do Not Resuscitate (DNR) orders were placed on their medical files so that if they needed resuscitation it wasn't done and 'Covid-19' went on their death certificates. Old people were not being 'protected' they were being culled – murdered in truth. DNR orders were being decreed for disabled and young people with learning difficulties or psychological problems. The UK Care Quality Commission, a non-departmental body of the Department of Health and Social Care, found that 34 percent of those working in health and social care were pressured into placing 'do not attempt cardiopulmonary resuscitation' orders on 'Covid' patients who suffered from disabilities and learning difficulties without involving the patient or their families in the decision. UK judges ruled that an elderly woman with dementia should have the DNA-manipulating 'Covid vaccine' against her son's wishes and that a man with severe learning difficulties should have the jab despite his family's objections. Never mind that many had already died. The judiciary always supports doctors and government in fascist dictatorships. They wouldn't dare do otherwise. A horrific video was posted showing fascist officers from Los Angeles police forcibly giving the 'Covid' shot to women with special needs who were screaming that they didn't want it. The same fascists are seen giving the jab to a sleeping elderly woman in a care home. This is straight out of the Nazi playbook. Hitler's Nazis committed mass murder of the mentally ill and physically disabled throughout Germany and occupied territories in the programme that became known as Aktion T4, or just T4. Sabbatian-controlled Hitler and his grotesque crazies set out to kill those they considered useless and unnecessary. The Reich Committee for the Scientific Registering of Hereditary and Congenital Illnesses registered the births of babies identified by physicians to have 'defects'. By 1941 alone more than 5,000 children were murdered by the state and it is estimated that in total the number of innocent people killed in Aktion T4 was between 275,000 and 300,000. Parents were told their children had been sent away for 'special treatment' never to return. It is rather pathetic to see claims about plans for new extermination camps being dismissed today

when the same force behind current events did precisely that 80 years ago. Margaret Sanger was a Cult operative who used 'birth control' to sanitise her programme of eugenics. Organisations she founded became what is now Planned Parenthood. Sanger proposed that 'the whole dysgenic population would have its choice of segregation or sterilization'. These included epileptics, 'feeble-minded', and prostitutes. Sanger opposed charity because it perpetuated 'human waste'. She reveals the Cult mentality and if anyone thinks that extermination camps are a 'conspiracy theory' their naivety is touching if breathtakingly stupid.

If you don't believe that doctors can act with callous disregard for their patients it is worth considering that doctors and medical staff agreed to put government-decreed DNR orders on medical files and do nothing when resuscitation is called for. I don't know what you call such people in your house. In mine they are Nazis from the Josef Mengele School of Medicine. Phenomenal numbers of old people have died worldwide from the effects of lockdown, depression, lack of treatment, the 'vaccine' (more later) and losing the will to live. A common response at the start of the manufactured pandemic was to remove old people from hospital beds and transfer them to nursing homes. The decision would result in a mass cull of elderly people in those homes through lack of treatment – *not* 'Covid'. Care home whistleblowers have told how once the 'Covid' era began doctors would not come to their homes to treat patients and they were begging for drugs like antibiotics that often never came. The most infamous example was ordered by New York governor Andrew Cuomo, brother of a moronic CNN host, who amazingly was given an Emmy Award for his handling of the 'Covid crisis' by the ridiculous Wokers that hand them out. Just how ridiculous could be seen in February, 2021, when a Department of Justice and FBI investigation began into how thousands of old people in New York died in nursing homes after being discharged from hospital to make way for 'Covid' patients on Cuomo's say-so – and how he and his staff covered up these facts. This couldn't have happened to a nicer psychopath. Even then there was a 'Covid' spin. Reports said that

thousands of old people who tested positive for 'Covid' in hospital were transferred to nursing homes to both die of 'Covid' and transmit it to others. No – they were in hospital because they were ill and the fact that they tested positive with a test not testing for the 'virus' is irrelevant. They were ill often with respiratory diseases ubiquitous in old people near the end of their lives. Their transfer out of hospital meant that their treatment stopped and many would go on to die.

They're old. Who gives a damn?

I have exposed in the books for decades the Cult plan to cull the world's old people and even to introduce at some point what they call a 'demise pill' which at a certain age everyone would take and be out of here by law. In March, 2021, Spain legalised euthanasia and assisted suicide following the Netherlands, Belgium, Luxembourg and Canada on the Tiptoe to the demise pill. Treatment of old people by many 'care' homes has been a disgrace in the 'Covid' era. There are many, many, caring staff – I know some. There have, however, been legions of stories about callous treatment of old people and their families. Police were called when families came to take their loved ones home in the light of isolation that was killing them. They became prisoners of the state. Care home residents in insane, fascist Ontario, Canada, were not allowed to leave their *room* once the 'Covid' hoax began. UK staff have even wheeled elderly people away from windows where family members were talking with them. Oriana Criscuolo from Stockport in the English North West dropped off some things for her 80-year-old father who has Parkinson's disease and dementia and she wanted to wave to him through a ground-floor window. She was told that was 'illegal'. When she went anyway they closed the curtains in the middle of the day. Oriana said:

> It's just unbelievable. I cannot understand how care home staff – people who are being paid to care – have become so uncaring. Their behaviour is inhumane and cruel. It's beyond belief.

She was right and this was not a one-off. What a way to end your life in such loveless circumstances. UK registered nurse Nicky Millen, a proper old school nurse for 40 years, said that when she started her career care was based on dignity, choice, compassion and empathy. Now she said 'the things that are important to me have gone out of the window.' She was appalled that people were dying without their loved ones and saying goodbye on iPads. Nicky described how a distressed 89-year-old lady stroked her face and asked her 'how many paracetamol would it take to finish me off'. Life was no longer worth living while not seeing her family. Nicky said she was humiliated in front of the ward staff and patients for letting the lady stroke her face and giving her a cuddle. Such is the dehumanisation that the 'Covid' hoax has brought to the surface. Nicky worked in care homes where patients told her they were being held prisoner. 'I want to live until I die', one said to her. 'I had a lady in tears because she hadn't seen her great-grandson.' Nicky was compassionate old school meeting psychopathic New Normal. She also said she had worked on a 'Covid' ward with no 'Covid' patients. Jewish writer Shai Held wrote an article in March, 2020, which was headlined 'The Staggering, Heartless Cruelty Toward the Elderly'. What he described was happening from the earliest days of lockdown. He said 'the elderly' were considered a group and not unique individuals (the way of the Woke). Shai Held said:

> Notice how the all-too-familiar rhetoric of dehumanization works: 'The elderly' are bunched together as a faceless mass, all of them considered culprits and thus effectively deserving of the suffering the pandemic will inflict upon them. Lost entirely is the fact that the elderly are individual human beings, each with a distinctive face and voice, each with hopes and dreams, memories and regrets, friendships and marriages, loves lost and loves sustained.

'The elderly' have become another dehumanised group for which anything goes and for many that has resulted in cold disregard for their rights and their life. The distinctive face that Held talks about is designed to be deleted by masks until everyone is part of a faceless mass.

'War-zone' hospitals myth

Again and again medical professionals have told me what was really going on and how hospitals 'overrun like war zones' according to the media were virtually empty. The mantra from medical whistleblowers was please don't use my name or my career is over. Citizen journalists around the world sneaked into hospitals to film evidence exposing the 'war-zone' lie. They really *were* largely empty with closed wards and operating theatres. I met a hospital worker in my town on the Isle of Wight during the first lockdown in 2020 who said the only island hospital had never been so quiet. Lockdown was justified by the psychopaths to stop hospitals being overrun. At the same time that the island hospital was near-empty the military arrived here to provide *extra beds*. It was all propaganda to ramp up the fear to ensure compliance with fascism as were never-used temporary hospitals with thousands of beds known as Nightingales and never-used make-shift mortuaries opened by the criminal UK government. A man who helped to install those extra island beds attributed to the army said they were never used and the hospital was empty. Doctors and nurses 'stood around talking or on their phones, wandering down to us to see what we were doing'. There were no masks or social distancing. He accused the useless local island paper, the *County Press*, of 'pumping the fear as if our hospital was overrun and we only have one so it should have been'. He described ambulances parked up with crews outside in deck chairs. When his brother called an ambulance he was told there was a two-hour backlog which he called 'bullshit'. An old lady on the island fell 'and was in a bad way', but a caller who rang for an ambulance was told the situation wasn't urgent enough. Ambulance stations were working under capacity while people would hear ambulances with sirens blaring driving through the streets. When those living near the stations realised what was going on they would follow them as they left, circulated around an urban area with the sirens going, and then came back without stopping. All this was to increase levels of fear and the same goes for the 'ventilator shortage crisis' that cost tens of millions for hastily produced ventilators never to be used.

Ambulance crews that agreed to be exploited in this way for fear propaganda might find themselves a mirror. I wish them well with that. Empty hospitals were the obvious consequence of treatment and diagnoses of non-'Covid' conditions cancelled and those involved handed a death sentence. People have been dying at home from undiagnosed and untreated cancer, heart disease and other life-threatening conditions to allow empty hospitals to deal with a 'pandemic' that wasn't happening.

Death of the innocent

'War-zones' have been laying off nursing staff, even doctors where they can. There was no work for them. Lockdown was justified by saving lives and protecting the vulnerable they were actually killing with DNR orders and preventing empty hospitals being 'overrun'. In Britain the mantra of stay at home to 'save the NHS' was everywhere and across the world the same story was being sold when it was all lies. Two California doctors, Dan Erickson and Artin Massihi at Accelerated Urgent Care in Bakersfield, held a news conference in April, 2020, to say that intensive care units in California were 'empty, essentially', with hospitals shutting floors, not treating patients and laying off doctors. The California health system was working at minimum capacity 'getting rid of doctors because we just don't have the volume'. They said that people with conditions such as heart disease and cancer were not coming to hospital out of fear of 'Covid-19'. Their video was deleted by Susan Wojcicki's Cult-owned YouTube after reaching five million views. Florida governor Ron Desantis, who rejected the severe lockdowns of other states and is being targeted for doing so, said that in March, 2020, every US governor was given models claiming they would run out of hospital beds in days. That was never going to happen and the 'modellers' knew it. Deceit can be found at every level of the system. Urgent children's operations were cancelled including fracture repairs and biopsies to spot cancer. Eric Nicholls, a consultant paediatrician, said 'this is obviously concerning and we need to return to normal operating and to increase capacity as soon as possible'. Psychopaths

in power were rather less concerned *because* they are psychopaths. Deletion of urgent care and diagnosis has been happening all over the world and how many kids and others have died as a result of the actions of these cold and heartless lunatics dictating 'health' policy? The number must be stratospheric. Richard Sullivan, professor of cancer and global health at King's College London, said people feared 'Covid' more than cancer such was the campaign of fear. 'Years of lost life will be quite dramatic', Sullivan said, with 'a huge amount of avoidable mortality'. Sarah Woolnough, executive director for policy at Cancer Research UK, said there had been a 75 percent drop in urgent referrals to hospitals by family doctors of people with suspected cancer. Sullivan said that 'a lot of services have had to scale back – we've seen a dramatic decrease in the amount of elective cancer surgery'. Lockdown deaths worldwide has been absolutely fantastic with the *New York Post* reporting how data confirmed that 'lockdowns end more lives than they save':

> There was a sharp decline in visits to emergency rooms and an increase in fatal heart attacks because patients didn't receive prompt treatment. Many fewer people were screened for cancer. Social isolation contributed to excess deaths from dementia and Alzheimer's.
>
> Researchers predicted that the social and economic upheaval would lead to tens of thousands of "deaths of despair" from drug overdoses, alcoholism and suicide. As unemployment surged and mental-health and substance-abuse treatment programs were interrupted, the reported levels of anxiety, depression and suicidal thoughts increased dramatically, as did alcohol sales and fatal drug overdoses.

This has been happening while nurses and other staff had so much time on their hands in the 'war-zones' that Tic-Tok dancing videos began appearing across the Internet with medical staff dancing around in empty wards and corridors as people died at home from causes that would normally have been treated in hospital.

Mentions in dispatches

One brave and truth-committed whistleblower was Louise Hampton, a call handler with the UK NHS who made a viral Internet video saying she had done 'fuck all' during the 'pandemic'

which was 'a load of bollocks'. She said that 'Covid-19' was rebranded flu and of course she lost her job. This is what happens in the medical and endless other professions now when you tell the truth. Louise filmed inside 'war-zone' accident and emergency departments to show they were empty and I mean *empty* as in no one there. The mainstream media could have done the same and blown the gaff on the whole conspiracy. They haven't to their eternal shame. Not that most 'journalists' seem capable of manifesting shame as with the psychopaths they slavishly repeat without question. The relative few who were admitted with serious health problems were left to die alone with no loved ones allowed to see them because of 'Covid' rules and they included kids dying without the comfort of mum and dad at their bedside while the evil behind this couldn't give a damn. It was all good fun to them. A Scottish NHS staff nurse publicly quit in the spring of 2021 saying: 'I can no longer be part of the lies and the corruption by the government.' She said hospitals 'aren't full, the beds aren't full, beds have been shut, wards have been shut'. Hospitals were never busy throughout 'Covid'. The staff nurse said that Nicola Sturgeon, tragically the leader of the Scottish government, was on television saying save the hospitals and the NHS – 'but the beds are empty' and 'we've not seen flu, we always see flu every year'. She wrote to government and spoke with her union Unison (the unions are Cult-compromised and *useless*, but nothing changed. Many of her colleagues were scared of losing their jobs if they spoke out as they wanted to. She said nursing staff were being affected by wearing masks all day and 'my head is splitting every shift from wearing a mask'. The NHS is part of the fascist tyranny and must be dismantled so we can start again with human beings in charge. (Ironically, hospitals were reported to be busier again when official 'Covid' cases *fell* in spring/summer of 2021 and many other conditions required treatment at the same time as *the fake vaccine rollout*.)

I will cover the 'Covid vaccine' scam in detail later, but it is another indicator of the sickening disregard for human life that I am highlighting here. The DNA-manipulating concoctions do not fulfil

the definition of a 'vaccine', have never been used on humans before and were given only emergency approval because trials were not completed and they continued using the unknowing public. The result was what a NHS senior nurse with responsibility for 'vaccine' procedure said was 'genocide'. She said the 'vaccines' were not 'vaccines'. They had not been shown to be safe and claims about their effectiveness by drug companies were 'poetic licence'. She described what was happening as a 'horrid act of human annihilation'. The nurse said that management had instigated a policy of not providing a Patient Information Leaflet (PIL) before people were 'vaccinated' even though health care professionals are supposed to do this according to protocol. Patients should also be told that they are taking part in an ongoing clinical trial. Her challenges to what is happening had seen her excluded from meetings and ridiculed in others. She said she was told to 'watch my step … or I would find myself surplus to requirements'. The nurse, who spoke anonymously in fear of her career, said she asked her NHS manager why he/she was content with taking part in genocide against those having the 'vaccines'. The reply was that everyone had to play their part and to 'put up, shut up, and get it done'. Government was 'leaning heavily' on NHS management which was clearly leaning heavily on staff. This is how the global 'medical' hierarchy operates and it starts with the Cult and its World Health Organization.

She told the story of a doctor who had the Pfizer jab and when questioned had no idea what was in it. The doctor had never read the literature. We have to stop treating doctors as intellectual giants when so many are moral and medical pygmies. The doctor did not even know that the 'vaccines' were not fully approved or that their trials were ongoing. They were, however, asking their patients if they minded taking part in follow-ups for research purposes – yes, the *ongoing clinical trial*. The nurse said the doctor's ignorance was not rare and she had spoken to a hospital consultant who had the jab without any idea of the background or that the 'trials' had not been completed. Nurses and pharmacists had shown the same ignorance.

'My NHS colleagues have forsaken their duty of care, broken their code of conduct – Hippocratic Oath – and have been brainwashed just the same as the majority of the UK public through propaganda …' She said she had not been able to recruit a single NHS colleague, doctor, nurse or pharmacist to stand with her and speak out. Her union had refused to help. She said that if the genocide came to light she would not hesitate to give evidence at a Nuremberg-type trial against those in power who could have affected the outcomes but didn't.

And all for what?

To put the nonsense into perspective let's say the 'virus' does exist and let's go completely crazy and accept that the official manipulated figures for cases and deaths are accurate. *Even then* a study by Stanford University epidemiologist Dr John Ioannidis published on the World Health Organization website produced an average infection to fatality rate of … *0.23 percent!* Ioannidis said: 'If one could sample equally from all locations globally, the median infection fatality rate might even be substantially lower than the 0.23% observed in my analysis.' For healthy people under 70 it was … *0.05 percent*! This compares with the 3.4 percent claimed by the Cult-owned World Health Organization when the hoax was first played and maximum fear needed to be generated. An updated Stanford study in April, 2021, put the 'infection' to 'fatality' rate at just 0.15 percent. Another team of scientists led by Megan O'Driscoll and Henrik Salje studied data from 45 countries and published their findings on the Nature website. For children and young people the figure is so small it virtually does not register although authorities will be hyping dangers to the young when they introduce DNA-manipulating 'vaccines' for children. The O'Driscoll study produced an average infection-fatality figure of 0.003 for children from birth to four; 0.001 for 5 to 14; 0.003 for 15 to 19; and it was still only 0.456 up to 64. To claim that children must be 'vaccinated' to protect them from 'Covid' is an obvious lie and so there must be another reason and there is. What's more the average age of a 'Covid' death is akin

to the average age that people die in general. The average age of death in England is about 80 for men and 83 for women. The average age of death from alleged 'Covid' is between 82 and 83. California doctors, Dan Erickson and Artin Massihi, said at their April media conference that projection models of millions of deaths had been 'woefully inaccurate'. They produced detailed figures showing that Californians had a 0.03 chance of dying from 'Covid' based on the number of people who tested positive (with a test not testing for the 'virus'). Erickson said there was a 0.1 percent chance of dying from 'Covid' in the *state* of New York, not just the city, and a 0.05 percent chance in Spain, a centre of 'Covid-19' hysteria at one stage. The Stanford studies supported the doctors' data with fatality rate estimates of 0.23 and 0.15 percent. How close are these figures to my estimate of *zero*? Death-rate figures claimed by the World Health Organization at the start of the hoax were some 15 times higher. The California doctors said there was no justification for lockdowns and the economic devastation they caused. Everything they had ever learned about quarantine was that you quarantine the *sick* and not the healthy. They had never seen this before and it made no medical sense.

Why in the in the light of all this would governments and medical systems the world over say that billions must go under house arrest; lose their livelihood; in many cases lose their mind, their health and their life; force people to wear masks dangerous to health and psychology; make human interaction and even family interaction a criminal offence; ban travel; close restaurants, bars, watching live sport, concerts, theatre, and any activity involving human togetherness and discourse; and closing schools to isolate children from their friends and cause many to commit suicide in acts of hopelessness and despair? The California doctors said lockdown consequences included increased child abuse, partner abuse, alcoholism, depression, and other impacts they were seeing every day. Who would do that to the entire human race if not mentally-ill psychopaths of almost unimaginable extremes like Bill Gates? We must face the reality of what we are dealing with and come out of

denial. Fascism and tyranny are made possible only by the target population submitting and acquiescing to fascism and tyranny. The whole of human history shows that to be true. Most people naively and unquestioning believed what they were told about a 'deadly virus' and meekly and weakly submitted to house arrest. Those who didn't believe it – at least in total – still submitted in fear of the consequences of not doing so. For the rest who wouldn't submit draconian fines have been imposed, brutal policing by psychopaths *for* psychopaths, and condemnation from the meek and weak who condemn the Pushbackers on behalf of the very force that has them, too, in its gunsights. 'Pathetic' does not even begin to suffice. Britain's brainless 'Health' Secretary Matt Hancock warned anyone lying to border officials about returning from a list of 'hotspot' countries could face a jail sentence of up to ten years which is more than for racially-aggravated assault, incest and attempting to have sex with a child under 13. Hancock is a lunatic, but he has the state apparatus behind him in a Cult-led chain reaction and the same with UK 'Vaccine Minister' Nadhim Zahawi, a prominent member of the mega-Cult secret society, Le Cercle, which featured in my earlier books. The Cult enforces its will on governments and medical systems; government and medical systems enforce their will on business and police; business enforces its will on staff who enforce it on customers; police enforce the will of the Cult on the population and play their essential part in creating a world of fascist control that their own children and grandchildren will have to live in their entire lives. It is a hierarchical pyramid of imposition and acquiescence and, yes indeedy, of clinical insanity.

Does anyone bright enough to read this book have to ask what the answer is? I think not, but I will reveal it anyway in the fewest of syllables: Tell the psychos and their moronic lackeys to fuck off and let's get on with our lives. We are many – They are few.

CHAPTER SEVEN

War on your mind

One believes things because one has been conditioned to believe them
Aldous Huxley, *Brave New World*

I have described the 'Covid' hoax as a 'Psyop' and that is true in every sense and on every level in accordance with the definition of that term which is psychological warfare. Break down the 'Covid pandemic' to the foundation themes and it is psychological warfare on the human individual and collective mind.

The same can be said for the entire human belief system involving every subject you can imagine. Huxley was right in his contention that people believe what they are conditioned to believe and this comes from the repetition throughout their lives of the same falsehoods. They spew from government, corporations, media and endless streams of 'experts' telling you what the Cult wants you to believe and often believing it themselves (although *far* from always). 'Experts' are rewarded with 'prestigious' jobs and titles and as agents of perceptual programming with regular access to the media. The Cult has to control the narrative – control *information* – or they lose control of the vital, crucial, without-which-they-cannot-prevail public perception of reality. The foundation of that control today is the Internet made possible by the Defense Advanced Research Projects Agency (DARPA), the incredibly sinister technological arm of the Pentagon. The Internet is the result of military technology.

DARPA openly brags about establishing the Internet which has been a long-term project to lasso the minds of the global population. I have said for decades the plan is to control information to such an extreme that eventually no one would see or hear anything that the Cult does not approve. We are closing in on that end with ferocious censorship since the 'Covid' hoax began and in my case it started back in the 1990s in terms of books and speaking venues. I had to create my own publishing company in 1995 precisely because no one else would publish my books even then. I think they're all still running.

Cult Internet

To secure total control of information they needed the Internet in which pre-programmed algorithms can seek out 'unclean' content for deletion and even stop it being posted in the first place. The Cult had to dismantle print and non-Internet broadcast media to ensure the transfer of information to the appropriate-named 'Web' – a critical expression of the *Cult* web. We've seen the ever-quickening demise of traditional media and control of what is left by a tiny number of corporations operating worldwide. Independent journalism in the mainstream is already dead and never was that more obvious than since the turn of 2020. The Cult wants all information communicated via the Internet to globally censor and allow the plug to be pulled any time. Lockdowns and forced isolation has meant that communication between people has been through electronic means and no longer through face-to-face discourse and discussion. Cult psychopaths have targeted the bars, restaurants, sport, venues and meeting places in general for this reason. None of this is by chance and it's to stop people gathering in any kind of privacy or number while being able to track and monitor all Internet communications and block them as necessary. Even private messages between individuals have been censored by these fascists that control Cult fronts like Facebook, Twitter, Google and YouTube which are all officially run by Sabbatian place-people and from the background by higher-level Sabbatian place people.

Facebook, Google, Amazon and their like were seed-funded and supported into existence with money-no-object infusions of funds either directly or indirectly from DARPA and CIA technology arm In-Q-Tel. The Cult plays the long game and prepares very carefully for big plays like 'Covid'. Amazon is another front in the psychological war and pretty much controls the global market in book sales and increasingly publishing. Amazon's limitless funds have deleted fantastic numbers of independent publishers to seize global domination on the way to deciding which books can be sold and circulated and which cannot. Moves in that direction are already happening. Amazon's leading light Jeff Bezos is the grandson of Lawrence Preston Gise who worked with DARPA predecessor ARPA. Amazon has big connections to the CIA and the Pentagon. The plan I have long described went like this:

1. Employ military technology to establish the Internet.

2. Sell the Internet as a place where people can freely communicate without censorship and allow that to happen until the Net becomes the central and irreversible pillar of human society. If the Internet had been highly censored from the start many would have rejected it.

3. Fund and manipulate major corporations into being to control the circulation of information on your Internet using cover stories about geeks in garages to explain how they came about. Give them unlimited funds to expand rapidly with no need to make a profit for years while non-Cult companies who need to balance the books cannot compete. You know that in these circumstances your Googles, YouTubes, Facebooks and Amazons are going to secure near monopolies by either crushing or buying up the opposition.

4. Allow freedom of expression on both the Internet and communication platforms to draw people in until the Internet is the central and irreversible pillar of human society and your communication corporations have reached a stage of near monopoly domination.

5. Then unleash your always-planned frenzy of censorship on the basis of 'where else are you going to go?' and continue to expand that until nothing remains that the Cult does not want its human targets to see.

The process was timed to hit the 'Covid' hoax to ensure the best chance possible of controlling the narrative which they knew they had to do at all costs. They were, after all, about to unleash a 'deadly virus' that didn't really exist. If you do that in an environment of free-flowing information and opinion you would be dead in the

water before you could say Gates is a psychopath. The network was in place through which the Cult-created-and-owned World Health Organization could dictate the 'Covid' narrative and response policy slavishly supported by Cult-owned Internet communication giants and mainstream media while those telling a different story were censored. Google, YouTube, Facebook and Twitter openly announced that they would do this. What else would we expect from Cult-owned operations like Facebook which former executives have confirmed set out to make the platform more addictive than cigarettes and coldly manipulates emotions of its users to sow division between people and groups and scramble the minds of the young? If Zuckerberg lives out the rest of his life without going to jail for crimes against humanity, and most emphatically against the young, it will be a travesty of justice. Still, no matter, cause and effect will catch up with him eventually and the same with Sergey Brin and Larry Page at Google with its CEO Sundar Pichai who fix the Google search results to promote Cult narratives and hide the opposition. Put the same key words into Google and other search engines like DuckDuckGo and you will see how different results can be. Wikipedia is another intensely biased 'encyclopaedia' which skews its content to the Cult agenda. YouTube links to Wikipedia's version of 'Covid' and 'climate change' on video pages in which experts in their field offer a different opinion (even that is increasingly rare with Wojcicki censorship). Into this 'Covid' silence-them network must be added government media censors, sorry 'regulators', such as Ofcom in the UK which imposed tyrannical restrictions on British broadcasters that had the effect of banning me from ever appearing. Just to debate with me about my evidence and views on 'Covid' would mean breaking the fascistic impositions of Ofcom and its CEO career government bureaucrat Melanie Dawes. Gutless British broadcasters tremble at the very thought of fascist Ofcom.

Psychos behind 'Covid'

The reason for the 'Covid' catastrophe in all its facets and forms can be seen by whom and what is driving the policies worldwide in such a coordinated way. Decisions are not being made to protect health, but to target psychology. The dominant group guiding and 'advising' government policy are not medical professionals. They are psychologists and behavioural scientists. Every major country has its own version of this phenomenon and I'll use the British example to show how it works. In many ways the British version has been affecting the wider world in the form of the huge behaviour manipulation network in the UK which operates in other countries. The network involves private companies, government, intelligence and military. The Cabinet Office is at the centre of the government 'Covid' Psyop and part-owns, with 'innovation charity' Nesta, the Behavioural Insights Team (BIT) which claims to be independent of government but patently isn't. The BIT was established in 2010 and its job is to manipulate the psyche of the population to acquiesce to government demands and so much more. It is also known as the 'Nudge Unit', a name inspired by the 2009 book by two ultra-Zionists, Cass Sunstein and Richard Thaler, called *Nudge: Improving Decisions About Health, Wealth, and Happiness*. The book, as with the Behavioural Insights Team, seeks to 'nudge' behaviour (manipulate it) to make the public follow patterns of action and perception that suit those in authority (the Cult). Sunstein is so skilled at this that he advises the World Health Organization and the UK Behavioural Insights Team and was Administrator of the White House Office of Information and Regulatory Affairs in the Obama administration. Biden appointed him to the Department of Homeland Security – another ultra-Zionist in the fold to oversee new immigration laws which is another policy the Cult wants to control. Sunstein is desperate to silence anyone exposing conspiracies and co-authored a 2008 report on the subject in which suggestions were offered to ban 'conspiracy theorizing' or impose 'some kind of tax, financial or otherwise, on those who disseminate such theories'. I guess a psychiatrist's chair is out of the question?

Sunstein's mate Richard Thaler, an 'academic affiliate' of the UK Behavioural Insights Team, is a proponent of 'behavioural economics' which is defined as the study of 'the effects of psychological, cognitive, emotional, cultural and social factors on the decisions of individuals and institutions'. Study the effects so they can be manipulated to be what you want them to be. Other leading names in the development of behavioural economics are ultra-Zionists Daniel Kahneman and Robert J. Shiller and they, with Thaler, won the Nobel Memorial Prize in Economic Sciences for their work in this field. The Behavioural Insights Team is operating at the heart of the UK government and has expanded globally through partnerships with several universities including Harvard, Oxford, Cambridge, University College London (UCL) and Pennsylvania. They claim to have 'trained' (reframed) 20,000 civil servants and run more than 750 projects involving 400 randomised controlled trials in dozens of countries' as another version of mind reframers Common Purpose. BIT works from its office in New York with cities and their agencies, as well as other partners, across the United States and Canada – this is a company part-owned by the British government Cabinet Office. An executive order by President Cult-servant Obama established a US Social and Behavioral Sciences Team in 2015. They all have the same reason for being and that's to brainwash the population directly and by brainwashing those in positions of authority.

'Covid' mind game

Another prime aspect of the UK mind-control network is the 'independent' [joke] Scientific Pandemic Insights Group on Behaviours (SPI-B) which 'provides behavioural science advice aimed at anticipating and helping people adhere to interventions that are recommended by medical or epidemiological experts'. That means manipulating public perception and behaviour to do whatever government tells them to do. It's disgusting and if they really want the public to be 'safe' this lot should all be under lock and key. According to the government website SPI-B consists of

'behavioural scientists, health and social psychologists, anthropologists and historians' and advises the Whitty-Vallance-led Scientific Advisory Group for Emergencies (SAGE) which in turn advises the government on 'the science' (it doesn't) and 'Covid' policy. When politicians say they are being guided by 'the science' this is the rabble in each country they are talking about and that 'science' is dominated by behaviour manipulators to enforce government fascism through public compliance. The Behaviour Insight Team is headed by psychologist David Solomon Halpern, a visiting professor at King's College London, and connects with a national and global web of other civilian and military organisations as the Cult moves towards its goal of fusing them into one fascistic whole in every country through its 'Fusion Doctrine'. The behaviour manipulation network involves, but is not confined to, the Foreign Office; National Security Council; government communications headquarters (GCHQ); MI5; MI6; the Cabinet Office-based Media Monitoring Unit; and the Rapid Response Unit which 'monitors digital trends to spot emerging issues; including misinformation and disinformation; and identifies the best way to respond'.

There is also the 77th Brigade of the UK military which operates like the notorious Israeli military's Unit 8200 in manipulating information and discussion on the Internet by posing as members of the public to promote the narrative and discredit those who challenge it. Here we have the military seeking to manipulate *domestic* public opinion while the Nazis in government are fine with that. Conservative Member of Parliament Tobias Ellwood, an advocate of lockdown and control through 'vaccine passports', is a Lieutenant Colonel reservist in the 77th Brigade which connects with the military operation jHub, the 'innovation centre' for the Ministry of Defence and Strategic Command. jHub has also been involved with the civilian National Health Service (NHS) in 'symptom tracing' the population. The NHS is a key part of this mind control network and produced a document in December, 2020, explaining to staff how to use psychological manipulation with different groups and ages to get them to have the DNA-manipulating 'Covid vaccine'

that's designed to cumulatively rewrite human genetics. The document, called 'Optimising Vaccination Roll Out – Do's and Dont's for all messaging, documents and "communications" in the widest sense', was published by NHS England and the NHS Improvement *Behaviour Change Unit* in partnership with Public Health England and Warwick Business School. I hear the mantra about 'save the NHS' and 'protect the NHS' when we need to scrap the NHS and start again. The current version is far too corrupt, far too anti-human and totally compromised by Cult operatives and their assets. UK government broadcast media censor Ofcom will connect into this web – as will the BBC with its tremendous Ofcom influence – to control what the public see and hear and dictate mass perception. Nuremberg trials must include personnel from all these organisations.

The fear factor

The 'Covid' hoax has led to the creation of the UK Cabinet Office-connected Joint Biosecurity Centre (JBC) which is officially described as providing 'expert advice on pandemics' using its independent [all Cult operations are 'independent'] analytical function to provide real-time analysis about infection outbreaks to identify and respond to outbreaks of Covid-19'. Another role is to advise the government on a response to spikes in infections – 'for example by closing schools or workplaces in local areas where infection levels have risen'. Put another way, promoting the Cult agenda. The Joint Biosecurity Centre is modelled on the Joint Terrorism Analysis Centre which analyses intelligence to set 'terrorism threat levels' and here again you see the fusion of civilian and military operations and intelligence that has led to military intelligence producing documents about 'vaccine hesitancy' and how it can be combated. Domestic civilian matters and opinions should not be the business of the military. The Joint Biosecurity Centre is headed by Tom Hurd, director general of the Office for Security and Counter-Terrorism from the establishment-to-its-fingertips Hurd family. His father is former Foreign Secretary Douglas Hurd. How coincidental that Tom

Hurd went to the elite Eton College and Oxford University with Boris Johnson. Imperial College with its ridiculous computer modeller Neil Ferguson will connect with this gigantic web that will itself interconnect with similar set-ups in other major and not so major countries. Compared with this Cult network the politicians, be they Boris Johnson, Donald Trump or Joe Biden, are bit-part players 'following the science'. The network of psychologists was on the 'Covid' case from the start with the aim of generating maximum fear of the 'virus' to ensure compliance by the population. A government behavioural science group known as SPI-B produced a paper in March, 2020, for discussion by the main government science advisory group known as SAGE. It was headed 'Options for increasing adherence to social distancing measures' and it said the following in a section headed 'Persuasion':

- A substantial number of people still do not feel sufficiently personally threatened; it could be that they are reassured by the low death rate in their demographic group, although levels of concern may be rising. Having a good understanding of the risk has been found to be positively associated with adoption of COVID-19 social distancing measures in Hong Kong.

- The perceived level of personal threat needs to be increased among those who are complacent, using hard-hitting evaluation of options for increasing social distancing emotional messaging. To be effective this must also empower people by making clear the actions they can take to reduce the threat.

- Responsibility to others: There seems to be insufficient understanding of, or feelings of responsibility about, people's role in transmitting the infection to others … Messaging about actions need to be framed positively in terms of protecting oneself and the community, and increase confidence that they will be effective.

- Some people will be more persuaded by appeals to play by the rules, some by duty to the community, and some to personal risk.

> All these different approaches are needed. The messaging also needs to take account of the realities of different people's lives. Messaging needs to take account of the different motivational levers and circumstances of different people.

All this could be achieved the SPI-B psychologists said by *using the media to increase the sense of personal threat* which translates as terrify the shit out of the population, including children, so they all do what we want. That's not happened has it? Those excuses for 'journalists' who wouldn't know journalism if it bit them on the arse (the great majority) have played their crucial part in serving this Cult-government Psyop to enslave their own kids and grandkids. How they live with themselves I have no idea. The psychological war has been underpinned by constant government 'Covid' propaganda in almost every television and radio ad break, plus the Internet and print media, which has pounded out the fear with taxpayers footing the bill for their own programming. The result has been people terrified of a 'virus' that doesn't exist or one with a tiny fatality rate even if you believe it does. People walk down the street and around the shops wearing face-nappies damaging their health and psychology while others report those who refuse to be that naïve to the police who turn up in their own face-nappies. I had a cameraman come to my flat and he was so frightened of 'Covid' he came in wearing a mask and refused to shake my hand in case he caught something. He had – naïveitis – and the thought that he worked in the mainstream media was both depressing and made his behaviour perfectly explainable. The fear which has gripped the minds of so many and frozen them into compliance has been carefully cultivated by these psychologists who are really psychopaths. If lives get destroyed and a lot of young people commit suicide it shows our plan is working. SPI-B then turned to compulsion on the public to comply. 'With adequate preparation, rapid change can be achieved', it said. Some countries had introduced mandatory self-isolation on a wide scale without evidence of major public unrest and a large majority of the UK's population appeared to be supportive of more coercive measures with 64 percent of adults saying they would

support putting London under a lockdown (watch the 'polls' which are designed to make people believe that public opinion is in favour or against whatever the subject in hand).

For 'aggressive protective measures' to be effective, the SPI-B paper said, special attention should be devoted to those population groups that are more at risk. Translated from the Orwellian this means making the rest of population feel guilty for not protecting the 'vulnerable' such as old people which the Cult and its agencies were about to kill on an industrial scale with lockdown, lack of treatment and the Gates 'vaccine'. Psychopath psychologists sold their guilt-trip so comprehensively that Los Angeles County Supervisor Hilda Solis reported that children were apologising (from a distance) to their parents and grandparents for bringing 'Covid' into their homes and getting them sick. '… These apologies are just some of the last words that loved ones will ever hear as they die alone,' she said. Gut-wrenchingly Solis then used this childhood tragedy to tell children to stay at home and 'keep your loved ones alive'. Imagine heaping such potentially life-long guilt on a kid when it has absolutely nothing to do with them. These people are deeply disturbed and the psychologists behind this even more so.

Uncivil war – divide and rule

Professional mind-controllers at SPI-B wanted the media to increase a sense of responsibility to others (do as you're told) and promote 'positive messaging' for those actions while in contrast to invoke 'social disapproval' by the unquestioning, obedient, community of anyone with a mind of their own. Again the compliant Goebbels-like media obliged. This is an old, old, trick employed by tyrannies the world over throughout human history. You get the target population to keep the target population in line – *your* line. SPI-B said this could 'play an important role in preventing anti-social behaviour or discouraging failure to enact pro-social behaviour'. For 'anti-social' in the Orwellian parlance of SPI-B see any behaviour that government doesn't approve. SPI-B recommendations said that 'social disapproval' should be accompanied by clear messaging and

promotion of strong collective identity – hence the government and celebrity mantra of 'we're all in this together'. Sure we are. The mind doctors have such contempt for their targets that they think some clueless comedian, actor or singer telling them to do what the government wants will be enough to win them over. We have had UK comedian Lenny Henry, actor Michael Caine and singer Elton John wheeled out to serve the propagandists by urging people to have the DNA-manipulating 'Covid' non-'vaccine'. The role of Henry and fellow black celebrities in seeking to coax a 'vaccine' reluctant black community into doing the government's will was especially stomach-turning. An emotion-manipulating script and carefully edited video featuring these black 'celebs' was such an insult to the intelligence of black people and where's the self-respect of those involved selling their souls to a fascist government agenda? Henry said he heard black people's 'legitimate worries and concerns', but people must 'trust the facts' when they were doing exactly that by not having the 'vaccine'. They had to include the obligatory reference to Black Lives Matter with the line … 'Don't let coronavirus cost even more black lives – because we matter'. My god, it was pathetic. 'I know the vaccine is safe and what it does.' How? 'I'm a comedian and it says so in my script.'

SPI-B said social disapproval needed to be carefully managed to avoid victimisation, scapegoating and misdirected criticism, but they knew that their 'recommendations' would lead to exactly that and the media were specifically used to stir-up the divide-and-conquer hostility. Those who conform like good little baa, baas, are praised while those who have seen through the tidal wave of lies are 'Covidiots'. The awake have been abused by the fast asleep for not conforming to fascism and impositions that the awake know are designed to endanger their health, dehumanise them, and tear asunder the very fabric of human society. We have had the curtain-twitchers and morons reporting neighbours and others to the face-nappied police for breaking 'Covid rules' with fascist police delighting in posting links and phone numbers where this could be done. The Cult cannot impose its will without a compliant police

and military or a compliant population willing to play their part in enslaving themselves and their kids. The words of a pastor in Nazi Germany are so appropriate today:

First they came for the socialists and I did not speak out because I was not a socialist.

Then they came for the trade unionists and I did not speak out because I was not a trade unionist.

Then they came for the Jews and I did not speak out because I was not a Jew.

Then they came for me and there was no one left to speak for me.

Those who don't learn from history are destined to repeat it and so many are.

'Covid' rules: Rewiring the mind

With the background laid out to this gigantic national and global web of psychological manipulation we can put 'Covid' rules into a clear and sinister perspective. Forget the claims about protecting health. 'Covid' rules are about dismantling the human mind, breaking the human spirit, destroying self-respect, and then putting Humpty Dumpty together again as a servile, submissive slave. Social isolation through lockdown and distancing have devastating effects on the human psyche as the psychological psychopaths well know and that's the real reason for them. Humans need contact with each other, discourse, closeness and touch, or they eventually, and literarily, go crazy. Masks, which I will address at some length, fundamentally add to the effects of isolation and the Cult agenda to dehumanise and de-individualise the population. To do this while knowing – in fact *seeking* – this outcome is the very epitome of evil and psychologists involved in this *are* the epitome of evil. They must like all the rest of the Cult demons and their assets stand trial for crimes against humanity on a scale that defies the imagination. Psychopaths in uniform use isolation to break enemy troops and agents and make them subservient and submissive to tell what they know. The technique is rightly considered a form of torture and

torture is most certainly what has been imposed on the human population.

Clinically-insane American psychologist Harry Harlow became famous for his isolation experiments in the 1950s in which he separated baby monkeys from their mothers and imprisoned them for months on end in a metal container or 'pit of despair'. They soon began to show mental distress and depression as any idiot could have predicted. Harlow put other monkeys in steel chambers for three, six or twelve months while denying them any contact with animals or humans. He said that the effects of total social isolation for six months were 'so devastating and debilitating that we had assumed initially that twelve months of isolation would not produce any additional decrement'; but twelve months of isolation 'almost obliterated the animals socially'. This is what the Cult and its psychopaths are doing to you and your children. Even monkeys in partial isolation in which they were not allowed to form relationships with other monkeys became 'aggressive and hostile, not only to others, but also towards their own bodies'. We have seen this in the young as a consequence of lockdown. UK government psychopaths launched a public relations campaign telling people not to hug each other even after they received the 'Covid-19 vaccine' which we were told with more lies would allow a return to 'normal life'. A government source told *The Telegraph*: 'It will be along the lines that it is great that you have been vaccinated, but if you are going to visit your family and hug your grandchildren there is a chance you are going to infect people you love.' The source was apparently speaking from a secure psychiatric facility. Janet Lord, director of Birmingham University's Institute of Inflammation and Ageing, said that parents and grandparents should avoid hugging their children. Well, how can I put it, Ms Lord? Fuck off. Yep, that'll do.

Destroying the kids – where are the parents?

Observe what has happened to people enslaved and isolated by lockdown as suicide and self-harm has soared worldwide,

particularly among the young denied the freedom to associate with their friends. A study of 49,000 people in English-speaking countries concluded that almost half of young adults are at clinical risk of mental health disorders. A national survey in America of 1,000 currently enrolled high school and college students found that 5 percent reported attempting suicide during the pandemic. Data from the US CDC's National Syndromic Surveillance Program from January 1st to October 17th, 2020, revealed a *31 percent* increase in mental health issues among adolescents aged 12 to 17 compared with 2019. The CDC reported that America in general suffered the biggest drop in life expectancy since World War Two as it fell by a year in the first half of 2020 as a result of 'deaths of despair' – overdoses and suicides. Deaths of despair have leapt by more than 20 percent during lockdown and include the highest number of fatal overdoses ever recorded in a single year – 81,000. Internet addiction is another consequence of being isolated at home which lowers interest in physical activities as kids fall into inertia and what's the point? Children and young people are losing hope and giving up on life, sometimes literally. A 14-year-old boy killed himself in Maryland because he had 'given up' when his school district didn't reopen; an 11-year-old boy shot himself during a zoom class; a teenager in Maine succumbed to the isolation of the 'pandemic' when he ended his life after experiencing a disrupted senior year at school. Children as young as nine have taken their life and all these stories can be repeated around the world. Careers are being destroyed before they start and that includes those in sport in which promising youngsters have not been able to take part. The plan of the psycho-psychologists is working all right. Researchers at Cambridge University found that lockdowns cause significant harm to children's mental health. Their study was published in the *Archives of Disease in Childhood*, and followed 168 children aged between 7 and 11. The researchers concluded:

> During the UK lockdown, children's depression symptoms have increased substantially, relative to before lockdown. The scale of this effect has direct relevance for the continuation of different elements of lockdown policy, such as complete or partial school closures …

… Specifically, we observed a statistically significant increase in ratings of depression, with a medium-to-large effect size. Our findings emphasise the need to incorporate the potential impact of lockdown on child mental health in planning the ongoing response to the global pandemic and the recovery from it.

Not a chance when the Cult's psycho-psychologists were getting exactly what they wanted. The UK's Royal College of Paediatrics and Child Health has urged parents to look for signs of eating disorders in children and young people after a three to four fold increase. Specialists say the 'pandemic' is a major reason behind the rise. You don't say. The College said isolation from friends during school closures, exam cancellations, loss of extra-curricular activities like sport, and an increased use of social media were all contributory factors along with fears about the virus (psycho-psychologists again), family finances, and students being forced to quarantine. Doctors said young people were becoming severely ill by the time they were seen with 'Covid' regulations reducing face-to-face consultations. Nor is it only the young that have been devastated by the psychopaths. Like all bullies and cowards the Cult is targeting the young, elderly, weak and infirm. A typical story was told by a British lady called Lynn Parker who was not allowed to visit her husband in 2020 for the last ten and half months of his life 'when he needed me most' between March 20th and when he died on December 19th. This vacates the criminal and enters the territory of evil. The emotional impact on the immune system alone is immense as are the number of people of all ages worldwide who have died as a result of Cult-demanded, Gates-demanded, lockdowns.

Isolation is torture

The experience of imposing solitary confinement on millions of prisoners around the world has shown how a large percentage become 'actively psychotic and/or acutely suicidal'. Social isolation has been found to trigger 'a specific psychiatric syndrome, characterized by hallucinations; panic attacks; overt paranoia; diminished impulse control; hypersensitivity to external stimuli; and difficulties with thinking, concentration and memory'. Juan Mendez,

a United Nations rapporteur (investigator), said that isolation is a form of torture. Research has shown that even after isolation prisoners find it far more difficult to make social connections and I remember chatting to a shop assistant after one lockdown who told me that when her young son met another child again he had no idea how to act or what to do. Hannah Flanagan, Director of Emergency Services at Journey Mental Health Center in Dane County, Wisconsin, said: 'The specificity about Covid social distancing and isolation that we've come across as contributing factors to the suicides are really new to us this year.' But they are not new to those that devised them. They are getting the effect they want as the population is psychologically dismantled to be rebuilt in a totally different way. Children and the young are particularly targeted. They will be the adults when the full-on fascist AI-controlled technocracy is planned to be imposed and they are being prepared to meekly submit. At the same time older people who still have a memory of what life was like before – and how fascist the new normal really is – are being deleted. You are going to see efforts to turn the young against the old to support this geriatric genocide. Hannah Flanagan said the big increase in suicide in her county proved that social isolation is not only harmful, but deadly. Studies have shown that isolation from others is one of the main risk factors in suicide and even more so with women. Warnings that lockdown could create a 'perfect storm' for suicide were ignored. After all this was one of the *reasons* for lockdown. Suicide, however, is only the most extreme of isolation consequences. There are many others. Dr Dhruv Khullar, assistant professor of healthcare policy at Weill Cornell Medical College, said in a *New York Times* article in 2016 long before the fake 'pandemic':

> A wave of new research suggests social separation is bad for us. Individuals with less social connection have disrupted sleep patterns, altered immune systems, more inflammation and higher levels of stress hormones. One recent study found that isolation increases the risk of heart disease by 29 percent and stroke by 32 percent. Another analysis that pooled data from 70 studies and 3.4 million people found that socially isolated individuals had a 30 percent higher risk of dying in the next seven years, and that this effect was largest in middle age.

Loneliness can accelerate cognitive decline in older adults, and isolated individuals are twice as likely to die prematurely as those with more robust social interactions. These effects start early: Socially isolated children have significantly poorer health 20 years later, even after controlling for other factors. All told, loneliness is as important a risk factor for early death as obesity and smoking.

There you have proof from that one article alone four years before 2020 that those who have enforced lockdown, social distancing and isolation knew what the effect would be and that is even more so with professional psychologists that have been driving the policy across the globe. We can go back even further to the years 2000 and 2003 and the start of a major study on the effects of isolation on health by Dr Janine Gronewold and Professor Dirk M. Hermann at the University Hospital in Essen, Germany, who analysed data on 4,316 people with an average age of 59 who were recruited for the long-term research project. They found that socially isolated people are more than 40 percent more likely to have a heart attack, stroke, or other major cardiovascular event and nearly 50 percent more likely to die from any cause. Given the financial Armageddon unleashed by lockdown we should note that the study found a relationship between increased cardiovascular risk and lack of financial support. After excluding other factors social isolation was still connected to a 44 percent increased risk of cardiovascular problems and a 47 percent increased risk of death by any cause. Lack of financial support was associated with a 30 percent increase in the risk of cardiovascular health events. Dr Gronewold said it had been known for some time that feeling lonely or lacking contact with close friends and family can have an impact on physical health and the study had shown that having strong social relationships is of high importance for heart health. Gronewold said they didn't understand yet why people who are socially isolated have such poor health outcomes, but this was obviously a worrying finding, particularly during these times of prolonged social distancing. Well, it can be explained on many levels. You only have to identify the point in the body where people feel loneliness and missing people they are parted from – it's in the centre of the chest where they feel the ache of loneliness and the ache of missing people. 'My heart aches for

you' … 'My heart aches for some company.' I will explain this more in the chapter Escaping Wetiko, but when you realise that the body is the mind – they are expressions of each other – the reason why state of the mind dictates state of the body becomes clear.

American psychologist Ranjit Powar was highlighting the effects of lockdown isolation as early as April, 2020. She said humans have evolved to be social creatures and are wired to live in interactive groups. Being isolated from family, friends and colleagues could be unbalancing and traumatic for most people and could result in short or even long-term psychological and physical health problems. An increase in levels of anxiety, aggression, depression, forgetfulness and hallucinations were possible psychological effects of isolation. 'Mental conditions may be precipitated for those with underlying pre-existing susceptibilities and show up in many others without any pre-condition.' Powar said personal relationships helped us cope with stress and if we lost this outlet for letting off steam the result can be a big emotional void which, for an average person, was difficult to deal with. 'Just a few days of isolation can cause increased levels of anxiety and depression' – so what the hell has been the effect on the global population of *18 months* of this at the time of writing? Powar said: 'Add to it the looming threat of a dreadful disease being repeatedly hammered in through the media and you have a recipe for many shades of mental and physical distress.' For those with a house and a garden it is easy to forget that billions have had to endure lockdown isolation in tiny overcrowded flats and apartments with nowhere to go outside. The psychological and physical consequences of this are unimaginable and with lunatic and abusive partners and parents the consequences have led to tremendous increases in domestic and child abuse and alcoholism as people seek to shut out the horror. Ranjit Powar said:

> Staying in a confined space with family is not all a rosy picture for everyone. It can be extremely oppressive and claustrophobic for large low-income families huddled together in small single-room houses. Children here are not lucky enough to have many board/electronic games or books to keep them occupied.

Add to it the deep insecurity of running out of funds for food and basic necessities. On the other hand, there are people with dysfunctional family dynamics, such as domineering, abusive or alcoholic partners, siblings or parents which makes staying home a period of trial. Incidence of suicide and physical abuse against women has shown a worldwide increase. Heightened anxiety and depression also affect a person's immune system, making them more susceptible to illness.

To think that Powar's article was published on April 11th, 2020.

Six-feet fantasy

Social (unsocial) distancing demanded that people stay six feet or two metres apart. UK government advisor Robert Dingwall from the New and Emerging Respiratory Virus Threats Advisory Group said in a radio interview that the two-metre rule was 'conjured up out of nowhere' and was not based on science. No, it was not based on *medical* science, but it didn't come out of nowhere. The distance related to *psychological* science. Six feet/two metres was adopted in many countries and we were told by people like the criminal Anthony Fauci and his ilk that it was founded on science. Many schools could not reopen because they did not have the space for six-feet distancing. Then in March, 2021, after a year of six-feet 'science', a study published in the *Journal of Infectious Diseases* involving more than 500,000 students and almost 100,000 staff over 16 weeks revealed no significant difference in 'Covid' cases between six feet and three feet and Fauci changed his tune. Now three feet was okay. There is no difference between six feet and three *inches* when there is no 'virus' and they got away with six feet for psychological reasons for as long as they could. I hear journalists and others talk about 'unintended consequences' of lockdown. They are not *unintended* at all; they have been coldly-calculated for a specific outcome of human control and that's why super-psychopaths like Gates have called for them so vehemently. Super-psychopath psychologists have demanded them and psychopathic or clueless, spineless, politicians have gone along with them by 'following the science'. But it's not science at all. 'Science' is not what is; it's only what people can be manipulated to believe it is. The whole 'Covid' catastrophe is

founded on mind control. Three word or three statement mantras issued by the UK government are a well-known mind control technique and so we've had 'Stay home/protect the NHS/save lives', 'Stay alert/control the virus/save lives' and 'hands/face/space'. One of the most vocal proponents of extreme 'Covid' rules in the UK has been Professor Susan Michie, a member of the British Communist Party, who is not a medical professional. Michie is the director of the Centre for Behaviour Change at University College London. She is a *behavioural psychologist* and another filthy rich 'Marxist' who praised China's draconian lockdown. She was known by fellow students at Oxford University as 'Stalin's nanny' for her extreme Marxism. Michie is an influential member of the UK government's Scientific Advisory Group for Emergencies (SAGE) and behavioural manipulation groups which have dominated 'Covid' policy. She is a consultant adviser to the World Health Organization on 'Covid-19' and behaviour. Why the hell are lockdowns anything to do with her when they are claimed to be about health? Why does a behavioural psychologist from a group charged with changing the behaviour of the public want lockdown, human isolation and mandatory masks? Does that question really need an answer? Michie *absolutely* has to explain herself before a Nuremberg court when humanity takes back its world again and even more so when you see the consequences of masks that she demands are compulsory. This is a Michie classic:

> The benefits of getting primary school children to wear masks is that regardless of what little degree of transmission is occurring in those age groups it could help normalise the practice. Young children wearing masks may be more likely to get their families to accept masks.

Those words alone should carry a prison sentence when you ponder on the callous disregard for children involved and what a statement it makes about the mind and motivations of Susan Michie. What a lovely lady and what she said there encapsulates the mentality of the psychopaths behind the 'Covid' horror. Let us compare what Michie said with a countrywide study in Germany published at researchsquare.com involving 25,000 school children and 17,854 health complaints submitted by parents. Researchers

found that masks are harming children physically, psychologically, and behaviourally with 24 health issues associated with mask wearing. They include: shortness of breath (29.7%); dizziness (26.4%); increased headaches (53%); difficulty concentrating (50%); drowsiness or fatigue (37%); and malaise (42%). Nearly a third of children experienced more sleep issues than before and a quarter developed new fears. Researchers found health issues and other impairments in 68 percent of masked children covering their faces for an average of 4.5 hours a day. Hundreds of those taking part experienced accelerated respiration, tightness in the chest, weakness, and short-term impairment of consciousness. A reminder of what Michie said again:

> The benefits of getting primary school children to wear masks is that regardless of what little degree of transmission is occurring in those age groups it could help normalise the practice. Young children wearing masks may be more likely to get their families to accept masks.

Psychopaths in government and psychology now have children and young people – plus all the adults – wearing masks for hours on end while clueless teachers impose the will of the psychopaths on the young they should be protecting. What the hell are parents doing?

Cult lab rats

We have some schools already imposing on students microchipped buzzers that activate when they get 'too close' to their pals in the way they do with lab rats. How apt. To the Cult and its brain-dead servants our children *are* lab rats being conditioned to be unquestioning, dehumanised slaves for the rest of their lives. Children and young people are being weaned and frightened away from the most natural human instincts including closeness and touch. I have tracked in the books over the years how schools were banning pupils from greeting each other with a hug and the whole Cult-induced Me Too movement has terrified men and boys from a relaxed and natural interaction with female friends and work colleagues to the point where many men try never to be in a room

alone with a woman that's not their partner. Airhead celebrities have as always played their virtue-signalling part in making this happen with their gross exaggeration. For every monster like Harvey Weinstein there are at least tens of thousands of men that don't treat women like that; but everyone must be branded the same and policy changed for them as well as the monster. I am going to be using the word 'dehumanise' many times in this chapter because that is what the Cult is seeking to do and it goes very deep as we shall see. Don't let them kid you that social distancing is planned to end one day. That's not the idea. We are seeing more governments and companies funding and producing wearable gadgets to keep people apart and they would not be doing that if this was meant to be short-term. A tech start-up company backed by GCHQ, the British Intelligence and military surveillance headquarters, has created a social distancing wrist sensor that alerts people when they get too close to others. The CIA has also supported tech companies developing similar devices. The wearable sensor was developed by Tended, one of a number of start-up companies supported by GCHQ (see the CIA and DARPA). The device can be worn on the wrist or as a tag on the waistband and will vibrate whenever someone wearing the device breaches social distancing and gets anywhere near natural human contact. The company had a lucky break in that it was developing a distancing sensor when the 'Covid' hoax arrived which immediately provided a potentially enormous market. How fortunate. The government in big-time Cult-controlled Ontario in Canada is investing $2.5 million in wearable contact tracing technology that 'will alert users if they may have been exposed to the Covid-19 in the workplace and will beep or vibrate if they are within six feet of another person'. Facedrive Inc., the technology company behind this, was founded in 2016 with funding from the Ontario Together Fund and obviously they, too, had a prophet on the board of directors. The human surveillance and control technology is called TraceSCAN and would be worn by the human cyborgs in places such as airports, workplaces, construction sites, care homes and … *schools*.

I emphasise schools with children and young people the prime targets. You know what is planned for society as a whole if you keep your eyes on the schools. They have always been places where the state program the next generation of slaves to be its compliant worker-ants – or Woker-ants these days; but in the mist of the 'Covid' madness they have been transformed into mind laboratories on a scale never seen before. Teachers and head teachers are just as programmed as the kids – often more so. Children are kept apart from human interaction by walk lanes, classroom distancing, staggered meal times, masks, and the rolling-out of buzzer systems. Schools are now physically laid out as a laboratory maze for lab-rats. Lunatics at a school in Anchorage, Alaska, who should be prosecuted for child abuse, took away desks and forced children to kneel (know your place) on a mat for five hours a day while wearing a mask and using their chairs as a desk. How this was supposed to impact on a 'virus' only these clinically insane people can tell you and even then it would be clap-trap. The school banned recess (interaction), art classes (creativity), and physical exercise (getting body and mind moving out of inertia). Everyone behind this outrage should be in jail or better still a mental institution. The behavioural manipulators are all for this dystopian approach to schools. Professor Susan Michie, the mind-doctor and British Communist Party member, said it was wrong to say that schools were safe. They had to be made so by 'distancing', masks and ventilation (sitting all day in the cold). I must ask this lady round for dinner on a night I know I am going to be out and not back for weeks. She probably wouldn't be able to make it, anyway, with all the visits to her own psychologist she must have block-booked.

Masking identity

I know how shocking it must be for you that a behaviour manipulator like Michie wants everyone to wear masks which have long been a feature of mind-control programs like the infamous MKUltra in the United States, but, there we are. We live and learn. I spent many years from 1996 to right across the millennium

researching mind control in detail on both sides of the Atlantic and elsewhere. I met a large number of mind-control survivors and many had been held captive in body and mind by MKUltra. MK stands for mind-control, but employs the German spelling in deference to the Nazis spirited out of Germany at the end of World War Two by Operation Paperclip in which the US authorities, with help from the Vatican, transported Nazi mind-controllers and engineers to America to continue their work. Many of them were behind the creation of NASA and they included Nazi scientist and SS officer Wernher von Braun who swapped designing V-2 rockets to bombard London with designing the Saturn V rockets that powered the NASA moon programme's Apollo craft. I think I may have mentioned that the Cult has no borders. Among Paperclip escapees was Josef Mengele, the Angel of Death in the Nazi concentration camps where he conducted mind and genetic experiments on children often using twins to provide a control twin to measure the impact of his 'work' on the other. If you want to observe the Cult mentality in all its extremes of evil then look into the life of Mengele. I have met many people who suffered mercilessly under Mengele in the United States where he operated under the name Dr Greene and became a stalwart of MKUltra programming and torture. Among his locations was the underground facility in the Mojave Desert in California called the China Lake Naval Weapons Station which is almost entirely below the surface. My books *The Biggest Secret*, *Children of the Matrix* and *The Perception Deception* have the detailed background to MKUltra.

The best-known MKUltra survivor is American Cathy O'Brien. I first met her and her late partner Mark Phillips at a conference in Colorado in 1996. Mark helped her escape and deprogram from decades of captivity in an offshoot of MKUltra known as Project Monarch in which 'sex slaves' were provided for the rich and famous including Father George Bush, Dick Cheney and the Clintons. Read Cathy and Mark's book *Trance-Formation of America* and if you are new to this you will be shocked to the core. I read it in 1996 shortly before, with the usual synchronicity of my life, I found

myself given a book table at the conference right next to hers. MKUltra never ended despite being very publicly exposed (only a small part of it) in the 1970s and continues in other guises. I am still in touch with Cathy. She contacted me during 2020 after masks became compulsory in many countries to tell me how they were used as part of MKUltra programming. I had been observing 'Covid regulations' and the relationship between authority and public for months. I saw techniques that I knew were employed on individuals in MKUltra being used on the global population. I had read many books and manuals on mind control including one called *Silent Weapons for Quiet Wars* which came to light in the 1980s and was a guide on how to perceptually program on a mass scale. 'Silent Weapons' refers to mind-control. I remembered a line from the manual as governments, medical authorities and law enforcement agencies have so obviously talked to – or rather at – the adult population since the 'Covid' hoax began as if they are children. The document said:

> If a person is spoken to by a T.V. advertiser as if he were a twelve-year-old, then, due to suggestibility, he will, with a certain probability, respond or react to that suggestion with the uncritical response of a twelve-year-old and will reach in to his economic reservoir and deliver its energy to buy that product on impulse when he passes it in the store.

That's why authority has spoken to adults like children since all this began.

Why did Michael Jackson wear masks?

Every aspect of the 'Covid' narrative has mind-control as its central theme. Cathy O'Brien wrote an article for davidicke.com about the connection between masks and mind control. Her daughter Kelly who I first met in the 1990s was born while Cathy was still held captive in MKUltra. Kelly was forced to wear a mask as part of her programming from the age of *two* to dehumanise her, target her sense of individuality and reduce the amount of oxygen her brain and body received. *Bingo*. This is the real reason for compulsory

masks, why they have been enforced en masse, and why they seek to increase the number they demand you wear. First one, then two, with one disgraceful alleged 'doctor' recommending four which is nothing less than a death sentence. Where and how often they must be worn is being expanded for the purpose of mass mind control and damaging respiratory health which they can call 'Covid-19'. Canada's government headed by the man-child Justin Trudeau, says it's fine for children of two and older to wear masks. An insane 'study' in Italy involving just 47 children concluded there was no problem for babies as young as *four months* wearing them. Even after people were 'vaccinated' they were still told to wear masks by the criminal that is Anthony Fauci. Cathy wrote that mandating masks is allowing the authorities literally to control the air we breathe which is what was done in MKUltra. You might recall how the singer Michael Jackson wore masks and there is a reason for that. He was subjected to MKUltra mind control through Project Monarch and his psyche was scrambled by these simpletons. Cathy wrote:

> In MKUltra Project Monarch mind control, Michael Jackson had to wear a mask to silence his voice so he could not reach out for help. Remember how he developed that whisper voice when he wasn't singing? Masks control the mind from the outside in, like the redefining of words is doing. By controlling what we can and cannot say for fear of being labeled racist or beaten, for example, it ultimately controls thought that drives our words and ultimately actions (or lack thereof).
>
> Likewise, a mask muffles our speech so that we are not heard, which controls voice … words … mind. This is Mind Control. Masks are an obvious mind control device, and I am disturbed so many people are complying on a global scale. Masks depersonalize while making a person feel as though they have no voice. It is a barrier to others. People who would never choose to comply but are forced to wear a mask in order to keep their job, and ultimately their family fed, are compromised. They often feel shame and are subdued. People have stopped talking with each other while media controls the narrative.

The 'no voice' theme has often become literal with train passengers told not to speak to each other in case they pass on the 'virus', singing banned for the same reason and bonkers California officials telling people riding roller coasters that they cannot shout and scream. Cathy said she heard every day from healed MKUltra survivors who cannot wear a mask without flashing back on ways

their breathing was controlled – 'from ball gags and penises to water boarding'. She said that through the years when she saw images of people in China wearing masks 'due to pollution' that it was really to control their oxygen levels. 'I knew it was as much of a population control mechanism of depersonalisation as are burkas', she said. Masks are another Chinese communist/fascist method of control that has been swept across the West as the West becomes China at lightning speed since we entered 2020.

Mask-19

There are other reasons for mandatory masks and these include destroying respiratory health to call it 'Covid-19' and stunting brain development of children and the young. Dr Margarite Griesz-Brisson MD, PhD, is a Consultant Neurologist and Neurophysiologist and the Founder and Medical Director of the London Neurology and Pain Clinic. Her CV goes down the street and round the corner. She is clearly someone who cares about people and won't parrot the propaganda. Griesz-Brisson has a PhD in pharmacology, with special interest in neurotoxicology, environmental medicine, neuroregeneration and neuroplasticity (the way the brain can change in the light of information received). She went public in October, 2020, with a passionate warning about the effects of mask-wearing laws:

> The reinhalation of our exhaled air will without a doubt create oxygen deficiency and a flooding of carbon dioxide. We know that the human brain is very sensitive to oxygen deprivation. There are nerve cells for example in the hippocampus that can't be longer than 3 minutes without oxygen – they cannot survive. The acute warning symptoms are headaches, drowsiness, dizziness, issues in concentration, slowing down of reaction time – reactions of the cognitive system.

Oh, I know, let's tell bus, truck and taxi drivers to wear them and people working machinery. How about pilots, doctors and police? Griesz-Brisson makes the important point that while the symptoms she mentions may fade as the body readjusts this does not alter the fact that people continue to operate in oxygen deficit with long list of

potential consequences. She said it was well known that neurodegenerative diseases take years or decades to develop. 'If today you forget your phone number, the breakdown in your brain would have already started 20 or 30 years ago.' She said degenerative processes in your brain are getting amplified as your oxygen deprivation continues through wearing a mask. Nerve cells in the brain are unable to divide themselves normally in these circumstances and lost nerve cells will no longer be regenerated. 'What is gone is gone.' Now consider that people like shop workers and *schoolchildren* are wearing masks for hours every day. What in the name of sanity is going to be happening to them? 'I do not wear a mask, I need my brain to think', Griesz-Brisson said, 'I want to have a clear head when I deal with my patients and not be in a carbon dioxide-induced anaesthesia'. If you are told to wear a mask anywhere ask the organisation, police, store, whatever, for their risk assessment on the dangers and negative effects on mind and body of enforcing mask-wearing. They won't have one because it has never been done not even by government. All of them must be subject to class-action lawsuits as the consequences come to light. They don't do mask risk assessments for an obvious reason. They know what the conclusions would be and independent scientific studies that *have* been done tell a horror story of consequences.

'Masks are criminal'

Dr Griesz-Brisson said that for children and adolescents, masks are an absolute no-no. They had an extremely active and adaptive immune system and their brain was incredibly active with so much to learn. 'The child's brain, or the youth's brain, is thirsting for oxygen.' The more metabolically active an organ was, the more oxygen it required; and in children and adolescents every organ was metabolically active. Griesz-Brisson said that to deprive a child's or adolescent's brain of oxygen, or to restrict it in any way, was not only dangerous to their health, it was absolutely criminal. 'Oxygen deficiency inhibits the development of the brain, and the damage that has taken place as a result CANNOT be reversed.' Mind

manipulators of MKUltra put masks on two-year-olds they wanted to neurologically rewire and you can see why. Griesz-Brisson said a child needs the brain to learn and the brain needs oxygen to function. 'We don't need a clinical study for that. This is simple, indisputable physiology.' Consciously and purposely induced oxygen deficiency was an absolutely deliberate health hazard, and an absolute medical contraindication which means that 'this drug, this therapy, this method or measure should not be used, and is not allowed to be used'. To coerce an entire population to use an absolute medical contraindication by force, she said, there had to be definite and serious reasons and the reasons must be presented to competent interdisciplinary and independent bodies to be verified and authorised. She had this warning of the consequences that were coming if mask wearing continued:

> When, in ten years, dementia is going to increase exponentially, and the younger generations couldn't reach their god-given potential, it won't help to say 'we didn't need the masks'. I know how damaging oxygen deprivation is for the brain, cardiologists know how damaging it is for the heart, pulmonologists know how damaging it is for the lungs. Oxygen deprivation damages every single organ. Where are our health departments, our health insurance, our medical associations? It would have been their duty to be vehemently against the lockdown and to stop it and stop it from the very beginning.
>
> Why do the medical boards issue punishments to doctors who give people exemptions? Does the person or the doctor seriously have to prove that oxygen deprivation harms people? What kind of medicine are our doctors and medical associations representing? Who is responsible for this crime? The ones who want to enforce it? The ones who let it happen and play along, or the ones who don't prevent it?

All of the organisations and people she mentions there either answer directly to the Cult or do whatever hierarchical levels above them tell them to do. The outcome of both is the same. 'It's not about masks, it's not about viruses, it's certainly not about your health', Griesz-Brisson said. 'It is about much, much more. I am not participating. I am not afraid.' They were taking our air to breathe and there was no unfounded medical exemption from face masks. Oxygen deprivation was dangerous for every single brain. It had to be the free decision of every human being whether they want to

wear a mask that was absolutely ineffective to protect themselves from a virus. She ended by rightly identifying where the responsibility lies for all this:

> The imperative of the hour is personal responsibility. We are responsible for what we think, not the media. We are responsible for what we do, not our superiors. We are responsible for our health, not the World Health Organization. And we are responsible for what happens in our country, not the government.

Halle-bloody-lujah.

But surgeons wear masks, right?

Independent studies of mask-wearing have produced a long list of reports detailing mental, emotional and physical dangers. What a definition of insanity to see police officers imposing mask-wearing on the public which will cumulatively damage their health while the police themselves wear masks that will cumulatively damage *their* health. It's utter madness and both public and police do this because 'the government says so' – yes a government of brain-donor idiots like UK Health Secretary Matt Hancock reading the 'follow the science' scripts of psychopathic, lunatic psychologists. The response you get from Stockholm syndrome sufferers defending the very authorities that are destroying them and their families is that 'surgeons wear masks'. This is considered the game, set and match that they must work and don't cause oxygen deficit. Well, actually, scientific studies have shown that they *do* and oxygen levels are monitored in operating theatres to compensate. Surgeons wear masks to stop spittle and such like dropping into open wounds – not to stop 'viral particles' which are so miniscule they can only be seen through an electron microscope. Holes in the masks are significantly bigger than 'viral particles' and if you sneeze or cough they will breach the mask. I watched an incredibly disingenuous 'experiment' that claimed to prove that masks work in catching 'virus' material from the mouth and nose. They did this with a slow motion camera and the mask did block big stuff which stayed inside the mask and

against the face to be breathed in or cause infections on the face as we have seen with many children. 'Viral particles', however, would never have been picked up by the camera as they came through the mask when they are far too small to be seen. The 'experiment' was therefore disingenuous *and* useless.

Studies have concluded that wearing masks in operating theatres (and thus elsewhere) make no difference to preventing infection while the opposite is true with toxic shite building up in the mask and this had led to an explosion in tooth decay and gum disease dubbed by dentists 'mask mouth'. You might have seen the Internet video of a furious American doctor urging people to take off their masks after a four-year-old patient had been rushed to hospital the night before and nearly died with a lung infection that doctors sourced to mask wearing. A study in the journal *Cancer Discovery* found that inhalation of harmful microbes can contribute to advanced stage lung cancer in adults and long-term use of masks can help breed dangerous pathogens. Microbiologists have said frequent mask wearing creates a moist environment in which microbes can grow and proliferate before entering the lungs. The Canadian Agency for Drugs and Technologies in Health, or CADTH, a Canadian national organisation that provides research and analysis to healthcare decision-makers, said this as long ago as 2013 in a report entitled 'Use of Surgical Masks in the Operating Room: A Review of the Clinical Effectiveness and Guidelines'. It said:

- No evidence was found to support the use of surgical face masks to reduce the frequency of surgical site infections
- No evidence was found on the effectiveness of wearing surgical face masks to protect staff from infectious material in the operating room.
- Guidelines recommend the use of surgical face masks by staff in the operating room to protect both operating room staff and patients (despite the lack of evidence).

We were told that the world could go back to 'normal' with the arrival of the 'vaccines'. When they came, fraudulent as they are, the story changed as I knew that it would. We are in the midst of transforming 'normal', not going back to it. Mary Ramsay, head of immunisation at Public Health England, echoed the words of US criminal Anthony Fauci who said masks and other regulations must stay no matter if people are vaccinated. The Fauci idiot continued to wear two masks – different colours so both could be clearly seen – after he *claimed* to have been vaccinated. Senator Rand Paul told Fauci in one exchange that his double-masks were 'theatre' and he was right. It's all theatre. Mary Ramsay back-tracked on the vaccine-return-to-normal theme when she said the public may need to wear masks and social-distance for years despite the jabs. 'People have got used to those lower-level restrictions now, and [they] can live with them', she said telling us what the idea has been all along. 'The vaccine does not give you a pass, even if you have had it, you must continue to follow all the guidelines' said a Public Health England statement which reneged on what we had been told before and made having the 'vaccine' irrelevant to 'normality' even by the official story. Spain's fascist government trumped everyone by passing a law mandating the wearing of masks on the beach and even when swimming in the sea. The move would have devastated what's left of the Spanish tourist industry, posed potential breathing dangers to swimmers and had Northern European sunbathers walking around with their forehead brown and the rest of their face white as a sheet. The ruling was so crazy that it had to be retracted after pressure from public and tourist industry, but it confirmed where the Cult wants to go with masks and how clinically insane authority has become. The determination to make masks permanent and hide the serious dangers to body and mind can be seen in the censorship of scientist Professor Denis Rancourt by Bill Gates-funded academic publishing website ResearchGate over his papers exposing the dangers and uselessness of masks. Rancourt said:

ResearchGate today has permanently locked my account, which I have had since 2015. Their reasons graphically show the nature of their attack against democracy, and their corruption of

science … By their obscene non-logic, a scientific review of science articles reporting on harms caused by face masks has a 'potential to cause harm'. No criticism of the psychological device (face masks) is tolerated, if the said criticism shows potential to influence public policy.

This is what happens in a fascist world.

Where are the 'greens' (again)?

Other dangers of wearing masks especially regularly relate to the inhalation of minute plastic fibres into the lungs and the deluge of discarded masks in the environment and oceans. Estimates predicted that more than 1.5 billion disposable masks will end up in the world's oceans every year polluting the water with tons of plastic and endangering marine wildlife. Studies project that humans are using 129 billion face masks each month worldwide – about three million a minute. Most are disposable and made from plastic, non-biodegradable microfibers that break down into smaller plastic particles that become widespread in ecosystems. They are littering cities, clogging sewage channels and turning up in bodies of water. I have written in other books about the immense amounts of microplastics from endless sources now being absorbed into the body. Rolf Halden, director of the Arizona State University (ASU) Biodesign Center for Environmental Health Engineering, was the senior researcher in a 2020 study that analysed 47 human tissue samples and found microplastics in all of them. 'We have detected these chemicals of plastics in every single organ that we have investigated', he said. I wrote in *The Answer* about the world being deluged with microplastics. A study by the Worldwide Fund for Nature (WWF) found that people are consuming on average every week some 2,000 tiny pieces of plastic mostly through water and also through marine life and the air. Every year humans are ingesting enough microplastics to fill a heaped dinner plate and in a life-time of 79 years it is enough to fill two large waste bins. Marco Lambertini, WWF International director general said: 'Not only are plastics polluting our oceans and waterways and killing marine life – it's in all of us and we can't escape consuming plastics,' American

geologists found tiny plastic fibres, beads and shards in rainwater samples collected from the remote slopes of the Rocky Mountain National Park near Denver, Colorado. Their report was headed: 'It is raining plastic.' Rachel Adams, senior lecturer in Biomedical Science at Cardiff Metropolitan University, said that among health consequences are internal inflammation and immune responses to a 'foreign body'. She further pointed out that microplastics become carriers of toxins including mercury, pesticides and dioxins (a known cause of cancer and reproductive and developmental problems). These toxins accumulate in the fatty tissues once they enter the body through microplastics. Now this is being compounded massively by people putting plastic on their face and throwing it away.

Workers exposed to polypropylene plastic fibres known as 'flock' have developed 'flock worker's lung' from inhaling small pieces of the flock fibres which can damage lung tissue, reduce breathing capacity and exacerbate other respiratory problems. *Now* … commonly used surgical masks have three layers of melt-blown textiles made of … polypropylene. We have billions of people putting these microplastics against their mouth, nose and face for hours at a time day after day in the form of masks. How does anyone think that will work out? I mean – what could possibly go wrong? We posted a number of scientific studies on this at davidicke.com, but when I went back to them as I was writing this book the links to the science research website where they were hosted were dead. Anything that challenges the official narrative in any way is either censored or vilified. The official narrative is so unsupportable by the evidence that only deleting the truth can protect it. A study by Chinese scientists still survived – with the usual twist which it why it was still active, I guess. Yes, they found that virtually all the masks they tested increased the daily intake of microplastic fibres, but people should still wear them because the danger from the 'virus' was worse said the crazy 'team' from the Institute of Hydrobiology in Wuhan. Scientists first discovered microplastics in lung tissue of some patients who died of lung cancer

in the 1990s. Subsequent studies have confirmed the potential health damage with the plastic degrading slowly and remaining in the lungs to accumulate in volume. Wuhan researchers used a machine simulating human breathing to establish that masks shed up to nearly 4,000 microplastic fibres in a month with reused masks producing more. Scientists said some masks are laced with toxic chemicals and a variety of compounds seriously restricted for both health and environmental reasons. They include cobalt (used in blue dye) and formaldehyde known to cause watery eyes, burning sensations in the eyes, nose, and throat, plus coughing, wheezing and nausea. No – that must be 'Covid-19'.

Mask 'worms'

There is another and potentially even more sinister content of masks. Mostly new masks of different makes filmed under a microscope around the world have been found to contain strange black fibres or 'worms' that appear to move or 'crawl' by themselves and react to heat and water. The nearest I have seen to them are the self-replicating fibres that are pulled out through the skin of those suffering from Morgellons disease which has been connected to the phenomena of 'chemtrails' which I will bring into the story later on. Morgellons fibres continue to grow outside the body and have a form of artificial intelligence. Black 'worm' fibres in masks have that kind of feel to them and there is a nanotechnology technique called 'worm micelles' which carry and release drugs or anything else you want to deliver to the body. For sure the suppression of humanity by mind altering drugs is the Cult agenda big time and the more excuses they can find to gain access to the body the more opportunities there are to make that happen whether through 'vaccines' or masks pushed against the mouth and nose for hours on end.

So let us summarise the pros and cons of masks:

Against masks: Breathing in your own carbon dioxide; depriving the body and brain of sufficient oxygen; build-up of toxins in the mask that can be breathed into the lungs and cause rashes on the face and 'mask-mouth'; breathing microplastic fibres and toxic chemicals into the lungs; dehumanisation and deleting individualisation by literally making people faceless; destroying human emotional interaction through facial expression and deleting parental connection with their babies which look for guidance to their facial expression.

For masks: They don't protect you from a 'virus' that doesn't exist and even if it did 'viral' particles are so minute they are smaller than the holes in the mask.

Governments, police, supermarkets, businesses, transport companies, and all the rest who seek to impose masks have done no risk assessment on their consequences for health and psychology and are now open to group lawsuits when the impact becomes clear with a cumulative epidemic of respiratory and other disease. Authorities will try to exploit these effects and hide the real cause by dubbing them 'Covid-19'. Can you imagine setting out to force the population to wear health-destroying masks without doing any assessment of the risks? It is criminal and it is evil, but then how many people targeted in this way, who see their children told to wear them all day at school, have asked for a risk assessment? Billions can't be imposed upon by the few unless the billions allow it. Oh, yes, with just a tinge of irony, 85 percent of all masks made worldwide come from *China*.

Wash your hands in toxic shite

'Covid' rules include the use of toxic sanitisers and again the health consequences of constantly applying toxins to be absorbed through the skin is obvious to any level of Renegade Mind. America's Food and Drug Administration (FDA) said that sanitisers are drugs and issued a warning about 75 dangerous brands which contain

methanol used in antifreeze and can cause death, kidney damage and blindness. The FDA circulated the following warning even for those brands that it claims to be safe:

> Store hand sanitizer out of the reach of pets and children, and children should use it only with adult supervision. Do not drink hand sanitizer. This is particularly important for young children, especially toddlers, who may be attracted by the pleasant smell or brightly colored bottles of hand sanitizer.
>
> Drinking even a small amount of hand sanitizer can cause alcohol poisoning in children. (However, there is no need to be concerned if your children eat with or lick their hands after using hand sanitizer.) During this coronavirus pandemic, poison control centers have had an increase in calls about accidental ingestion of hand sanitizer, so it is important that adults monitor young children's use.
>
> Do not allow pets to swallow hand sanitizer. If you think your pet has eaten something potentially dangerous, call your veterinarian or a pet poison control center right away. Hand sanitizer is flammable and should be stored away from heat and flames. When using hand sanitizer, rub your hands until they feel completely dry before performing activities that may involve heat, sparks, static electricity, or open flames.

There you go, perfectly safe, then, and that's without even a mention of the toxins absorbed through the skin. Come on kids – sanitise your hands everywhere you go. It will save you from the 'virus'. Put all these elements together of the 'Covid' normal and see how much health and psychology is being cumulatively damaged, even devastated, to 'protect your health'. Makes sense, right? They are only imposing these things because they care, right? *Right?*

Submitting to insanity

Psychological reframing of the population goes very deep and is done in many less obvious ways. I hear people say how contradictory and crazy 'Covid' rules are and how they are ever changing. This is explained away by dismissing those involved as idiots. It is a big mistake. The Cult is delighted if its cold calculation is perceived as incompetence and idiocy when it is anything but. Oh, yes, there are idiots within the system – lots of them – but they are *administering* the Cult agenda, mostly unknowingly. They are not deciding and dictating it. The bulwark against tyranny is self-

respect, always has been, always will be. It is self-respect that has broken every tyranny in history. By its very nature self-respect will not bow to oppression and its perpetrators. There is so little self-respect that it's always the few that overturn dictators. Many may eventually follow, but the few with the iron spines (self-respect) kick it off and generate the momentum. The Cult targets self-respect in the knowledge that once this has gone only submission remains. Crazy, contradictory, ever-changing 'Covid' rules are systematically applied by psychologists to delete self-respect. They *want* you to see that the rules make no sense. It is one thing to decide to do something when *you* have made the choice based on evidence and logic. You still retain your self-respect. It is quite another when you can see what you are being told to do is insane, ridiculous and makes no sense, and *yet you still do it*. Your self-respect is extinguished and this has been happening as ever more obviously stupid and nonsensical things have been demanded and the great majority have complied even when they can see they are stupid and nonsensical.

People walk around in face-nappies knowing they are damaging their health and make no difference to a 'virus'. They do it in fear of not doing it. I know it's daft, but I'll do it anyway. When that happens something dies inside of you and submissive reframing has begun. Next there's a need to hide from yourself that you have conceded your self-respect and you convince yourself that you have not really submitted to fear and intimidation. You begin to believe that you are complying with craziness because it's the right thing to do. When first you concede your self-respect of 2+2 = 4 to 2+2 = 5 you *know* you are compromising your self-respect. Gradually to avoid facing that fact you begin to *believe* that 2+2=5. You have been reframed and I have been watching this process happening in the human psyche on an industrial scale. The Cult is working to break your spirit and one of its major tools in that war is humiliation. I read how former American soldier Bradley Manning (later Chelsea Manning after a sex-change) was treated after being jailed for supplying WikiLeaks with documents exposing the enormity of

government and elite mendacity. Manning was isolated in solitary confinement for eight months, put under 24-hour surveillance, forced to hand over clothing before going to bed, and stand naked for every roll call. This is systematic humiliation. The introduction of anal swab 'Covid' tests in China has been done for the same reason to delete self-respect and induce compliant submission. Anal swabs are mandatory for incoming passengers in parts of China and American diplomats have said they were forced to undergo the indignity which would have been calculated humiliation by the Cult-owned Chinese government that has America in its sights.

Government-people: An abusive relationship

Spirit-breaking psychological techniques include giving people hope and apparent respite from tyranny only to take it away again. This happened in the UK during Christmas, 2020, when the psycho-psychologists and their political lackeys announced an easing of restrictions over the holiday only to reimpose them almost immediately on the basis of yet another lie. There is a big psychological difference between getting used to oppression and being given hope of relief only to have that dashed. Psychologists know this and we have seen the technique used repeatedly. Then there is traumatising people before you introduce more extreme regulations that require compliance. A perfect case was the announcement by the dark and sinister Whitty and Vallance in the UK that 'new data' predicted that 4,000 could die every day over the winter of 2020/2021 if we did not lockdown again. I think they call it lying and after traumatising people with that claim out came Jackboot Johnson the next day with new curbs on human freedom. Psychologists know that a frightened and traumatised mind becomes suggestable to submission and behaviour reframing. Underpinning all this has been to make people fearful and suspicious of each other and see themselves as a potential danger to others. In league with deleted self-respect you have the perfect psychological recipe for self-loathing. The relationship between authority and public is now demonstrably the same as that of

subservience to an abusive partner. These are signs of an abusive relationship explained by psychologist Leslie Becker-Phelps:

Psychological and emotional abuse: Undermining a partner's self-worth with verbal attacks, name-calling, and belittling. Humiliating the partner in public, unjustly accusing them of having an affair, or interrogating them about their every behavior. Keeping partner confused or off balance by saying they were just kidding or blaming the partner for 'making' them act this way … Feigning in public that they care while turning against them in private. This leads to victims frequently feeling confused, incompetent, unworthy, hopeless, and chronically self-doubting. [Apply these techniques to how governments have treated the population since New Year, 2020, and the parallels are obvious.]

Physical abuse: The abuser might physically harm their partner in a range of ways, such as grabbing, hitting, punching, or shoving them. They might throw objects at them or harm them with a weapon. [Observe the physical harm imposed by masks, lockdown, and so on.]

Threats and intimidation: One way abusers keep their partners in line is by instilling fear. They might be verbally threatening, or give threatening looks or gestures. Abusers often make it known that they are tracking their partner's every move. They might destroy their partner's possessions, threaten to harm them, or threaten to harm their family members. Not surprisingly, victims of this abuse often feel anxiety, fear, and panic. [No words necessary.]

Isolation: Abusers often limit their partner's activities, forbidding them to talk or interact with friends or family. They might limit access to a car or even turn off their phone. All of this might be done by physically holding them against their will, but is often accomplished through psychological abuse and intimidation. The more isolated a person feels, the fewer resources they have to help gain perspective on their situation and to escape from it. [No words necessary.]

Economic abuse: Abusers often make their partners beholden to them for money by controlling access to funds of any kind. They might prevent their partner from getting a job or withhold access to money they earn from a job. This creates financial dependency that makes leaving the relationship very difficult. [See destruction of livelihoods and the proposed meagre 'guaranteed income' so long as you do whatever you are told.]

Using children: An abuser might disparage their partner's parenting skills, tell their children lies about their partner, threaten to take custody of their children, or threaten to harm their children. These tactics instil fear and often elicit compliance. [See reframed social service mafia and how children are being mercilessly abused by the state over 'Covid' while their parents look on too frightened to do anything.]

A further recurring trait in an abusive relationship is the abused blaming themselves for their abuse and making excuses for the abuser. We have the public blaming each other for lockdown abuse by government and many making excuses for the government while attacking those who challenge the government. How often we have heard authorities say that rules are being imposed or reimposed only because people have refused to 'behave' and follow the rules. We don't want to do it – it's *you*.

Renegade Minds are an antidote to all of these things. They will never concede their self-respect no matter what the circumstances. Even when apparent humiliation is heaped upon them they laugh in its face and reflect back the humiliation on the abuser where it belongs. Renegade Minds will never wear masks they know are only imposed to humiliate, suppress and damage both physically and psychologically. Consequences will take care of themselves and they will never break their spirit or cause them to concede to tyranny. UK newspaper columnist Peter Hitchens was one of the few in the mainstream media to speak out against lockdowns and forced vaccinations. He then announced he had taken the jab. He wanted to see family members abroad and he believed vaccine passports were inevitable even though they had not yet been introduced. Hitchens

has a questioning and critical mind, but not a Renegade one. If he had no amount of pressure would have made him concede. Hitchens excused his action by saying that the battle has been lost. Renegade Minds never accept defeat when freedom is at stake and even if they are the last one standing the self-respect of not submitting to tyranny is more important than any outcome or any consequence.

That's why Renegade Minds are the only minds that ever changed anything worth changing.

CHAPTER EIGHT

'Reframing' insanity

Insanity is relative. It depends on who has who locked in what cage
Ray Bradbury

'Reframing' a mind means simply to change its perception and behaviour. This can be done subconsciously to such an extent that subjects have no idea they have been 'reframed' while to any observer changes in behaviour and attitudes are obvious.

Human society is being reframed on a ginormous scale since the start of 2020 and here we have the reason why psychologists rather than doctors have been calling the shots. Ask most people who have succumbed to 'Covid' reframing if they have changed and most will say 'no'; but they *have* and fundamentally. The Cult's long-game has been preparing for these times since way back and crucial to that has been to prepare both population and officialdom mentally and emotionally. To use the mind-control parlance they had to reframe the population with a mentality that would submit to fascism and reframe those in government and law enforcement to impose fascism or at least go along with it. The result has been the fact-deleted mindlessness of 'Wokeness' and officialdom that has either enthusiastically or unquestioningly imposed global tyranny demanded by reframed politicians on behalf of psychopathic and deeply evil cultists. 'Cognitive reframing' identifies and challenges the way someone sees the world in the form of situations, experiences and emotions and then restructures those perceptions to view the same set of circumstances in a different way. This can have

benefits if the attitudes are personally destructive while on the other side it has the potential for individual and collective mind control which the subject has no idea has even happened.

Cognitive therapy was developed in the 1960s by Aaron T. Beck who was born in Rhode Island in 1921 as the son of Jewish immigrants from the Ukraine. He became interested in the techniques as a treatment for depression. Beck's daughter Judith S. Beck is prominent in the same field and they founded the Beck Institute for Cognitive Behavior Therapy in Philadelphia in 1994. Cognitive reframing, however, began to be used worldwide by those with a very dark agenda. The Cult reframes politicians to change their attitudes and actions until they are completely at odds with what they once appeared to stand for. The same has been happening to government administrators at all levels, law enforcement, military and the human population. Cultists love mind control for two main reasons: It allows them to control what people think, do and say to secure agenda advancement and, by definition, it calms their legendary insecurity and fear of the unexpected. I have studied mind control since the time I travelled America in 1996. I may have been talking to next to no one in terms of an audience in those years, but my goodness did I gather a phenomenal amount of information and knowledge about so many things including the techniques of mind control. I have described this in detail in other books going back to *The Biggest Secret* in 1998. I met a very large number of people recovering from MKUltra and its offshoots and successors and I began to see how these same techniques were being used on the population in general. This was never more obvious than since the 'Covid' hoax began.

Reframing the enforcers

I have observed over the last two decades and more the very clear transformation in the dynamic between the police, officialdom and the public. I tracked this in the books as the relationship mutated from one of serving the public to seeing them as almost the enemy and certainly a lower caste. There has always been a class divide

based on income and always been some psychopathic, corrupt, and big-I-am police officers. This was different. Wholesale change was unfolding in the collective dynamic; it was less about money and far more about position and perceived power. An us-and-them was emerging. Noses were lifted skyward by government administration and law enforcement and their attitude to the public they were *supposed* to be serving changed to one of increasing contempt, superiority and control. The transformation was so clear and widespread that it had to be planned. Collective attitudes and dynamics do not change naturally and organically that quickly on that scale. I then came across an organisation in Britain called Common Purpose created in the late 1980s by Julia Middleton who would work in the office of Deputy Prime Minister John Prescott during the long and disastrous premiership of war criminal Tony Blair. When Blair speaks the Cult is speaking and the man should have been in jail a long time ago. Common Purpose proclaims itself to be one of the biggest 'leadership development' organisations in the world while functioning as a *charity* with all the financial benefits which come from that. It hosts 'leadership development' courses and programmes all over the world and claims to have 'brought together' what it calls 'leaders' from more than 100 countries on six continents. The modus operandi of Common Purpose can be compared with the work of the UK government's reframing network that includes the Behavioural Insights Team 'nudge unit' and 'Covid' reframing specialists at SPI-B. WikiLeaks described Common Purpose long ago as 'a hidden virus in our government and schools' which is unknown to the general public: 'It recruits and trains "leaders" to be loyal to the directives of Common Purpose and the EU, instead of to their own departments, which they then undermine or subvert, the NHS [National Health Service] being an example.' This is a vital point to understand the 'Covid' hoax. The NHS, and its equivalent around the world, has been utterly reframed in terms of administrators and much of the medical personnel with the transformation underpinned by recruitment policies. The outcome has been the criminal and psychopathic behaviour of the

NHS over 'Covid' and we have seen the same in every other major country. WikiLeaks said Common Purpose trainees are 'learning to rule without regard to democracy' and to usher in a police state (current events explained). Common Purpose operated like a 'glue' and had members in the NHS, BBC, police, legal profession, church, many of Britain's 7,000 quangos, local councils, the Civil Service, government ministries and Parliament, and controlled many RDA's (Regional Development Agencies). Here we have one answer for how and why British institutions and their like in other countries have changed so negatively in relation to the public. This further explains how and why the beyond-disgraceful reframed BBC has become a propaganda arm of 'Covid' fascism. They are all part of a network pursuing the same goal.

By 2019 Common Purpose was quoting a figure of 85,000 'leaders' that had attended its programmes. These 'students' of all ages are known as Common Purpose 'graduates' and they consist of government, state and local government officials and administrators, police chiefs and officers, and a whole range of others operating within the national, local and global establishment. Cressida Dick, Commissioner of the London Metropolitan Police, is the Common Purpose graduate who was the 'Gold Commander' that oversaw what can only be described as the murder of Brazilian electrician Jean Charles de Menezes in 2005. He was held down by psychopathic police and shot seven times in the head by a psychopathic lunatic after being mistaken for a terrorist when he was just a bloke going about his day. Dick authorised officers to pursue and keep surveillance on de Menezes and ordered that he be stopped from entering the underground train system. Police psychopaths took her at her word clearly. She was 'disciplined' for this outrage by being *promoted* – eventually to the top of the 'Met' police where she has been a disaster. Many Chief Constables controlling the police in different parts of the UK are and have been Common Purpose graduates. I have heard the 'graduate' network described as a sort of Mafia or secret society operating within the fabric of government at all levels pursuing a collective policy

ingrained at Common Purpose training events. Founder Julia Middleton herself has said:

> Locally and internationally, Common Purpose graduates will be 'lighting small fires' to create change in their organisations and communities … The Common Purpose effect is best illustrated by the many stories of small changes brought about by leaders, who themselves have changed.

A Common Purpose mission statement declared:

> Common Purpose aims to improve the way society works by expanding the vision, decision-making ability and influence of all kinds of leaders. The organisation runs a variety of educational programmes for leaders of all ages, backgrounds and sectors, in order to provide them with the inspirational, information and opportunities they need to change the world.

Yes, but into what? Since 2020 the answer has become clear.

NLP and the Delphi technique

Common Purpose would seem to be a perfect name or would common programming be better? One of the foundation methods of reaching 'consensus' (group think) is by setting the agenda theme and then encouraging, cajoling or pressuring everyone to agree a 'consensus' in line with the core theme promoted by Common Purpose. The methodology involves the 'Delphi technique', or an adaption of it, in which opinions are expressed that are summarised by a 'facilitator or change agent' at each stage. Participants are 'encouraged' to modify their views in the light of what others have said. Stage by stage the former individual opinions are merged into group consensus which just happens to be what Common Purpose wants them to believe. A key part of this is to marginalise anyone refusing to concede to group think and turn the group against them to apply pressure to conform. We are seeing this very technique used on the general population to make 'Covid' group-thinkers hostile to those who have seen through the bullshit. People can be reframed by using perception manipulation methods such as Neuro-Linguistic Programming (NLP) in which you change perception with the use of

carefully constructed language. An NLP website described the technique this way:

> … A method of influencing brain behaviour (the 'neuro' part of the phrase) through the use of language (the 'linguistic' part) and other types of communication to enable a person to 'recode' the way the brain responds to stimuli (that's the 'programming') and manifest new and better behaviours. Neuro-Linguistic Programming often incorporates hypnosis and self-hypnosis to help achieve the change (or 'programming') that is wanted.

British alternative media operation UKColumn has done very detailed research into Common Purpose over a long period. I quoted co-founder and former naval officer Brian Gerrish in my book *Remember Who You Are*, published in 2011, as saying the following years before current times:

> It is interesting that many of the mothers who have had children taken by the State speak of the Social Services people being icily cool, emotionless and, as two ladies said in slightly different words, '… like little robots'. We know that NLP is cumulative, so people can be given small imperceptible doses of NLP in a course here, another in a few months, next year etc. In this way, major changes are accrued in their personality, but the day by day change is almost unnoticeable.

In these and other ways 'graduates' have had their perceptions uniformly reframed and they return to their roles in the institutions of government, law enforcement, legal profession, military, 'education', the UK National Health Service and the whole swathe of the establishment structure to pursue a common agenda preparing for the 'post-industrial', 'post-democratic' society. I say 'preparing' but we are now there. 'Post-industrial' is code for the Great Reset and 'post-democratic' is 'Covid' fascism. UKColumn has spoken to partners of those who have attended Common Purpose 'training'. They have described how personalities and attitudes of 'graduates' changed very noticeably for the worse by the time they had completed the course. They had been 'reframed' and told they are the 'leaders' – the special ones – who know better than the population. There has also been the very demonstrable recruitment of psychopaths and narcissists into government administration at all

levels and law enforcement. If you want psychopathy hire psychopaths and you get a simple cause and effect. If you want administrators, police officers and 'leaders' to perceive the public as lesser beings who don't matter then employ narcissists. These personalities are identified using 'psychometrics' that identifies knowledge, abilities, attitudes and personality traits, mostly through carefully-designed questionnaires and tests. As this policy has passed through the decades we have had power-crazy, power-trippers appointed into law enforcement, security and government administration in preparation for current times and the dynamic between public and law enforcement/officialdom has been transformed. UKColumn's Brian Gerrish said of the narcissistic personality:

> Their love of themselves and power automatically means that they will crush others who get in their way. I received a major piece of the puzzle when a friend pointed out that when they made public officials re-apply for their own jobs several years ago they were also required to do psychometric tests. This was undoubtedly the start of the screening process to get 'their' sort of people in post.

How obvious that has been since 2020 although it was clear what was happening long before if people paid attention to the changing public-establishment dynamic.

Change agents

At the centre of events in 'Covid' Britain is the National Health Service (NHS) which has behaved disgracefully in slavishly following the Cult agenda. The NHS management structure is awash with Common Purpose graduates or 'change agents' working to a common cause. Helen Bevan, a Chief of Service Transformation at the NHS Institute for Innovation and Improvement, co-authored a document called 'Towards a million change agents, a review of the social movements literature: implications for large scale change in the NHS'. The document compared a project management approach to that of change and social movements where 'people change

themselves and each other – peer to peer'. Two definitions given for a 'social movement' were:

A group of people who consciously attempt to build a radically new social order; involves people of a broad range of social backgrounds; and deploys politically confrontational and socially disruptive tactics – Cyrus Zirakzadeh 1997

Collective challenges, based on common purposes and social solidarities, in sustained interaction with elites, opponents, and authorities – Sidney Tarrow 1994

Helen Bevan wrote another NHS document in which she defined 'framing' as 'the process by which leaders construct, articulate and put across their message in a powerful and compelling way in order to win people to their cause and call them to action'. I think I could come up with another definition that would be rather more accurate. The National Health Service and institutions of Britain and the wider world have been taken over by reframed 'change agents' and that includes everything from the United Nations to national governments, local councils and social services which have been kidnapping children from loving parents on an extraordinary and gathering scale on the road to the end of parenthood altogether. Children from loving homes are stolen and kidnapped by the state and put into the 'care' (inversion) of the local authority through council homes, foster parents and forced adoption. At the same time children are allowed to be abused without response while many are under council 'care'. UKColumn highlighted the Common Purpose connection between South Yorkshire Police and Rotherham council officers in the case of the scandal in that area of the sexual exploitation of children to which the authorities turned not one blind eye, but both:

We were alarmed to discover that the Chief Executive, the Strategic Director of Children and Young People's Services, the Manager for the Local Strategic Partnership, the Community Cohesion Manager, the Cabinet Member for Cohesion, the Chief Constable and his predecessor had all attended Leadership training courses provided by the pseudo-charity Common Purpose.

Once 'change agents' have secured positions of hire and fire within any organisation things start to move very quickly. Personnel are then hired and fired on the basis of whether they will work towards the agenda the change agent represents. If they do they are rapidly promoted even though they may be incompetent. Those more qualified and skilled who are pre-Common Purpose 'old school' see their careers stall and even disappear. This has been happening for decades in every institution of state, police, 'health' and social services and all of them have been transformed as a result in their attitudes to their jobs and the public. Medical professions, including nursing, which were once vocations for the caring now employ many cold, callous and couldn't give a shit personality types. The UKColumn investigation concluded:

By blurring the boundaries between people, professions, public and private sectors, responsibility and accountability, Common Purpose encourages 'graduates' to believe that as new selected leaders, they can work together, outside of the established political and social structures, to achieve a paradigm shift or CHANGE – so called 'Leading Beyond Authority'. In doing so, the allegiance of the individual becomes 'reframed' on CP colleagues and their NETWORK.

Reframing the Face-Nappies

Nowhere has this process been more obvious than in the police where recruitment of psychopaths and development of unquestioning mind-controlled group-thinkers have transformed law enforcement into a politically-correct 'Woke' joke and a travesty of what should be public service. Today they wear their face-nappies like good little gofers and enforce 'Covid' rules which are fascism under another name. Alongside the specifically-recruited psychopaths we have software minds incapable of free thought. Brian Gerrish again:

An example is the policeman who would not get on a bike for a press photo because he had not done the cycling proficiency course. Normal people say this is political correctness gone mad. Nothing could be further from the truth. The policeman has been reframed, and in his reality it is perfect common sense not to get on the bike 'because he hasn't done the cycling course'.

Another example of this is where the police would not rescue a boy from a pond until they had taken advice from above on the 'risk assessment'. A normal person would have arrived, perhaps thought of the risk for a moment, and dived in. To the police now 'reframed', they followed 'normal' procedure.

There are shocking cases of reframed ambulance crews doing the same. Sheer unthinking stupidity of London Face-Nappies headed by Common Purpose graduate Cressida Dick can be seen in their behaviour at a vigil in March, 2021, for a murdered woman, Sarah Everard. A police officer had been charged with the crime. Anyone with a brain would have left the vigil alone in the circumstances. Instead they 'manhandled' women to stop them breaking 'Covid rules' to betray classic reframing. Minds in the thrall of perception control have no capacity for seeing a situation on its merits and acting accordingly. 'Rules is rules' is their only mind-set. My father used to say that rules and regulations are for the guidance of the intelligent and the blind obedience of the idiot. Most of the intelligent, decent, coppers have gone leaving only the other kind and a few old school for whom the job must be a daily nightmare. The combination of psychopaths and rule-book software minds has been clearly on public display in the 'Covid' era with automaton robots in uniform imposing fascistic 'Covid' regulations on the population without any personal initiative or judging situations on their merits. There are thousands of examples around the world, but I'll make my point with the infamous Derbyshire police in the English East Midlands – the ones who think pouring dye into beauty spots and using drones to track people walking in the countryside away from anyone is called 'policing'. To them there are rules decreed by the government which they have to enforce and in their bewildered state a group gathering in a closed space and someone walking alone in the countryside are the same thing. It is beyond idiocy and enters the realm of clinical insanity.

Police officers in Derbyshire said they were 'horrified' – *horrified* – to find 15 to 20 'irresponsible' kids playing a football match at a closed leisure centre 'in breach of coronavirus restrictions'. When they saw the police the kids ran away leaving their belongings behind and the reframed men and women of Derbyshire police were seeking to establish their identities with a view to fining their parents. The most natural thing for youngsters to do – kicking a ball about – is turned into a criminal activity and enforced by the moronic software programs of Derbyshire police. You find the same mentality in every country. These barely conscious 'horrified' officers said they had to take action because 'we need to ensure these rules are being followed' and 'it is of the utmost importance that you ensure your children are following the rules and regulations for Covid-19'. Had any of them done ten seconds of research to see if this parroting of their masters' script could be supported by any evidence? Nope. Reframed people don't think – others think for them and that's the whole idea of reframing. I have seen police officers one after the other repeating without question word for word what officialdom tells them just as I have seen great swathes of the public doing the same. Ask either for 'their' opinion and out spews what they have been told to think by the official narrative. Police and public may seem to be in different groups, but their mentality is the same. Most people do whatever they are told in fear not doing so or because they believe what officialdom tells them; almost the entirety of the police do what they are told for the same reason. Ultimately it's the tiny inner core of the global Cult that's telling both what to do.

So Derbyshire police were 'horrified'. Oh, really? Why did they think those kids were playing football? It was to relieve the psychological consequences of lockdown and being denied human contact with their friends and interaction, touch and discourse vital to human psychological health. Being denied this month after month has dismantled the psyche of many children and young people as depression and suicide have exploded. Were Derbyshire police *horrified by that*? Are you kidding? Reframed people don't have those

mental and emotional processes that can see how the impact on the psychological health of youngsters is far more dangerous than any 'virus' even if you take the mendacious official figures to be true. The reframed are told (programmed) how to act and so they do. The Derbyshire Chief Constable in the first period of lockdown when the black dye and drones nonsense was going on was Peter Goodman. He was the man who severed the connection between his force and the Derbyshire Constabulary *Male Voice* Choir when he decided that it was not inclusive enough to allow women to join. The fact it was a male voice choir making a particular sound produced by male voices seemed to elude a guy who terrifyingly ran policing in Derbyshire. He retired weeks after his force was condemned as disgraceful by former Supreme Court Justice Jonathan Sumption for their behaviour over extreme lockdown impositions. Goodman was replaced by his deputy Rachel Swann who was in charge when her officers were 'horrified'. The police statement over the boys committing the hanging-offence of playing football included the line about the youngsters being 'irresponsible in the times we are all living through' missing the point that the real relevance of the 'times we are all living through' is the imposition of fascism enforced by psychopaths and reframed minds of police officers playing such a vital part in establishing the fascist tyranny that their own children and grandchildren will have to live in their entire lives. As a definition of insanity that is hard to beat although it might be run close by imposing masks on people that can have a serious effect on their health while wearing a face nappy all day themselves. Once again public and police do it for the same reason – the authorities tell them to and who are they to have the self-respect to say no?

Wokers in uniform

How reframed do you have to be to arrest a *six-year-old* and take him to court for *picking a flower* while waiting for a bus? Brain dead police and officialdom did just that in North Carolina where criminal proceedings happen regularly for children under nine. Attorney Julie Boyer gave the six-year-old crayons and a colouring book

during the 'flower' hearing while the 'adults' decided his fate. County Chief District Court Judge Jay Corpening asked: 'Should a child that believes in Santa Claus, the Easter Bunny and the tooth fairy be making life-altering decisions?' Well, of course not, but common sense has no meaning when you have a common purpose and a reframed mind. Treating children in this way, and police operating in American schools, is all part of the psychological preparation for children to accept a police state as normal all their adult lives. The same goes for all the cameras and biometric tracking technology in schools. Police training is focused on reframing them as snowflake Wokers and this is happening in the military. Pentagon top brass said that 'training sessions on extremism' were needed for troops who asked why they were so focused on the Capitol Building riot when Black Lives Matter riots were ignored. What's the difference between them some apparently and rightly asked. Actually, there is a difference. Five people died in the Capitol riot, only one through violence, and that was a police officer shooting an unarmed protestor. BLM riots killed at least 25 people and cost billions. Asking the question prompted the psychopaths and reframed minds that run the Pentagon to say that more 'education' (programming) was needed. Troop training is all based on psychological programming to make them fodder for the Cult – 'Military men are just dumb, stupid animals to be used as pawns in foreign policy' as Cult-to-his-DNA former Secretary of State Henry Kissinger famously said. Governments see the police in similar terms and it's time for those among them who can see this to defend the people and stop being enforcers of the Cult agenda upon the people.

The US military, like the country itself, is being targeted for destruction through a long list of Woke impositions. Cult-owned gaga 'President' Biden signed an executive order when he took office to allow taxpayer money to pay for transgender surgery for active military personnel and veterans. Are you a man soldier? No, I'm a LGBTQIA+ with a hint of Skoliosexual and Spectrasexual. Oh, good man. Bad choice of words you bigot. The Pentagon announced in March, 2021, the appointment of the first 'diversity and inclusion

officer' for US Special Forces. Richard Torres-Estrada arrived with the publication of a 'D&I Strategic Plan which will guide the enterprise-wide effort to institutionalize and sustain D&I'. If you think a Special Forces 'Strategic Plan' should have something to do with defending America you haven't been paying attention. Defending Woke is now the military's new role. Torres-Estrada has posted images comparing Donald Trump with Adolf Hitler and we can expect no bias from him as a representative of the supposedly non-political Pentagon. Cable news host Tucker Carlson said: 'The Pentagon is now the Yale faculty lounge but with cruise missiles.' Meanwhile Secretary of Defense Lloyd Austin, a board member of weapons-maker Raytheon with stock and compensation interests in October, 2020, worth $1.4 million, said he was purging the military of the 'enemy within' – anyone who isn't Woke and supports Donald Trump. Austin refers to his targets as 'racist extremists' while in true Woke fashion being himself a racist extremist. Pentagon documents pledge to 'eradicate, eliminate and conquer all forms of racism, sexism and homophobia'. The definitions of these are decided by 'diversity and inclusion committees' peopled by those who see racism, sexism and homophobia in every situation and opinion. Woke (the Cult) is dismantling the US military and purging testosterone as China expands its military and gives its troops 'masculinity training'. How do we think that is going to end when this is all Cult coordinated? The US military, like the British military, is controlled by Woke and spineless top brass who just go along with it out of personal career interests.

'Woke' means fast asleep

Mind control and perception manipulation techniques used on individuals to create group-think have been unleashed on the global population in general. As a result many have no capacity to see the obvious fascist agenda being installed all around them or what 'Covid' is really all about. Their brains are firewalled like a computer system not to process certain concepts, thoughts and realisations that are bad for the Cult. The young are most targeted as the adults they

will be when the whole fascist global state is planned to be fully implemented. They need to be prepared for total compliance to eliminate all pushback from entire generations. The Cult has been pouring billions into taking complete control of 'education' from schools to universities via its operatives and corporations and not least Bill Gates as always. The plan has been to transform 'education' institutions into programming centres for the mentality of 'Woke'. James McConnell, professor of psychology at the University of Michigan, wrote in *Psychology Today* in 1970:

> The day has come when we can combine sensory deprivation with drugs, hypnosis, and astute manipulation of reward and punishment, to gain almost absolute control over an individual's behaviour. It should then be possible to achieve a very rapid and highly effective type of brainwashing that would allow us to make dramatic changes in a person's behaviour and personality ...
>
> … We should reshape society so that we all would be trained from birth to want to do what society wants us to do. We have the techniques to do it... no-one owns his own personality you acquired, and there's no reason to believe you should have the right to refuse to acquire a new personality if your old one is anti-social.

This was the potential for mass brainwashing in 1970 and the mentality there displayed captures the arrogant psychopathy that drives it forward. I emphasise that not all young people have succumbed to Woke programming and those that haven't are incredibly impressive people given that today's young are the most perceptually-targeted generations in history with all the technology now involved. Vast swathes of the young generations, however, have fallen into the spell – and that's what it is – of Woke. The Woke mentality and perceptual program is founded on *inversion* and you will appreciate later why that is so significant. Everything with Woke is inverted and the opposite of what it is claimed to be. Woke was a term used in African-American culture from the 1900s and referred to an awareness of social and racial justice. This is not the meaning of the modern version or 'New Woke' as I call it in *The Answer*. Oh, no, Woke today means something very different no matter how much Wokers may seek to hide that and insist Old Woke and New

Woke are the same. See if you find any 'awareness of social justice' here in the modern variety:

- Woke demands 'inclusivity' while excluding anyone with a different opinion and calls for mass censorship to silence other views.

- Woke claims to stand against oppression when imposing oppression is the foundation of all that it does. It is the driver of political correctness which is nothing more than a Cult invention to manipulate the population to silence itself.

- Woke believes itself to be 'liberal' while pursuing a global society that can only be described as fascist (see 'anti-fascist' fascist Antifa).

- Woke calls for 'social justice' while spreading injustice wherever it goes against the common 'enemy' which can be easily identified as a differing view.

- Woke is supposed to be a metaphor for 'awake' when it is solid-gold asleep and deep in a Cult-induced coma that meets the criteria for 'off with the fairies'.

I state these points as obvious facts if people only care to look. I don't do this with a sense of condemnation. We need to appreciate that the onslaught of perceptual programming on the young has been incessant and merciless. I can understand why so many have been reframed, or, given their youth, framed from the start to see the world as the Cult demands. The Cult has had access to their minds day after day in its 'education' system for their entire formative years. Perception is formed from information received and the Cult-created system is a life-long download of information delivered to elicit a particular perception, thus behaviour. The more this has expanded into still new extremes in recent decades and ever-increasing censorship has deleted other opinions and information why wouldn't that lead to a perceptual reframing on a mass scale? I

have described already cradle-to-grave programming and in more recent times the targeting of young minds from birth to adulthood has entered the stratosphere. This has taken the form of skewing what is 'taught' to fit the Cult agenda and the omnipresent techniques of group-think to isolate non-believers and pressure them into line. There has always been a tendency to follow the herd, but we really are in a new world now in relation to that. We have parents who can see the 'Covid' hoax told by their children not to stop them wearing masks at school, being 'Covid' tested or having the 'vaccine' in fear of the peer-pressure consequences of being different. What is 'peer-pressure' if not pressure to conform to group-think? Renegade Minds never group-think and always retain a set of perceptions that are unique to them. Group-think is always underpinned by consequences for not group-thinking. Abuse now aimed at those refusing DNA-manipulating 'Covid vaccines' are a potent example of this. The biggest pressure to conform comes from the very group which is itself being manipulated. 'I am programmed to be part of a hive mind and so you must be.'

Woke control structures in 'education' now apply to every mainstream organisation. Those at the top of the 'education' hierarchy (the Cult) decide the policy. This is imposed on governments through the Cult network; governments impose it on schools, colleges and universities; their leadership impose the policy on teachers and academics and they impose it on children and students. At any level where there is resistance, perhaps from a teacher or university lecturer, they are targeted by the authorities and often fired. Students themselves regularly demand the dismissal of academics (increasingly few) at odds with the narrative that the students have been programmed to believe in. It is quite a thought that students who are being targeted by the Cult become so consumed by programmed group-think that they launch protests and demand the removal of those who are trying to push back against those targeting the students. Such is the scale of perceptual inversion. We see this with 'Covid' programming as the Cult imposes the rules via psycho-psychologists and governments on

shops, transport companies and businesses which impose them on their staff who impose them on their customers who pressure Pushbackers to conform to the will of the Cult which is in the process of destroying them and their families. Scan all aspects of society and you will see the same sequence every time.

Fact free Woke and hijacking the 'left'

There is no more potent example of this than 'Woke', a mentality only made possible by the deletion of factual evidence by an 'education' system seeking to produce an ever more uniform society. Why would you bother with facts when you don't know any? Deletion of credible history both in volume and type is highly relevant. Orwell said: 'Who controls the past controls the future: who controls the present controls the past.' They who control the perception of the past control the perception of the future and they who control the present control the perception of the past through the writing and deleting of history. Why would you oppose the imposition of Marxism in the name of Wokeism when you don't know that Marxism cost at least 100 million lives in the 20th century alone? Watch videos and read reports in which Woker generations are asked basic historical questions – it's mind-blowing. A survey of 2,000 people found that six percent of millennials (born approximately early1980s to early 2000s) believed the Second World War (1939-1945) broke out with the assassination of President Kennedy (in 1963) and one in ten thought Margaret Thatcher was British Prime Minister at the time. She was in office between 1979 and 1990. We are in a post-fact society. Provable facts are no defence against the fascism of political correctness or Silicon Valley censorship. Facts don't matter anymore as we have witnessed with the 'Covid' hoax. Sacrificing uniqueness to the Woke group-think religion is all you are required to do and that means thinking for yourself is the biggest Woke no, no. All religions are an expression of group-think and censorship and Woke is just another religion with an orthodoxy defended by group-think and censorship. Burned at

the stake becomes burned on Twitter which leads back eventually to burned at the stake as Woke humanity regresses to ages past.

The biggest Woke inversion of all is its creators and funders. I grew up in a traditional left of centre political household on a council estate in Leicester in the 1950s and 60s – you know, the left that challenged the power of wealth-hoarding elites and threats to freedom of speech and opinion. In those days students went on marches defending freedom of speech while today's Wokers march for its deletion. What on earth could have happened? Those very elites (collectively the Cult) that we opposed in my youth and early life have funded into existence the antithesis of that former left and hijacked the 'brand' while inverting everything it ever stood for. We have a mentality that calls itself 'liberal' and 'progressive' while acting like fascists. Cult billionaires and their corporations have funded themselves into control of 'education' to ensure that Woke programming is unceasing throughout the formative years of children and young people and that non-Wokers are isolated (that word again) whether they be students, teachers or college professors. The Cult has funded into existence the now colossal global network of Woke organisations that have spawned and promoted all the 'causes' on the Cult wish-list for global transformation and turned Wokers into demanders of them. Does anyone really think it's a coincidence that the Cult agenda for humanity is a carbon (sorry) copy of the societal transformations desired by Woke?? These are only some of them:

Political correctness: The means by which the Cult deletes all public debates that it knows it cannot win if we had the free-flow of information and evidence.

Human-caused 'climate change': The means by which the Cult seeks to transform society into a globally-controlled dictatorship imposing its will over the fine detail of everyone's lives 'to save the planet' which doesn't actually need saving.

Transgender obsession: Preparing collective perception to accept the 'new human' which would not have genders because it would be created technologically and not through procreation. I'll have much more on this in Human 2.0.

Race obsession: The means by which the Cult seeks to divide and rule the population by triggering racial division through the perception that society is more racist than ever when the opposite is the case. Is it perfect in that regard? No. But to compare today with the racism of apartheid and segregation brought to an end by the civil rights movement in the 1960s is to insult the memory of that movement and inspirations like Martin Luther King. Why is the 'anti-racism' industry (which it is) so dominated by privileged white people?

White supremacy: This is a label used by privileged white people to demonise poor and deprived white people pushing back on tyranny to marginalise and destroy them. White people are being especially targeted as the dominant race by number within Western society which the Cult seeks to transform in its image. If you want to change a society you must weaken and undermine its biggest group and once you have done that by using the other groups you next turn on them to do the same … 'Then they came for the Jews and I was not a Jew so I did nothing.'

Mass migration: The mass movement of people from the Middle East, Africa and Asia into Europe, from the south into the United States and from Asia into Australia are another way the Cult seeks to dilute the racial, cultural and political influence of white people on Western society. White people ask why their governments appear to be working against them while being politically and culturally biased towards incoming cultures. Well, here's your answer. In the same way sexually 'straight' people, men and women, ask why the

authorities are biased against them in favour of other sexualities. The answer is the same – that's the way the Cult wants it to be for very sinister motives.

These are all central parts of the Cult agenda and central parts of the Woke agenda and Woke was created and continues to be funded to an immense degree by Cult billionaires and corporations. If anyone begins to say 'coincidence' the syllables should stick in their throat.

Billionaire 'social justice warriors'

Joe Biden is a 100 percent-owned asset of the Cult and the Wokers' man in the White House whenever he can remember his name and for however long he lasts with his rapidly diminishing cognitive function. Even walking up the steps of an aircraft without falling on his arse would appear to be a challenge. He's not an empty-shell puppet or anything. From the minute Biden took office (or the Cult did) he began his executive orders promoting the Woke wish-list. You will see the Woke agenda imposed ever more severely because it's really the *Cult* agenda. Woke organisations and activist networks spawned by the Cult are funded to the extreme so long as they promote what the Cult wants to happen. Woke is funded to promote 'social justice' by billionaires who become billionaires by destroying social justice. The social justice mantra is only a cover for dismantling social justice and funded by billionaires that couldn't give a damn about social justice. Everything makes sense when you see that. One of Woke's premier funders is Cult billionaire financier George Soros who said: 'I am basically there to make money, I cannot and do not look at the social consequences of what I do.' This is the same Soros who has given more than $32 billion to his Open Society Foundations global Woke network and funded Black Lives Matter, mass immigration into Europe and the United States, transgender activism, climate change activism, political correctness and groups targeting 'white supremacy' in the form of privileged white thugs that dominate Antifa. What a scam it all is and when

you are dealing with the unquestioning fact-free zone of Woke scamming them is child's play. All you need to pull it off in all these organisations are a few in-the-know agents of the Cult and an army of naïve, reframed, uninformed, narcissistic, know-nothings convinced of their own self-righteousness, self-purity and virtue.

Soros and fellow billionaires and billionaire corporations have poured hundreds of millions into Black Lives Matter and connected groups and promoted them to a global audience. None of this is motivated by caring about black people. These are the billionaires that have controlled and exploited a system that leaves millions of black people in abject poverty and deprivation which they do absolutely nothing to address. The same Cult networks funding BLM were behind the *slave trade!* Black Lives Matter hijacked a phrase that few would challenge and they have turned this laudable concept into a political weapon to divide society. You know that BLM is a fraud when it claims that *All* Lives Matter, the most inclusive statement of all, is 'racist'. BLM and its Cult masters don't want to end racism. To them it's a means to an end to control all of humanity never mind the colour, creed, culture or background. What has destroying the nuclear family got to do with ending racism? Nothing – but that is one of the goals of BLM and also happens to be a goal of the Cult as I have been exposing in my books for decades. Stealing children from loving parents and giving schools ever more power to override parents is part of that same agenda. BLM is a Marxist organisation and why would that not be the case when the Cult created Marxism *and* BLM? Patrisse Cullors, a BLM co-founder, said in a 2015 video that she and her fellow organisers, including co-founder Alicia Garza, are 'trained Marxists'. The lady known after marriage as Patrisse Khan-Cullors bought a $1.4 million home in 2021 in one of the whitest areas of California with a black population of just 1.6 per cent and has so far bought *four* high-end homes for a total of $3.2 million. How very Marxist. There must be a bit of spare in the BLM coffers, however, when Cult corporations and billionaires have handed over the best part of $100 million. Many black people can see that Black Lives Matter is not

working for them, but against them, and this is still more confirmation. Black journalist Jason Whitlock, who had his account suspended by Twitter for simply linking to the story about the 'Marxist's' home buying spree, said that BLM leaders are 'making millions of dollars off the backs of these dead black men who they wouldn't spit on if they were on fire and alive'.

Black Lies Matter

Cult assets and agencies came together to promote BLM in the wake of the death of career criminal George Floyd who had been jailed a number of times including for forcing his way into the home of a black woman with others in a raid in which a gun was pointed at her stomach. Floyd was filmed being held in a Minneapolis street in 2020 with the knee of a police officer on his neck and he subsequently died. It was an appalling thing for the officer to do, but the same technique has been used by police on peaceful protestors of lockdown without any outcry from the Woke brigade. As unquestioning supporters of the Cult agenda Wokers have supported lockdown and all the 'Covid' claptrap while attacking anyone standing up to the tyranny imposed in its name. Court documents would later include details of an autopsy on Floyd by County Medical Examiner Dr Andrew Baker who concluded that Floyd had taken a fatal level of the drug fentanyl. None of this mattered to fact-free, question-free, Woke. Floyd's death was followed by worldwide protests against police brutality amid calls to defund the police. Throwing babies out with the bathwater is a Woke speciality. In the wake of the murder of British woman Sarah Everard a Green Party member of the House of Lords, Baroness Jones of Moulescoomb (Nincompoopia would have been better), called for a 6pm curfew for all men. This would be in breach of the Geneva Conventions on war crimes which ban collective punishment, but that would never have crossed the black and white Woke mind of Baroness Nincompoopia who would have been far too convinced of her own self-righteousness to compute such details. Many American cities did defund the police in the face of Floyd riots

and after $15 million was deleted from the police budget in Washington DC under useless Woke mayor Muriel Bowser car-jacking alone rose by 300 percent and within six months the US capital recorded its highest murder rate in 15 years. The same happened in Chicago and other cities in line with the Cult/Soros plan to bring fear to streets and neighbourhoods by reducing the police, releasing violent criminals and not prosecuting crime. This is the mob-rule agenda that I have warned in the books was coming for so long. Shootings in the area of Minneapolis where Floyd was arrested increased by 2,500 percent compared with the year before. Defunding the police over George Floyd has led to a big increase in dead people with many of them black. Police protection for politicians making these decisions stayed the same or increased as you would expect from professional hypocrites. The Cult doesn't actually want to abolish the police. It wants to abolish local control over the police and hand it to federal government as the psychopaths advance the Hunger Games Society. Many George Floyd protests turned into violent riots with black stores and businesses destroyed by fire and looting across America fuelled by Black Lives Matter. Woke doesn't do irony. If you want civil rights you must loot the liquor store and the supermarket and make off with a smart TV. It's the only way.

It's not a race war – it's a class war

Black people are patronised by privileged blacks and whites alike and told they are victims of white supremacy. I find it extraordinary to watch privileged blacks supporting the very system and bloodline networks behind the slave trade and parroting the same Cult-serving manipulative crap of their privileged white, often billionaire, associates. It is indeed not a race war but a class war and colour is just a diversion. Black Senator Cory Booker and black Congresswoman Maxine Waters, more residents of Nincompoopia, personify this. Once you tell people they are victims of someone else you devalue both their own responsibility for their plight and the power they have to impact on their reality and experience. Instead

we have: 'You are only in your situation because of whitey – turn on them and everything will change.' It won't change. Nothing changes in our lives unless *we* change it. Crucial to that is never seeing yourself as a victim and always as the creator of your reality. Life is a simple sequence of choice and consequence. Make different choices and you create different consequences. *You* have to make those choices – not Black Lives Matter, the Woke Mafia and anyone else that seeks to dictate your life. Who are they these Wokers, an emotional and psychological road traffic accident, to tell you what to do? Personal empowerment is the last thing the Cult and its Black Lives Matter want black people or anyone else to have. They claim to be defending the underdog while *creating* and perpetuating the underdog. The Cult's worst nightmare is human unity and if they are going to keep blacks, whites and every other race under economic servitude and control then the focus must be diverted from what they have in common to what they can be manipulated to believe divides them. Blacks have to be told that their poverty and plight is the fault of the white bloke living on the street in the same poverty and with the same plight they are experiencing. The difference is that your plight black people is due to him, a white supremacist with 'white privilege' living on the street. Don't unite as one human family against your mutual oppressors and suppressors – fight the oppressor with the white face who is as financially deprived as you are. The Cult knows that as its 'Covid' agenda moves into still new levels of extremism people are going to respond and it has been spreading the seeds of disunity everywhere to stop a united response to the evil that targets *all of us*.

Racist attacks on 'whiteness' are getting ever more outrageous and especially through the American Democratic Party which has an appalling history for anti-black racism. Barack Obama, Joe Biden, Hillary Clinton and Nancy Pelosi all eulogised about Senator Robert Byrd at his funeral in 2010 after a nearly 60-year career in Congress. Byrd was a brutal Ku Klux Klan racist and a violent abuser of Cathy O'Brien in MKUltra. He said he would never fight in the military 'with a negro by my side' and 'rather I should die a thousand times,

and see Old Glory trampled in the dirt never to rise again, than to see this beloved land of ours become degraded by race mongrels, a throwback to the blackest specimen from the wilds'. Biden called Byrd a 'very close friend and mentor'. These 'Woke' hypocrites are not anti-racist they are anti-poor and anti-people not of their perceived class. Here is an illustration of the scale of anti-white racism to which we have now descended. Seriously Woke and moronic *New York Times* contributor Damon Young described whiteness as a 'virus' that 'like other viruses will not die until there are no bodies left for it to infect'. He went on: '… the only way to stop it is to locate it, isolate it, extract it, and kill it.' Young can say that as a black man with no consequences when a white man saying the same in reverse would be facing a jail sentence. *That's* racism. We had super-Woke numbskull senators Tammy Duckworth and Mazie Hirono saying they would object to future Biden Cabinet appointments if he did not nominate more Asian Americans and Pacific Islanders. Never mind the ability of the candidate what do they look like? Duckworth said: 'I will vote for racial minorities and I will vote for LGBTQ, but anyone else I'm not voting for.' Appointing people on the grounds of race is illegal, but that was not a problem for this ludicrous pair. They were on-message and that's a free pass in any situation.

Critical race racism

White children are told at school they are intrinsically racist as they are taught the divisive 'critical race theory'. This claims that the law and legal institutions are inherently racist and that race is a socially constructed concept used by white people to further their economic and political interests at the expense of people of colour. White is a 'virus' as we've seen. Racial inequality results from 'social, economic, and legal differences that white people create between races to maintain white interests which leads to poverty and criminality in minority communities'. I must tell that to the white guy sleeping on the street. The principal of East Side Community School in New York sent white parents a manifesto that called on

them to become 'white traitors' and advocate for full 'white abolition'. These people are teaching your kids when they urgently need a psychiatrist. The 'school' included a chart with 'eight white identities' that ranged from 'white supremacist' to 'white abolition' and defined the behaviour white people must follow to end 'the regime of whiteness'. Woke blacks and their privileged white associates are acting exactly like the slave owners of old and Ku Klux Klan racists like Robert Byrd. They are too full of their own self-purity to see that, but it's true. Racism is not a body type; it's a state of mind that can manifest through any colour, creed or culture.

Another racial fraud is *'equity'*. Not equality of treatment and opportunity – equity. It's a term spun as equality when it means something very different. Equality in its true sense is a raising up while 'equity' is a race to the bottom. Everyone in the same level of poverty is 'equity'. Keep everyone down – that's equity. The Cult doesn't want anyone in the human family to be empowered and BLM leaders, like all these 'anti-racist' organisations, continue their privileged, pampered existence by perpetuating the perception of gathering racism. When is the last time you heard an 'anti-racist' or 'anti-Semitism' organisation say that acts of racism and discrimination have *fallen?* It's not in the interests of their fund-raising and power to influence and the same goes for the professional soccer anti-racism operation, Kick It Out. Two things confirmed that the Black Lives Matter riots in the summer of 2020 were Cult creations. One was that while anti-lockdown protests were condemned in this same period for 'transmitting 'Covid' the authorities supported mass gatherings of Black Lives Matter supporters. I even saw self-deluding people claiming to be doctors say the two types of protest were not the same. No – the non-existent 'Covid' was in favour of lockdowns and attacked those that protested against them while 'Covid' supported Black Lives Matter and kept well away from its protests. The whole thing was a joke and as lockdown protestors were arrested, often brutally, by reframed Face-Nappies we had the grotesque sight of police officers taking the knee to Black Lives Matter, a Cult-funded Marxist

organisation that supports violent riots and wants to destroy the nuclear family and white people.

He's not white? Shucks!

Woke obsession with race was on display again when ten people were shot dead in Boulder, Colorado, in March, 2021. Cult-owned Woke TV channels like CNN said the shooter appeared to be a white man and Wokers were on Twitter condemning 'violent white men' with the usual mantras. Then the shooter's name was released as Ahmad Al Aliwi Alissa, an anti-Trump Arab-American, and the sigh of disappointment could be heard five miles away. Never mind that ten people were dead and what that meant for their families. Race baiting was all that mattered to these sick Cult-serving people like Barack Obama who exploited the deaths to further divide America on racial grounds which is his job for the Cult. This is the man that 'racist' white Americans made the first black president of the United States and then gave him a second term. Not-very-bright Obama has become filthy rich on the back of that and today appears to have a big influence on the Biden administration. Even so he's still a downtrodden black man and a victim of white supremacy. This disingenuous fraud reveals the contempt he has for black people when he puts on a Deep South Alabama accent whenever he talks to them, no, *at* them.

Another BLM red flag was how the now fully-Woke (fully-Cult) and fully-virtue-signalled professional soccer authorities had their teams taking the knee before every match in support of Marxist Black Lives Matter. Soccer authorities and clubs displayed 'Black Lives Matter' on the players' shirts and flashed the name on electronic billboards around the pitch. Any fans that condemned what is a Freemasonic taking-the-knee ritual were widely condemned as you would expect from the Woke virtue-signallers of professional sport and the now fully-Woke media. We have reverse racism in which you are banned from criticising any race or culture except for white people for whom anything goes – say what you like, no problem. What has this got to do with racial harmony and

equality? We've had black supremacists from Black Lives Matter telling white people to fall to their knees in the street and apologise for their white supremacy. Black supremacists acting like white supremacist slave owners of the past couldn't breach their self-obsessed, race-obsessed sense of self-purity. Joe Biden appointed a race-obsessed black supremacist Kristen Clarke to head the Justice Department Civil Rights Division. Clarke claimed that blacks are endowed with 'greater mental, physical and spiritual abilities' than whites. If anyone reversed that statement they would be vilified. Clarke is on-message so no problem. She's never seen a black-white situation in which the black figure is anything but a virtuous victim and she heads the Civil Rights Division which should treat everyone the same or it isn't civil rights. Another perception of the Renegade Mind: If something or someone is part of the Cult agenda they will be supported by Woke governments and media no matter what. If they're not, they will be condemned and censored. It really is that simple and so racist Clarke prospers despite (make that because of) her racism.

The end of culture

Biden's administration is full of such racial, cultural and economic bias as the Cult requires the human family to be divided into warring factions. We are now seeing racially-segregated graduations and everything, but everything, is defined through the lens of perceived 'racism. We have 'racist' mathematics, 'racist' food and even 'racist' *plants*. World famous Kew Gardens in London said it was changing labels on plants and flowers to tell its pre-'Covid' more than two million visitors a year how racist they are. Kew director Richard Deverell said this was part of an effort to 'move quickly to decolonise collections' after they were approached by one Ajay Chhabra 'an actor with an insight into how sugar cane was linked to slavery'. They are *plants* you idiots. 'Decolonisation' in the Woke manual really means colonisation of society with its mentality and by extension colonisation by the Cult. We are witnessing a new Chinese-style 'Cultural Revolution' so essential to the success of all

Marxist takeovers. Our cultural past and traditions have to be swept away to allow a new culture to be built-back-better. Woke targeting of long-standing Western cultural pillars including historical monuments and cancelling of historical figures is what happened in the Mao revolution in China which 'purged remnants of capitalist and traditional elements from Chinese society' and installed Maoism as the dominant ideology'. For China see the Western world today and for 'dominant ideology' see Woke. Better still see Marxism or Maoism. The 'Covid' hoax has specifically sought to destroy the arts and all elements of Western culture from people meeting in a pub or restaurant to closing theatres, music venues, sports stadiums, places of worship and even banning *singing*. Destruction of Western society is also why criticism of any religion is banned except for Christianity which again is the dominant religion as white is the numerically-dominant race. Christianity may be fading rapidly, but its history and traditions are weaved through the fabric of Western society. Delete the pillars and other structures will follow until the whole thing collapses. I am not a Christian defending that religion when I say that. I have no religion. It's just a fact. To this end Christianity has itself been turned Woke to usher its own downfall and its ranks are awash with 'change agents' – knowing and unknowing – at every level including Pope Francis (*definitely* knowing) and the clueless Archbishop of Canterbury Justin Welby (possibly not, but who can be sure?). Woke seeks to coordinate attacks on Western culture, traditions, and ways of life through 'intersectionality' defined as 'the complex, cumulative way in which the effects of multiple forms of discrimination (such as racism, sexism, and classism) combine, overlap, or intersect especially in the experiences of marginalised individuals or groups'. Wade through the Orwellian Woke-speak and this means coordinating disparate groups in a common cause to overthrow freedom and liberal values.

The entire structure of public institutions has been infested with Woke – government at all levels, political parties, police, military, schools, universities, advertising, media and trade unions. This abomination has been achieved through the Cult web by appointing

Wokers to positions of power and battering non-Wokers into line through intimidation, isolation and threats to their job. Many have been fired in the wake of the empathy-deleted, vicious hostility of 'social justice' Wokers and the desire of gutless, spineless employers to virtue-signal their Wokeness. Corporations are filled with Wokers today, most notably those in Silicon Valley. Ironically at the top they are not Woke at all. They are only exploiting the mentality their Cult masters have created and funded to censor and enslave while the Wokers cheer them on until it's their turn. Thus the Woke 'liberal left' is an inversion of the traditional liberal left. Campaigning for justice on the grounds of power and wealth distribution has been replaced by campaigning for identity politics. The genuine traditional left would never have taken money from today's billionaire abusers of fairness and justice and nor would the billionaires have wanted to fund that genuine left. It would not have been in their interests to do so. The division of opinion in those days was between the haves and have nots. This all changed with Cult manipulated and funded identity politics. The division of opinion today is between Wokers and non-Wokers and not income brackets. Cult corporations and their billionaires may have taken wealth disparity to cataclysmic levels of injustice, but as long as they speak the language of Woke, hand out the dosh to the Woke network and censor the enemy they are 'one of us'. Billionaires who don't give a damn about injustice are laughing at them till their bellies hurt. Wokers are not even close to self-aware enough to see that. The transformed 'left' dynamic means that Wokers who drone on about 'social justice' are funded by billionaires that have destroyed social justice the world over. It's *why* they are billionaires.

The climate con

Nothing encapsulates what I have said more comprehensively than the hoax of human-caused global warming. I have detailed in my books over the years how Cult operatives and organisations were the pump-primers from the start of the climate con. A purpose-built vehicle for this is the Club of Rome established by the Cult in 1968

with the Rockefellers and Rothschilds centrally involved all along. Their gofer frontman Maurice Strong, a Canadian oil millionaire, hosted the Earth Summit in Rio de Janeiro, Brazil, in 1992 where the global 'green movement' really expanded in earnest under the guiding hand of the Cult. The Earth Summit established Agenda 21 through the Cult-created-and-owned United Nations to use the illusion of human-caused climate change to justify the transformation of global society to save the world from climate disaster. It is a No-Problem-Reaction-Solution sold through governments, media, schools and universities as whole generations have been terrified into believing that the world was going to end in their lifetimes unless what old people had inflicted upon them was stopped by a complete restructuring of how everything is done. Chill, kids, it's all a hoax. Such restructuring is precisely what the Cult agenda demands (purely by coincidence of course). Today this has been given the codename of the Great Reset which is only an updated term for Agenda 21 and its associated Agenda 2030. The latter, too, is administered through the UN and was voted into being by the General Assembly in 2015. Both 21 and 2030 seek centralised control of all resources and food right down to the raindrops falling on your own land. These are some of the demands of Agenda 21 established in 1992. See if you recognise this society emerging today:

- End national sovereignty
- State planning and management of all land resources, ecosystems, deserts, forests, mountains, oceans and fresh water; agriculture; rural development; biotechnology; and ensuring *'equity'*
- The state to 'define the role' of business and financial resources
- Abolition of private property
- 'Restructuring' the family unit (see BLM)
- Children raised by the state
- People told what their job will be
- Major restrictions on movement
- Creation of 'human settlement zones'

- Mass resettlement as people are forced to vacate land where they live
- Dumbing down education
- Mass global depopulation in pursuit of all the above

The United Nations was created as a Trojan horse for world government. With the climate con of critical importance to promoting that outcome you would expect the UN to be involved. Oh, it's involved all right. The UN is promoting Agenda 21 and Agenda 2030 justified by 'climate change' while also driving the climate hoax through its Intergovernmental Panel on Climate Change (IPCC), one of the world's most corrupt organisations. The IPCC has been lying ferociously and constantly since the day it opened its doors with the global media hanging unquestioningly on its every mendacious word. The Green movement is entirely Woke and has long lost its original environmental focus since it was co-opted by the Cult. An obsession with 'global warming' has deleted its values and scrambled its head. I experienced a small example of what I mean on a beautiful country walk that I have enjoyed several times a week for many years. The path merged into the fields and forests and you felt at one with the natural world. Then a 'Green' organisation, the Hampshire and Isle of Wight Wildlife Trust, took over part of the land and proceeded to cut down a large number of trees, including mature ones, to install a horrible big, bright steel 'this-is-ours-stay-out' fence that destroyed the whole atmosphere of this beautiful place. No one with a feel for nature would do that. Day after day I walked to the sound of chainsaws and a magnificent mature weeping willow tree that I so admired was cut down at the base of the trunk. When I challenged a Woke young girl in a green shirt (of course) about this vandalism she replied: 'It's a weeping willow – it will grow back.' This is what people are paying for when they donate to the Hampshire and Isle of Wight Wildlife Trust and many other 'green' organisations today. It is not the environmental movement that I knew and instead has become a support-system – as with Extinction Rebellion – for a very dark agenda.

Private jets for climate justice

The Cult-owned, Gates-funded, World Economic Forum and its founder Klaus Schwab were behind the emergence of Greta Thunberg to harness the young behind the climate agenda and she was invited to speak to the world at … the UN. Schwab published a book, *Covid-19: The Great Reset* in 2020 in which he used the 'Covid' hoax and the climate hoax to lay out a new society straight out of Agenda 21 and Agenda 2030. Bill Gates followed in early 2021 when he took time out from destroying the world to produce a book in his name about the way to save it. Gates flies across the world in private jets and admitted that 'I probably have one of the highest greenhouse gas footprints of anyone on the planet … my personal flying alone is gigantic.' He has also bid for the planet's biggest private jet operator. Other climate change saviours who fly in private jets include John Kerry, the US Special Presidential Envoy for Climate, and actor Leonardo DiCaprio, a 'UN Messenger of Peace with special focus on climate change'. These people are so full of bullshit they could corner the market in manure. We mustn't be sceptical, though, because the Gates book, *How to Avoid a Climate Disaster: The Solutions We Have and the Breakthroughs We Need*, is a genuine attempt to protect the world and not an obvious pile of excrement attributed to a mega-psychopath aimed at selling his masters' plans for humanity. The Gates book and the other shite-pile by Klaus Schwab could have been written by the same person and may well have been. Both use 'climate change' and 'Covid' as the excuses for their new society and by coincidence the Cult's World Economic Forum and Bill and Melinda Gates Foundation promote the climate hoax and hosted Event 201 which pre-empted with a 'simulation' the very 'coronavirus' hoax that would be simulated for real on humanity within weeks. The British 'royal' family is promoting the 'Reset' as you would expect through Prince 'climate change caused the war in Syria' Charles and his hapless son Prince William who said that we must 'reset our relationship with nature and our trajectory as a species' to avoid a climate disaster. Amazing how many promotors of the 'Covid' and 'climate change' control

systems are connected to Gates and the World Economic Forum. A 'study' in early 2021 claimed that carbon dioxide emissions must fall by the equivalent of a global lockdown roughly every two years for the next decade to save the planet. The 'study' appeared in the same period that the Schwab mob claimed in a video that lockdowns destroying the lives of billions are good because they make the earth 'quieter' with less 'ambient noise'. They took down the video amid a public backlash for such arrogant, empathy-deleted stupidity You see, however, where they are going with this. Corinne Le Quéré, a professor at the Tyndall Centre for Climate Change Research, University of East Anglia, was lead author of the climate lockdown study, and she writes for … the World Economic Forum. Gates calls in 'his' book for changing 'every aspect of the economy' (long-time Cult agenda) and for humans to eat synthetic 'meat' (predicted in my books) while cows and other farm animals are eliminated. Australian TV host and commentator Alan Jones described what carbon emission targets would mean for farm animals in Australia alone if emissions were reduced as demanded by 35 percent by 2030 and zero by 2050:

> Well, let's take agriculture, the total emissions from agriculture are about 75 million tonnes of carbon dioxide, equivalent. Now reduce that by 35 percent and you have to come down to 50 million tonnes, I've done the maths. So if you take for example 1.5 million cows, you're going to have to reduce the herd by 525,000 [by] 2030, nine years, that's 58,000 cows a year. The beef herd's 30 million, reduce that by 35 percent, that's 10.5 million, which means 1.2 million cattle have to go every year between now and 2030. This is insanity!
>
> There are 75 million sheep. Reduce that by 35 percent, that's 26 million sheep, that's almost 3 million a year. So under the Paris Agreement over 30 million beasts. dairy cows, cattle, pigs and sheep would go. More than 8,000 every minute of every hour for the next decade, do these people know what they're talking about?

Clearly they don't at the level of campaigners, politicians and administrators. The Cult *does* know; that's the outcome it wants. We are faced with not just a war on humanity. Animals and the natural world are being targeted and I have been saying since the 'Covid' hoax began that the plan eventually was to claim that the 'deadly virus' is able to jump from animals, including farm animals and

domestic pets, to humans. Just before this book went into production came this story: 'Russia registers world's first Covid-19 vaccine for cats & dogs as makers of Sputnik V warn pets & farm animals could spread virus'. The report said 'top scientists warned that the deadly pathogen could soon begin spreading through homes and farms' and 'the next stage is the infection of farm and domestic animals'. Know the outcome and you'll see the journey. Think what that would mean for animals and keep your eye on a term called zoonosis or zoonotic diseases which transmit between animals and humans. The Cult wants to break the connection between animals and people as it does between people and people. Farm animals fit with the Cult agenda to transform food from natural to synthetic.

The gas of life is killing us

There can be few greater examples of Cult inversion than the condemnation of carbon dioxide as a dangerous pollutant when it is the gas of life. Without it the natural world would be dead and so we would all be dead. We breathe in oxygen and breathe out carbon dioxide while plants produce oxygen and absorb carbon dioxide. It is a perfect symbiotic relationship that the Cult wants to dismantle for reasons I will come to in the final two chapters. Gates, Schwab, other Cult operatives and mindless repeaters, want the world to be 'carbon neutral' by at least 2050 and the earlier the better. 'Zero carbon' is the cry echoed by lunatics calling for 'Zero Covid' when we already have it. These carbon emission targets will deindustrialise the world in accordance with Cult plans – the post-industrial, post-democratic society – and with so-called renewables like solar and wind not coming even close to meeting human energy needs blackouts and cold are inevitable. Texans got the picture in the winter of 2021 when a snow storm stopped wind turbines and solar panels from working and the lights went down along with water which relies on electricity for its supply system. Gates wants everything to be powered by electricity to ensure that his masters have the kill switch to stop all human activity, movement, cooking, water and warmth any time they like. The climate lie is so

stupendously inverted that it claims we must urgently reduce carbon dioxide when we *don't have enough*.

Co2 in the atmosphere is a little above 400 parts per million when the optimum for plant growth is 2,000 ppm and when it falls anywhere near 150 ppm the natural world starts to die and so do we. It fell to as low as 280 ppm in an 1880 measurement in Hawaii and rose to 413 ppm in 2019 with industrialisation which is why the planet has become *greener* in the industrial period. How insane then that psychopathic madman Gates is not satisfied only with blocking the rise of Co2. He's funding technology to suck it out of the atmosphere. The reason why will become clear. The industrial era is not destroying the world through Co2 and has instead turned around a potentially disastrous ongoing fall in Co2. Greenpeace co-founder and scientist Patrick Moore walked away from Greenpeace in 1986 and has exposed the green movement for fear-mongering and lies. He said that 500 million years ago there was *17 times* more Co2 in the atmosphere than we have today and levels have been falling for hundreds of millions of years. In the last 150 million years Co2 levels in Earth's atmosphere had reduced by *90 percent*. Moore said that by the time humanity began to unlock carbon dioxide from fossil fuels we were at '38 seconds to midnight' and in that sense: 'Humans are [the Earth's] salvation.' Moore made the point that only half the Co2 emitted by fossil fuels stays in the atmosphere and we should remember that all pollution pouring from chimneys that we are told is carbon dioxide is in fact nothing of the kind. It's pollution. Carbon dioxide is an invisible gas.

William Happer, Professor of Physics at Princeton University and long-time government adviser on climate, has emphasised the Co2 deficiency for maximum growth and food production. Greenhouse growers don't add carbon dioxide for a bit of fun. He said that most of the warming in the last 100 years, after the earth emerged from the super-cold period of the 'Little Ice Age' into a natural warming cycle, was over by 1940. Happer said that a peak year for warming in 1988 can be explained by a 'monster El Nino' which is a natural and cyclical warming of the Pacific that has nothing to do with 'climate

change'. He said the effect of Co2 could be compared to painting a wall with red paint in that once two or three coats have been applied it didn't matter how much more you slapped on because the wall will not get much redder. Almost all the effect of the rise in Co2 has already happened, he said, and the volume in the atmosphere would now have to *double* to increase temperature by a single degree. Climate hoaxers know this and they have invented the most ridiculously complicated series of 'feedback' loops to try to overcome this rather devastating fact. You hear puppet Greta going on cluelessly about feedback loops and this is why.

The Sun affects temperature? No you *climate denier*

Some other nonsense to contemplate: Climate graphs show that rises in temperature do not follow rises in Co2 – *it's the other way round* with a lag between the two of some 800 years. If we go back 800 years from present time we hit the Medieval Warm Period when temperatures were higher than now without any industrialisation and this was followed by the Little Ice Age when temperatures plummeted. The world was still emerging from these centuries of serious cold when many climate records began which makes the ever-repeated line of the 'hottest year since records began' meaningless when you are not comparing like with like. The coldest period of the Little Ice Age corresponded with the lowest period of sunspot activity when the Sun was at its least active. Proper scientists will not be at all surprised by this when it confirms the obvious fact that earth temperature is affected by the scale of Sun activity and the energetic power that it subsequently emits; but when is the last time you heard a climate hoaxer talking about the Sun as a source of earth temperature?? Everything has to be focussed on Co2 which makes up just 0.117 percent of so-called greenhouse gases and only a fraction of even that is generated by human activity. The rest is natural. More than *90 percent* of those greenhouse gases are water vapour and clouds (Fig 9). Ban moisture I say. Have you noticed that the climate hoaxers no longer use the polar bear as their promotion image? That's because far from becoming extinct polar

bear communities are stable or thriving. Joe Bastardi, American meteorologist, weather forecaster and outspoken critic of the climate lie, documents in his book *The Climate Chronicles* how weather patterns and events claimed to be evidence of climate change have been happening since long before industrialisation: 'What happened before naturally is happening again, as is to be expected given the cyclical nature of the climate due to the design of the planet.' If you read the detailed background to the climate hoax in my other books you will shake your head and wonder how anyone could believe the crap which has spawned a multi-trillion dollar industry based on absolute garbage (see HIV causes AIDs and Sars-Cov-2 causes 'Covid-19'). Climate and 'Covid' have much in common given they have the same source. They both have the contradictory *everything* factor in which everything is explained by reference to them. It's hot – 'it's climate change'. It's cold – 'it's climate change'. I got a sniffle – 'it's Covid'. I haven't got a sniffle – 'it's Covid'. Not having a sniffle has to be a symptom of 'Covid'. Everything is and not having a sniffle is especially dangerous if you are a slow walker. For sheer audacity I offer you a Cambridge University 'study' that actually linked 'Covid' to 'climate change'. It had to happen eventually. They concluded that climate change played a role in 'Covid-19' spreading from animals to humans because … wait for it … I kid you not … *the two groups were forced closer together as populations grow*. Er, that's it. The whole foundation on which this depended was that 'Bats are the likely zoonotic origin of SARS-CoV-1 and SARS-CoV-2'. Well, they are not. They are nothing to do with it. Apart from bats not being the origin and therefore 'climate change' effects on bats being irrelevant I am in awe of their academic insight. Where would we be without them? Not where we are that's for sure.

Figure 9: The idea that the gas of life is disastrously changing the climate is an insult to brain cell activity.

One other point about the weather is that climate modification is now well advanced and not every major weather event is natural – or earthquake come to that. I cover this subject at some length in other books. China is openly planning a rapid expansion of its weather modification programme which includes changing the climate in an area more than one and a half times the size of India. China used weather manipulation to ensure clear skies during the 2008 Olympics in Beijing. I have quoted from US military documents detailing how to employ weather manipulation as a weapon of war and they did that in the 1960s and 70s during the conflict in Vietnam with Operation Popeye manipulating monsoon rains for military purposes. Why would there be international treaties on weather modification if it wasn't possible? Of course it is. Weather is energetic information and it can be changed.

How was the climate hoax pulled off? See 'Covid'

If you can get billions to believe in a 'virus' that doesn't exist you can get them to believe in human-caused climate change that doesn't exist. Both are being used by the Cult to transform global society in the way it has long planned. Both hoaxes have been achieved in pretty much the same way. First you declare a lie is a fact. There's a

'virus' you call SARS-Cov-2 or humans are warming the planet with their behaviour. Next this becomes, via Cult networks, the foundation of government, academic and science policy and belief. Those who parrot the mantra are given big grants to produce research that confirms the narrative is true and ever more 'symptoms' are added to make the 'virus'/'climate change' sound even more scary. Scientists and researchers who challenge the narrative have their grants withdrawn and their careers destroyed. The media promote the lie as the unquestionable truth and censor those with an alternative view or evidence. A great percentage of the population believe what they are told as the lie becomes an everybody-knows-that and the believing-masses turn on those with a mind of their own. The technique has been used endlessly throughout human history. Wokers are the biggest promotors of the climate lie *and* 'Covid' fascism because their minds are owned by the Cult; their sense of self-righteous self-purity knows no bounds; and they exist in a bubble of reality in which facts are irrelevant and only get in the way of looking without seeing.

Running through all of this like veins in a blue cheese is control of information, which means control of perception, which means control of behaviour, which collectively means control of human society. The Cult owns the global media and Silicon Valley fascists for the simple reason that it *has* to. Without control of information it can't control perception and through that human society. Examine every facet of the Cult agenda and you will see that anything supporting its introduction is never censored while anything pushing back is always censored. I say again: Psychopaths that know why they are doing this must go before Nuremberg trials and those that follow their orders must trot along behind them into the same dock. 'I was just following orders' didn't work the first time and it must not work now. Nuremberg trials must be held all over the world before public juries for politicians, government officials, police, compliant doctors, scientists and virologists, and all Cult operatives such as Gates, Tedros, Fauci, Vallance, Whitty, Ferguson, Zuckerberg, Wojcicki, Brin, Page, Dorsey, the whole damn lot of

them – including, no *especially*, the psychopath psychologists. Without them and the brainless, gutless excuses for journalists that have repeated their lies, none of this could be happening. Nobody can be allowed to escape justice for the psychological and economic Armageddon they are all responsible for visiting upon the human race.

As for the compliant, unquestioning, swathes of humanity, and the self-obsessed, all-knowing ignorance of the Wokers … don't start me. God help their kids. God help their grandkids. God *help them*.

CHAPTER NINE

We must have it? So what is it?

Well I won't back down. No, I won't back down. You can stand me up at the Gates of Hell. But I won't back down
Tom Petty

I will now focus on the genetically-manipulating 'Covid vaccines' which do not meet this official definition of a vaccine by the US Centers for Disease Control (CDC): 'A product that stimulates a person's immune system to produce immunity to a specific disease, protecting the person from that disease.' On that basis 'Covid vaccines' are not a vaccine in that the makers don't even claim they stop infection or transmission.

They are instead part of a multi-levelled conspiracy to change the nature of the human body and what it means to be 'human' and to depopulate an enormous swathe of humanity. What I shall call Human 1.0 is on the cusp of becoming Human 2.0 and for very sinister reasons. Before I get to the 'Covid vaccine' in detail here's some background to vaccines in general. Government regulators do not test vaccines – the makers do – and the makers control which data is revealed and which isn't. Children in America are given 50 vaccine doses by age six and 69 by age 19 and the effect of the whole combined schedule has never been tested. Autoimmune diseases when the immune system attacks its own body have soared in the mass vaccine era and so has disease in general in children and the young. Why wouldn't this be the case when vaccines target the *immune system?* The US government gave Big Pharma drug

companies immunity from prosecution for vaccine death and injury in the 1986 National Childhood Vaccine Injury Act (NCVIA) and since then the government (taxpayer) has been funding compensation for the consequences of Big Pharma vaccines. The criminal and satanic drug giants can't lose and the vaccine schedule has increased dramatically since 1986 for this reason. There is no incentive to make vaccines safe and a big incentive to make money by introducing ever more. Even against a ridiculously high bar to prove vaccine liability, and with the government controlling the hearing in which it is being challenged for compensation, the vaccine court has so far paid out more than $4 billion. These are the vaccines we are told are safe and psychopaths like Zuckerberg censor posts saying otherwise. The immunity law was even justified by a ruling that vaccines by their nature were 'unavoidably unsafe'.

Check out the ingredients of vaccines and you will be shocked if you are new to this. *They put that in children's bodies?? What??* Try aluminium, a brain toxin connected to dementia, aborted foetal tissue and formaldehyde which is used to embalm corpses. World-renowned aluminium expert Christopher Exley had his research into the health effect of aluminium in vaccines shut down by Keele University in the UK when it began taking funding from the Bill and Melinda Gates Foundation. Research when diseases 'eradicated' by vaccines began to decline and you will find the fall began long *before* the vaccine was introduced. Sometimes the fall even plateaued after the vaccine. Diseases like scarlet fever for which there was no vaccine declined in the same way because of environmental and other factors. A perfect case in point is the polio vaccine. Polio began when lead arsenate was first sprayed as an insecticide and residues remained in food products. Spraying started in 1892 and the first US polio epidemic came in Vermont in 1894. The simple answer was to stop spraying, but Rockefeller-created Big Pharma had a better idea. Polio was decreed to be caused by the *poliovirus* which 'spreads from person to person and can infect a person's spinal cord'. Lead arsenate was replaced by the lethal DDT which had the same effect of causing paralysis by damaging the brain and central nervous

system. Polio plummeted when DDT was reduced and then banned, but the vaccine is still given the credit for something it didn't do. Today by far the biggest cause of polio is the vaccines promoted by Bill Gates. Vaccine justice campaigner Robert Kennedy Jr, son of assassinated (by the Cult) US Attorney General Robert Kennedy, wrote:

> In 2017, the World Health Organization (WHO) reluctantly admitted that the global explosion in polio is predominantly vaccine strain. The most frightening epidemics in Congo, Afghanistan, and the Philippines, are all linked to vaccines. In fact, by 2018, 70% of global polio cases were vaccine strain.

Vaccines make fortunes for Cult-owned Gates and Big Pharma while undermining the health and immune systems of the population. We had a glimpse of the mentality behind the Big Pharma cartel with a report on WION (World is One News), an international English language TV station based in India, which exposed the extraordinary behaviour of US drug company Pfizer over its 'Covid vaccine'. The WION report told how Pfizer had made fantastic demands of Argentina, Brazil and other countries in return for its 'vaccine'. These included immunity from prosecution, even for Pfizer negligence, government insurance to protect Pfizer from law suits and handing over as collateral sovereign assets of the country to include Argentina's bank reserves, military bases and embassy buildings. Pfizer demanded the same of Brazil in the form of waiving sovereignty of its assets abroad; exempting Pfizer from Brazilian laws; and giving Pfizer immunity from all civil liability. This is a 'vaccine' developed with government funding. Big Pharma is evil incarnate as a creation of the Cult and all must be handed tickets to Nuremberg.

Phantom 'vaccine' for a phantom 'disease'

I'll expose the 'Covid vaccine' fraud and then go on to the wider background of why the Cult has set out to 'vaccinate' every man, woman and child on the planet for an alleged 'new disease' with a survival rate of 99.77 percent (or more) even by the grotesquely-

manipulated figures of the World Health Organization and Johns Hopkins University. The 'infection' to 'death' ratio is 0.23 to 0.15 percent according to Stanford epidemiologist Dr John Ioannidis and while estimates vary the danger remains tiny. I say that if the truth be told the fake infection to fake death ratio is zero. Never mind all the evidence I have presented here and in *The Answer* that there is no 'virus' let us just focus for a moment on that death-rate figure of say 0.23 percent. The figure includes all those worldwide who have tested positive with a test not testing for the 'virus' and then died within 28 days or even longer of any other cause – *any other cause*. Now subtract all those illusory 'Covid' deaths on the global data sheets from the 0.23 percent. What do you think you would be left with? *Zero*. A vaccination has never been successfully developed for a so-called coronavirus. They have all failed at the animal testing stage when they caused hypersensitivity to what they were claiming to protect against and made the impact of a disease far worse. Cult-owned vaccine corporations got around that problem this time by bypassing animal trials, going straight to humans and making the length of the 'trials' before the public rollout as short as they could get away with. Normally it takes five to ten years or more to develop vaccines that still cause demonstrable harm to many people and that's without including the long-term effects that are never officially connected to the vaccination. 'Covid' non-vaccines have been officially produced and approved in a matter of months from a standing start and part of the reason is that (a) they were developed before the 'Covid' hoax began and (b) they are based on computer programs and not natural sources. Official non-trials were so short that government agencies gave *emergency,* not full, approval. 'Trials' were not even completed and full approval cannot be secured until they are. Public 'Covid vaccination' is actually a *continuation of the trial*. Drug company 'trials' are not scheduled to end until 2023 by which time a lot of people are going to be dead. Data on which government agencies gave this emergency approval was supplied by the Big Pharma corporations themselves in the form of Pfizer/BioNTech, AstraZeneca, Moderna, Johnson & Johnson, and

others, and this is the case with all vaccines. By its very nature *emergency* approval means drug companies do not have to prove that the 'vaccine' is 'safe and effective'. How could they with trials way short of complete? Government regulators only have to *believe* that they *could* be safe and effective. It is criminal manipulation to get products in circulation with no testing worth the name. Agencies giving that approval are infested with Big Pharma-connected place-people and they act in the interests of Big Pharma (the Cult) and not the public about whom they do not give a damn.

More human lab rats

'Covid vaccines' produced in record time by Pfizer/BioNTech and Moderna employ a technique *never approved before for use on humans*. They are known as mRNA 'vaccines' and inject a synthetic version of 'viral' mRNA or 'messenger RNA'. The key is in the term 'messenger'. The body works, or doesn't, on the basis of information messaging. Communications are constantly passing between and within the genetic system and the brain. Change those messages and you change the state of the body and even its very nature and you can change psychology and behaviour by the way the brain processes information. I think you are going to see significant changes in personality and perception of many people who have had the 'Covid vaccine' synthetic potions. Insider Aldous Huxley predicted the following in 1961 and mRNA 'vaccines' can be included in the term 'pharmacological methods':

> There will be, in the next generation or so, a pharmacological method of making people love their servitude, and producing dictatorship without tears, so to speak, producing a kind of painless concentration camp for entire societies, so that people will in fact have their own liberties taken away from them, but rather enjoy it, because they will be distracted from any desire to rebel by propaganda or brainwashing, or brainwashing enhanced by pharmacological methods. And this seems to be the final revolution.

Apologists claim that mRNA synthetic 'vaccines' don't change the DNA genetic blueprint because RNA does not affect DNA only the other way round. This is so disingenuous. A process called 'reverse

transcription' can convert RNA into DNA and be integrated into DNA in the cell nucleus. This was highlighted in December, 2020, by scientists at Harvard and Massachusetts Institute of Technology (MIT). Geneticists report that more than 40 percent of mammalian genomes results from reverse transcription. On the most basic level if messaging changes then that sequence must lead to changes in DNA which is receiving and transmitting those communications. How can introducing synthetic material into cells not change the cells where DNA is located? The process is known as transfection which is defined as 'a technique to insert foreign nucleic acid (DNA or RNA) into a cell, typically with the intention of altering the properties of the cell'. Researchers at the Sloan Kettering Institute in New York found that changes in messenger RNA can deactivate tumour-suppressing proteins and thereby promote cancer. This is what happens when you mess with messaging. 'Covid vaccine' maker Moderna was founded in 2010 by Canadian stem cell biologist Derrick J. Rossi after his breakthrough discovery in the field of transforming and reprogramming stem cells. These are neutral cells that can be programmed to become any cell including sperm cells. Moderna was therefore founded on the principle of genetic manipulation and has never produced any vaccine or drug before its genetically-manipulating synthetic 'Covid' shite. Look at the name – Mode-RNA or Modify-RNA. Another important point is that the US Supreme Court has ruled that genetically-modified DNA, or complementary DNA (cDNA) synthesized in the laboratory from messenger RNA, can be patented and owned. These psychopaths are doing this to the human body.

Cells replicate synthetic mRNA in the 'Covid vaccines' and in theory the body is tricked into making antigens which trigger antibodies to target the 'virus spike proteins' which as Dr Tom Cowan said have *never been seen*. Cut the crap and these 'vaccines' deliver *self-replicating* synthetic material to the cells with the effect of changing human DNA. The more of them you have the more that process is compounded while synthetic material is all the time self-replicating. 'Vaccine'-maker Moderna describes mRNA as 'like

software for the cell' and so they are messing with the body's software. What happens when you change the software in a computer? Everything changes. For this reason the Cult is preparing a production line of mRNA 'Covid vaccines' and a long list of excuses to use them as with all the 'variants' of a 'virus' never shown to exist. The plan is further to transfer the mRNA technique to other vaccines mostly given to children and young people. The cumulative consequences will be a transformation of human DNA through a constant infusion of synthetic genetic material which will kill many and change the rest. Now consider that governments that have given emergency approval for a vaccine that's not a vaccine; never been approved for humans before; had no testing worth the name; and the makers have been given immunity from prosecution for any deaths or adverse effects suffered by the public. The UK government awarded *permanent legal indemnity* to itself and its employees for harm done when a patient is being treated for 'Covid-19' or 'suspected Covid-19'. That is quite a thought when these are possible 'side-effects' from the 'vaccine' (they are not 'side', they are effects) listed by the US Food and Drug Administration:

Guillain-Barre syndrome; acute disseminated encephalomyelitis; transverse myelitis; encephalitis; myelitis; encephalomyelitis; meningoencephalitis; meningitis; encephalopathy; convulsions; seizures; stroke; narcolepsy; cataplexy; anaphylaxis; acute myocardial infarction (heart attack); myocarditis; pericarditis; autoimmune disease; death; implications for pregnancy, and birth outcomes; other acute demyelinating diseases; non anaphylactic allergy reactions; thrombocytopenia ; disseminated intravascular coagulation; venous thromboembolism; arthritis; arthralgia; joint pain; Kawasaki disease; multisystem inflammatory syndrome in children; vaccine enhanced disease. The latter is the way the 'vaccine' has the potential to make diseases far worse than they would otherwise be.

UK doctor and freedom campaigner Vernon Coleman described the conditions in this list as 'all unpleasant, most of them very serious, and you can't get more serious than death'. The thought that anyone at all has had the 'vaccine' in these circumstances is testament to the potential that humanity has for clueless, unquestioning, stupidity and for many that programmed stupidity has already been terminal.

An insider speaks

Dr Michael Yeadon is a former Vice President, head of research and Chief Scientific Adviser at vaccine giant Pfizer. Yeadon worked on the inside of Big Pharma, but that did not stop him becoming a vocal critic of 'Covid vaccines' and their potential for multiple harms, including infertility in women. By the spring of 2021 he went much further and even used the no, no, term 'conspiracy'. When you begin to see what is going on it is impossible not to do so. Yeadon spoke out in an interview with freedom campaigner James Delingpole and I mentioned earlier how he said that no one had samples of 'the virus'. He explained that the mRNA technique originated in the anti-cancer field and ways to turn on and off certain genes which could be advantageous if you wanted to stop cancer growing out of control. 'That's the origin of them. They are a very unusual application, really.' Yeadon said that treating a cancer patient with an aggressive procedure might be understandable if the alternative was dying, but it was quite another thing to use the same technique as a public health measure. Most people involved wouldn't catch the infectious agent you were vaccinating against and if they did they probably wouldn't die:

> If you are really using it as a public health measure you really want to as close as you can get to zero sides-effects … I find it odd that they chose techniques that were really cutting their teeth in the field of oncology and I'm worried that in using gene-based vaccines that have to be injected in the body and spread around the body, get taken up into some cells, and the regulators haven't quite told us which cells they get taken up into … you are going to be generating a wide range of responses … with multiple steps each of which could go well or badly.

I doubt the Cult intends it to go well. Yeadon said that you can put any gene you like into the body through the 'vaccine'. 'You can certainly give them a gene that would do them some harm if you wanted.' I was intrigued when he said that when used in the cancer field the technique could turn genes on and off. I explore this process in *The Answer* and with different genes having different functions you could create mayhem – physically and psychologically – if you turned the wrong ones on and the right ones off. I read reports of an experiment by researchers at the University of Washington's school of computer science and engineering in which they encoded DNA to infect computers. The body is itself a biological computer and if human DNA can inflict damage on a computer why can't the computer via synthetic material mess with the human body? It can. The Washington research team said it was possible to insert malicious malware into 'physical DNA strands' and corrupt the computer system of a gene sequencing machine as it 'reads gene letters and stores them as binary digits 0 and 1'. They concluded that hackers could one day use blood or spit samples to access computer systems and obtain sensitive data from police forensics labs or infect genome files. It is at this level of digital interaction that synthetic 'vaccines' need to be seen to get the full picture and that will become very clear later on. Michael Yeadon said it made no sense to give the 'vaccine' to younger people who were in no danger from the 'virus'. What was the benefit? It was all downside with potential effects:

> The fact that my government in what I thought was a civilised, rational country, is raining [the 'vaccine'] on people in their 30s and 40s, even my children in their 20s, they're getting letters and phone calls, I know this is not right and any of you doctors who are vaccinating you know it's not right, too. They are not at risk. They are not at risk from the disease, so you are now hoping that the side-effects are so rare that you get away with it. You don't give new technology … that you don't understand to 100 percent of the population.

Blood clot problems with the AstraZeneca 'vaccine' have been affecting younger people to emphasise the downside risks with no benefit. AstraZeneca's version, produced with Oxford University, does not use mRNA, but still gets its toxic cocktail inside cells where

it targets DNA. The Johnson & Johnson 'vaccine' which uses a similar technique has also produced blood clot effects to such an extent that the United States paused its use at one point. They are all 'gene therapy' (cell modification) procedures and not 'vaccines'. The truth is that once the content of these injections enter cells we have no idea what the effect will be. People can speculate and some can give very educated opinions and that's good. In the end, though, only the makers know what their potions are designed to do and even they won't know every last consequence. Michael Yeadon was scathing about doctors doing what they knew to be wrong. 'Everyone's mute', he said. Doctors in the NHS must know this was not right, coming into work and injecting people. 'I don't know how they sleep at night. I know I couldn't do it. I know that if I were in that position I'd have to quit.' He said he knew enough about toxicology to know this was not a good risk-benefit. Yeadon had spoken to seven or eight university professors and all except two would not speak out publicly. Their universities had a policy that no one said anything that countered the government and its medical advisors. They were afraid of losing their government grants. This is how intimidation has been used to silence the truth at every level of the system. I say silence, but these people could still speak out if they made that choice. Yeadon called them 'moral cowards' – 'This is about your children and grandchildren's lives and you have just buggered off and left it.'

'Variant' nonsense

Some of his most powerful comments related to the alleged 'variants' being used to instil more fear, justify more lockdowns, and introduce more 'vaccines'. He said government claims about 'variants' were nonsense. He had checked the alleged variant 'codes' and they were 99.7 percent identical to the 'original'. This was the human identity difference equivalent to putting a baseball cap on and off or wearing it the other way round. A 0.3 percent difference would make it impossible for that 'variant' to escape immunity from the 'original'. This made no sense of having new 'vaccines' for

'variants'. He said there would have to be at least a *30 percent* difference for that to be justified and even then he believed the immune system would still recognise what it was. Gates-funded 'variant modeller' and 'vaccine'-pusher John Edmunds might care to comment. Yeadon said drug companies were making new versions of the 'vaccine' as a 'top up' for 'variants'. Worse than that, he said, the 'regulators' around the world like the MHRA in the UK had got together and agreed that because 'vaccines' for 'variants' were so similar to the first 'vaccines' *they did not have to do safety studies*. How transparently sinister that is. This is when Yeadon said: 'There is a conspiracy here.' There was no need for another vaccine for 'variants' and yet we were told that there was and the country had shut its borders because of them. 'They are going into hundreds of millions of arms without passing 'go' or any regulator. Why did they do that? Why did they pick this method of making the vaccine?'

The reason had to be something bigger than that it seemed and 'it's not protection against the virus'. It's was a far bigger project that meant politicians and advisers were willing to do things and not do things that knowingly resulted in avoidable deaths – 'that's already happened when you think about lockdown and deprivation of health care for a year.' He spoke of people prepared to do something that results in the avoidable death of their fellow human beings and it not bother them. This is the penny-drop I have been working to get across for more than 30 years – the level of pure evil we are dealing with. Yeadon said his friends and associates could not believe there could be that much evil, but he reminded them of Stalin, Pol Pot and Hitler and of what Stalin had said: 'One death is a tragedy. A million? A statistic.' He could not think of a benign explanation for why you need top-up vaccines 'which I'm sure you don't' and for the regulators 'to just get out of the way and wave them through'. Why would the regulators do that when they were still wrestling with the dangers of the 'parent' vaccine? He was clearly shocked by what he had seen since the 'Covid' hoax began and now he was thinking the previously unthinkable:

> If you wanted to depopulate a significant proportion of the world and to do it in a way that doesn't involve destruction of the environment with nuclear weapons, poisoning everyone with anthrax or something like that, and you wanted plausible deniability while you had a multi-year infectious disease crisis, I actually don't think you could come up with a better plan of work than seems to be in front of me. I can't say that's what they are going to do, but I can't think of a benign explanation why they are doing it.

He said he never thought that they would get rid of 99 percent of humans, but now he wondered. 'If you wanted to that this would be a hell of a way to do it – it would be unstoppable folks.' Yeadon had concluded that those who submitted to the 'vaccine' would be allowed to have some kind of normal life (but for how long?) while screws were tightened to coerce and mandate the last few percent. 'I think they'll put the rest of them in a prison camp. I wish I was wrong, but I don't think I am.' Other points he made included: There were no coronavirus vaccines then suddenly they all come along at the same time; we have no idea of the long term affect with trials so short; coercing or forcing people to have medical procedures is against the Nuremberg Code instigated when the Nazis did just that; people should at least delay having the 'vaccine'; a quick Internet search confirms that masks don't reduce respiratory viral transmission and 'the government knows that'; they have smashed civil society and they know that, too; two dozen peer-reviewed studies show no connection between lockdown and reducing deaths; he knew from personal friends the elite were still flying around and going on holiday while the public were locked down; the elite were not having the 'vaccines'. He was also asked if 'vaccines' could be made to target difference races. He said he didn't know, but the document by the Project for the New American Century in September, 2000, said developing 'advanced forms of biological warfare that can target *specific genotypes* may transform biological warfare from the realm of terror to a politically useful tool.' Oh, they're evil all right. Of that we can be *absolutely* sure.

Another cull of old people

We have seen from the CDC definition that the mRNA 'Covid vaccine' is not a vaccine and nor are the others that *claim* to reduce 'severity of symptoms' in *some* people, but not protect from infection or transmission. What about all the lies about returning to 'normal' if people were 'vaccinated'? If they are not claimed to stop infection and transmission of the alleged 'virus', how does anything change? This was all lies to manipulate people to take the jabs and we are seeing that now with masks and distancing still required for the 'vaccinated'. How did they think that elderly people with fragile health and immune responses were going to be affected by infusing their cells with synthetic material and other toxic substances? They *knew* that in the short and long term it would be devastating and fatal as the culling of the old that began with the first lockdowns was continued with the 'vaccine'. Death rates in care homes soared immediately residents began to be 'vaccinated' – infused with synthetic material. Brave and committed whistleblower nurses put their careers at risk by exposing this truth while the rest kept their heads down and their mouths shut to put their careers before those they are supposed to care for. A long-time American Certified Nursing Assistant who gave his name as James posted a video in which he described emotionally what happened in his care home when vaccination began. He said that during 2020 very few residents were sick with 'Covid' and no one died during the entire year; but shortly after the Pfizer mRNA injections 14 people died within two weeks and many others were near death. 'They're dropping like flies', he said. Residents who walked on their own before the shot could no longer and they had lost their ability to conduct an intelligent conversation. The home's management said the sudden deaths were caused by a 'super-spreader' of 'Covid-19'. Then how come, James asked, that residents who refused to take the injections were not sick? It was a case of inject the elderly with mRNA synthetic potions and blame their illness and death that followed on the 'virus'. James described what was happening in care homes as 'the greatest crime of genocide this country has ever seen'. Remember the NHS staff nurse from earlier who used the same

word 'genocide' for what was happening with the 'vaccines' and that it was an 'act of human annihilation'. A UK care home whistleblower told a similar story to James about the effect of the 'vaccine' in deaths and 'outbreaks' of illness dubbed 'Covid' after getting the jab. She told how her care home management and staff had zealously imposed government regulations and no one was allowed to even question the official narrative let alone speak out against it. She said the NHS was even worse. Again we see the results of reframing. A worker at a local care home where I live said they had not had a single case of 'Covid' there for almost a year and when the residents were 'vaccinated' they had 19 positive cases in two weeks with eight dying.

It's not the 'vaccine' – honest

The obvious cause and effect was being ignored by the media and most of the public. Australia's health minister Greg Hunt (a former head of strategy at the World Economic Forum) was admitted to hospital after he had the 'vaccine'. He was suffering according to reports from the skin infection 'cellulitis' and it must have been a severe case to have warranted days in hospital. Immediately the authorities said this was nothing to do with the 'vaccine' when an effect of some vaccines is a 'cellulitis-like reaction'. We had families of perfectly healthy old people who died after the 'vaccine' saying that if only they had been given the 'vaccine' earlier they would still be alive. As a numbskull rating that is off the chart. A father of four 'died of Covid' at aged 48 when he was taken ill two days after having the 'vaccine'. The man, a health administrator, had been 'shielding during the pandemic' and had 'not really left the house' until he went for the 'vaccine'. Having the 'vaccine' and then falling ill and dying does not seem to have qualified as a possible cause and effect and 'Covid-19' went on his death certificate. His family said they had no idea how he 'caught the virus'. A family member said: 'Tragically, it could be that going for a vaccination ultimately led to him catching Covid …The sad truth is that they are never going to know where it came from.' The family warned people to remember

that the virus still existed and was 'very real'. So was their stupidity. Nurses and doctors who had the first round of the 'vaccine' were collapsing, dying and ending up in a hospital bed while they or their grieving relatives were saying they'd still have the 'vaccine' again despite what happened. I kid you not. You mean if your husband returned from the dead he'd have the same 'vaccine' again that killed him??

Doctors at the VCU Medical Center in Richmond, Virginia, said the Johnson & Johnson 'vaccine' was to blame for a man's skin peeling off. Patient Richard Terrell said: 'It all just happened so fast. My skin peeled off. It's still coming off on my hands now.' He said it was stinging, burning and itching and when he bent his arms and legs it was very painful with 'the skin swollen and rubbing against itself'. Pfizer/BioNTech and Moderna vaccines use mRNA to change the cell while the Johnson & Johnson version uses DNA in a process similar to AstraZeneca's technique. Johnson & Johnson and AstraZeneca have both had their 'vaccines' paused by many countries after causing serious blood problems. Terrell's doctor Fnu Nutan said he could have died if he hadn't got medical attention. It sounds terrible so what did Nutan and Terrell say about the 'vaccine' now? Oh, they still recommend that people have it. A nurse in a hospital bed 40 minutes after the vaccination and unable to swallow due to throat swelling was told by a doctor that he lost mobility in his arm for 36 hours following the vaccination. What did he say to the ailing nurse? 'Good for you for getting the vaccination.' We are dealing with a serious form of cognitive dissonance madness in both public and medical staff. There is a remarkable correlation between those having the 'vaccine' and trumpeting the fact and suffering bad happenings shortly afterwards. Witold Rogiewicz, a Polish doctor, made a video of his 'vaccination' and ridiculed those who were questioning its safety and the intentions of Bill Gates: 'Vaccinate yourself to protect yourself, your loved ones, friends and also patients. And to mention quickly I have info for anti-vaxxers and anti-Coviders if you want to contact Bill Gates you can do this through me.' He further ridiculed the dangers of 5G. Days later he

was dead, but naturally the vaccination wasn't mentioned in the verdict of 'heart attack'.

Lies, lies and more lies

So many members of the human race have slipped into extreme states of insanity and unfortunately they include reframed doctors and nursing staff. Having a 'vaccine' and dying within minutes or hours is not considered a valid connection while death from any cause within 28 days or longer of a positive test with a test not testing for the 'virus' means 'Covid-19' goes on the death certificate. How could that 'vaccine'-death connection not have been made except by calculated deceit? US figures in the initial rollout period to February 12th, 2020, revealed that a third of the deaths reported to the CDC after 'Covid vaccines' happened within 48 hours. Five men in the UK suffered an 'extremely rare' blood clot problem after having the AstraZeneca 'vaccine', but no causal link was established said the Gates-funded Medicines and Healthcare products Regulatory Agency (MHRA) which had given the 'vaccine' emergency approval to be used. Former Pfizer executive Dr Michael Yeadon explained in his interview how the procedures could cause blood coagulation and clots. People who should have been at no risk were dying from blood clots in the brain and he said he had heard from medical doctor friends that people were suffering from skin bleeding and massive headaches. The AstraZeneca 'shot' was stopped by some 20 countries over the blood clotting issue and still the corrupt MHRA, the European Medicines Agency (EMA) and the World Health Organization said that it should continue to be given even though the EMA admitted that it 'still cannot rule out definitively' a link between blood clotting and the 'vaccine'. Later Marco Cavaleri, head of EMA vaccine strategy, said there was indeed a clear link between the 'vaccine' and thrombosis, but they didn't know why. So much for the trials showing the 'vaccine' is safe. Blood clots were affecting younger people who would be under virtually no danger from 'Covid' even if it existed which makes it all the more stupid and sinister.

The British government responded to public alarm by wheeling out June Raine, the terrifyingly weak infant school headmistress sound-alike who heads the UK MHRA drug 'regulator'. The idea that she would stand up to Big Pharma and government pressure is laughable and she told us that all was well in the same way that she did when allowing untested, never-used-on-humans-before, genetically-manipulating 'vaccines' to be exposed to the public in the first place. Mass lying is the new normal of the 'Covid' era. The MHRA later said 30 cases of rare blood clots had by then been connected with the AstraZeneca 'vaccine' (that means a lot more in reality) while stressing that the benefits of the jab in preventing 'Covid-19' outweighed any risks. A more ridiculous and disingenuous statement with callous disregard for human health it is hard to contemplate. Immediately after the mendacious 'all-clears' two hospital workers in Denmark experienced blood clots and cerebral haemorrhaging following the AstraZeneca jab and one died. Top Norwegian health official Pål Andre Holme said the 'vaccine' was the only common factor: 'There is nothing in the patient history of these individuals that can give such a powerful immune response … I am confident that the antibodies that we have found are the cause, and I see no other explanation than it being the vaccine which triggers it.' Strokes, a clot or bleed in the brain, were clearly associated with the 'vaccine' from word of mouth and whistleblower reports. Similar consequences followed with all these 'vaccines' that we were told were so safe and as the numbers grew by the day it was clear we were witnessing human carnage.

Learning the hard way

A woman interviewed by UKColumn told how her husband suffered dramatic health effects after the vaccine when he'd been in good health all his life. He went from being a little unwell to losing all feeling in his legs and experiencing 'excruciating pain'. Misdiagnosis followed twice at Accident and Emergency (an 'allergy' and 'sciatica') before he was admitted to a neurology ward where doctors said his serious condition had been caused by the

'vaccine'. Another seven 'vaccinated' people were apparently being treated on the same ward for similar symptoms. The woman said he had the 'vaccine' because they believed media claims that it was safe. 'I didn't think the government would give out a vaccine that does this to somebody; I believed they would be bringing out a vaccination that would be safe.' What a tragic way to learn that lesson. Another woman posted that her husband was transporting stroke patients to hospital on almost every shift and when he asked them if they had been 'vaccinated' for 'Covid' they all replied 'yes'. One had a 'massive brain bleed' the day after his second dose. She said her husband reported the 'just been vaccinated' information every time to doctors in A and E only for them to ignore it, make no notes and appear annoyed that it was even mentioned. This particular report cannot be verified, but it expresses a common theme that confirms the monumental underreporting of 'vaccine' consequences. Interestingly as the 'vaccines' and their brain blood clot/stroke consequences began to emerge the UK National Health Service began a publicity campaign telling the public what to do in the event of a stroke. A Scottish NHS staff nurse who quit in disgust in March, 2021, said:

> I have seen traumatic injuries from the vaccine, they're not getting reported to the yellow card [adverse reaction] scheme, they're treating the symptoms, not asking why, why it's happening. It's just treating the symptoms and when you speak about it you're dismissed like you're crazy, I'm not crazy, I'm not crazy because every other colleague I've spoken to is terrified to speak out, they've had enough.

Videos appeared on the Internet of people uncontrollably shaking after the 'vaccine' with no control over muscles, limbs and even their face. A Scottish mother broke out in a severe rash all over her body almost immediately after she was given the AstraZeneca 'vaccine'. The pictures were horrific. Leigh King, a 41-year-old hairdresser from Lanarkshire said: 'Never in my life was I prepared for what I was about to experience … My skin was so sore and constantly hot … I have never felt pain like this …' But don't you worry, the 'vaccine' is perfectly safe. Then there has been the effect on medical

staff who have been pressured to have the 'vaccine' by psychopathic 'health' authorities and government. A London hospital consultant who gave the name K. Polyakova wrote this to the *British Medical Journal* or *BMJ*:

> I am currently struggling with … the failure to report the reality of the morbidity caused by our current vaccination program within the health service and staff population. The levels of sickness after vaccination is unprecedented and staff are getting very sick and some with neurological symptoms which is having a huge impact on the health service function. Even the young and healthy are off for days, some for weeks, and some requiring medical treatment. Whole teams are being taken out as they went to get vaccinated together.
>
> Mandatory vaccination in this instance is stupid, unethical and irresponsible when it comes to protecting our staff and public health. We are in the voluntary phase of vaccination, and encouraging staff to take an unlicensed product that is impacting on their immediate health … it is clearly stated that these vaccine products do not offer immunity or stop transmission. In which case why are we doing it?

Not to protect health that's for sure. Medical workers are lauded by governments for agenda reasons when they couldn't give a toss about them any more than they can for the population in general. Schools across America faced the same situation as they closed due to the high number of teachers and other staff with bad reactions to the Pfizer/BioNTech, Moderna, and Johnson & Johnson 'Covid vaccines' all of which were linked to death and serious adverse effects. The *BMJ* took down the consultant's comments pretty quickly on the grounds that they were being used to spread 'disinformation'. They were exposing the truth about the 'vaccine' was the real reason. The cover-up is breathtaking.

Hiding the evidence

The scale of the 'vaccine' death cover-up worldwide can be confirmed by comparing official figures with the personal experience of the public. I heard of many people in my community who died immediately or soon after the vaccine that would never appear in the media or even likely on the official totals of 'vaccine' fatalities and adverse reactions when only about ten percent are estimated to be

reported and I have seen some estimates as low as one percent in a Harvard study. In the UK alone by April 29th, 2021, some 757,654 adverse reactions had been officially reported from the Pfizer/BioNTech, Oxford/AstraZeneca and Moderna 'vaccines' with more than a thousand deaths linked to jabs and that means an estimated ten times this number in reality from a ten percent reporting rate percentage. That's seven million adverse reactions and 10,000 potential deaths and a one percent reporting rate would be ten times *those* figures. In 1976 the US government pulled the swine flu vaccine after *53* deaths. The UK data included a combined 10,000 eye disorders from the 'Covid vaccines' with more than 750 suffering visual impairment or blindness and again multiply by the estimated reporting percentages. As 'Covid cases' officially fell hospitals virtually empty during the 'Covid crisis' began to fill up with a range of other problems in the wake of the 'vaccine' rollout. The numbers across America have also been catastrophic. Deaths linked to *all* types of vaccine increased by *6,000 percent* in the first quarter of 2021 compared with 2020. A 39-year-old woman from Ogden, Utah, died four days after receiving a second dose of Moderna's 'Covid vaccine' when her liver, heart and kidneys all failed despite the fact that she had no known medical issues or conditions. Her family sought an autopsy, but Dr Erik Christensen, Utah's chief medical examiner, said proving vaccine injury as a cause of death almost never happened. He could think of only one instance where an autopsy would name a vaccine as the official cause of death and that would be anaphylaxis where someone received a vaccine and died almost instantaneously. 'Short of that, it would be difficult for us to definitively say this is the vaccine,' Christensen said. If that is true this must be added to the estimated ten percent (or far less) reporting rate of vaccine deaths and serious reactions and the conclusion can only be that vaccine deaths and serious reactions – including these 'Covid' potions' – are phenomenally understated in official figures. The same story can be found everywhere. Endless accounts of deaths and serious reactions among the public, medical

and care home staff while official figures did not even begin to reflect this.

Professional script-reader Dr David Williams, a 'top public-health official' in Ontario, Canada, insulted our intelligence by claiming only four serious adverse reactions and no deaths from the more than 380,000 vaccine doses then given. This bore no resemblance to what people knew had happened in their owns circles and we had Dirk Huyer in charge of getting millions vaccinated in Ontario while at the same time he was Chief Coroner for the province investigating causes of death including possible death from the vaccine. An aide said he had stepped back from investigating deaths, but evidence indicated otherwise. Rosemary Frei, who secured a Master of Science degree in molecular biology at the Faculty of Medicine at Canada's University of Calgary before turning to investigative journalism, was one who could see that official figures for 'vaccine' deaths and reactions made no sense. She said that doctors seldom reported adverse events and when people got really sick or died after getting a vaccination they would attribute that to anything except the vaccines. It had been that way for years and anyone who wondered aloud whether the 'Covid vaccines' or other shots cause harm is immediately branded as 'anti-vax' and 'anti-science'. This was 'career-threatening' for health professionals. Then there was the huge pressure to support the push to 'vaccinate' billions in the quickest time possible. Frei said:

> So that's where we're at today. More than half a million vaccine doses have been given to people in Ontario alone. The rush is on to vaccinate all 15 million of us in the province by September. And the mainstream media are screaming for this to be sped up even more. That all adds up to only a very slim likelihood that we're going to be told the truth by officials about how many people are getting sick or dying from the vaccines.

What is true of Ontario is true of everywhere.

They KNEW – and still did it

The authorities knew what was going to happen with multiple deaths and adverse reactions. The UK government's Gates-funded

and Big Pharma-dominated Medicines and Healthcare products Regulatory Agency (MHRA) hired a company to employ AI in compiling the projected reactions to the 'vaccine' that would otherwise be uncountable. The request for applications said: 'The MHRA urgently seeks an Artificial Intelligence (AI) software tool to process the expected high volume of Covid-19 vaccine Adverse Drug Reaction …' This was from the agency, headed by the disingenuous June Raine, that gave the 'vaccines' emergency approval and the company was hired before the first shot was given. 'We are going to kill and maim you – is that okay?' 'Oh, yes, perfectly fine – I'm very grateful, thank you, doctor.' The range of 'Covid vaccine' adverse reactions goes on for page after page in the MHRA criminally underreported 'Yellow Card' system and includes affects to eyes, ears, skin, digestion, blood and so on. Raine's MHRA amazingly claimed that the 'overall safety experience … is so far as expected from the clinical trials'. The death, serious adverse effects, deafness and blindness were *expected?* When did they ever mention that? If these human tragedies were expected then those that gave approval for the use of these 'vaccines' must be guilty of crimes against humanity including murder – a definition of which is 'killing a person with malice aforethought or with recklessness manifesting extreme indifference to the value of human life.' People involved at the MHRA, the CDC in America and their equivalent around the world must go before Nuremberg trials to answer for their callous inhumanity. We are only talking here about the immediate effects of the 'vaccine'. The longer-term impact of the DNA synthetic manipulation is the main reason they are so hysterically desperate to inoculate the entire global population in the shortest possible time.

Africa and the developing world are a major focus for the 'vaccine' depopulation agenda and a mass vaccination sales-pitch is underway thanks to caring people like the Rockefellers and other Cult assets. The Rockefeller Foundation, which pre-empted the 'Covid pandemic' in a document published in 2010 that 'predicted' what happened a decade later, announced an initial $34.95 million grant in February, 2021, 'to ensure more equitable access to Covid-19

testing and vaccines' among other things in Africa in collaboration with '24 organizations, businesses, and government agencies'. The pan-Africa initiative would focus on 10 countries: Burkina Faso, Ethiopia, Ghana, Kenya, Nigeria, Rwanda, South Africa, Tanzania, Uganda, and Zambia'. Rajiv Shah, President of the Rockefeller Foundation and former administrator of CIA-controlled USAID, said that if Africa was not mass-vaccinated (to change the DNA of its people) it was a 'threat to all of humanity' and not fair on Africans. When someone from the Rockefeller Foundation says they want to do something to help poor and deprived people and countries it is time for a belly-laugh. They are doing this out of the goodness of their 'heart' because 'vaccinating' the entire global population is what the 'Covid' hoax set out to achieve. Official 'decolonisation' of Africa by the Cult was merely a prelude to financial colonisation on the road to a return to physical colonisation. The 'vaccine' is vital to that and the sudden and convenient death of the 'Covid' sceptic president of Tanzania can be seen in its true light. A lot of people in Africa are aware that this is another form of colonisation and exploitation and they need to stand their ground.

The 'vaccine is working' scam

A potential problem for the Cult was that the 'vaccine' is meant to change human DNA and body messaging and not to protect anyone from a 'virus' never shown to exist. The vaccine couldn't work because it was not designed to work and how could they make it *appear* to be working so that more people would have it? This was overcome by lowering the amplification rate of the PCR test to produce fewer 'cases' and therefore fewer 'deaths'. Some of us had been pointing out since March, 2020, that the amplification rate of the test not testing for the 'virus' had been made artificially high to generate positive tests which they could call 'cases' to justify lockdowns. The World Health Organization recommended an absurdly high 45 amplification cycles to ensure the high positives required by the Cult and then remained silent on the issue until January 20th, 2021 – Biden's Inauguration Day. This was when the

'vaccinations' were seriously underway and on that day the WHO recommended after discussions with America's CDC that laboratories *lowered their testing amplification*. Dr David Samadi, a certified urologist and health writer, said the WHO was encouraging all labs to reduce their cycle count for PCR tests. He said the current cycle was much too high and was 'resulting in any particle being declared a positive case'. Even one mainstream news report I saw said this meant the number of 'Covid' infections may have been 'dramatically inflated'. Oh, just a little bit. The CDC in America issued new guidance to laboratories in April, 2021, to use 28 cycles *but only for 'vaccinated' people*. The timing of the CDC/WHO interventions were cynically designed to make it appear the 'vaccines' were responsible for falling cases and deaths when the real reason can be seen in the following examples. New York's state lab, the Wadsworth Center, identified 872 positive tests in July, 2020, based on a threshold of 40 cycles. When the figure was lowered to 35 cycles *43 percent* of the 872 were no longer 'positives'. At 30 cycles the figure was 63 percent. A Massachusetts lab found that between *85 to 90 percent* of people who tested positive in July with a cycle threshold of 40 would be negative at 30 cycles, Ashish Jha, MD, director of the Harvard Global Health Institute, said: 'I'm really shocked that it could be that high … Boy, does it really change the way we need to be thinking about testing.' I'm shocked that I could see the obvious in the spring of 2020, with no medical background, and most medical professionals still haven't worked it out. No, that's not shocking – it's terrifying.

Three weeks after the WHO directive to lower PCR cycles the London *Daily Mail* ran this headline: 'Why ARE Covid cases plummeting? New infections have fallen 45% in the US and 30% globally in the past 3 weeks but experts say vaccine is NOT the main driver because only 8% of Americans and 13% of people worldwide have received their first dose.' They acknowledged that the drop could not be attributed to the 'vaccine', but soon this morphed throughout the media into the 'vaccine' has caused cases and deaths to fall when it was the PCR threshold. In December, 2020, there was

chaos at English Channel ports with truck drivers needing negative 'Covid' tests before they could board a ferry home for Christmas. The government wanted to remove the backlog as fast as possible and they brought in troops to do the 'testing'. Out of 1,600 drivers just *36* tested positive and the rest were given the all clear to cross the Channel. I guess the authorities thought that 36 was the least they could get away with without the unquestioning catching on. The amplification trick which most people believed in the absence of information in the mainstream applied more pressure on those refusing the 'vaccine' to succumb when it 'obviously worked'. The truth was the exact opposite with deaths in care homes soaring with the 'vaccine' and in Israel the term used was 'skyrocket'. A re-analysis of published data from the Israeli Health Ministry led by Dr Hervé Seligmann at the Medicine Emerging Infectious and Tropical Diseases at Aix-Marseille University found that Pfizer's 'Covid vaccine' killed 'about 40 times more [elderly] people than the disease itself would have killed' during a five-week vaccination period and *260 times* more younger people than would have died from the 'virus' even according to the manipulated 'virus' figures. Dr Seligmann and his co-study author, Haim Yativ, declared after reviewing the Israeli 'vaccine' death data: 'This is a new Holocaust.'

Then, in mid-April, 2021, after vast numbers of people worldwide had been 'vaccinated', the story changed with clear coordination. The UK government began to prepare the ground for more future lockdowns when Nuremberg-destined Boris Johnson told yet another whopper. He said that cases had fallen because of *lockdowns* not 'vaccines'. Lockdowns are irrelevant when *there is no 'virus'* and the test and fraudulent death certificates are deciding the number of 'cases' and 'deaths'. Study after study has shown that lockdowns don't work and instead kill and psychologically destroy people. Meanwhile in the United States Anthony Fauci and Rochelle Walensky, the ultra-Zionist head of the CDC, peddled the same line. More lockdown was the answer and not the 'vaccine', a line repeated on cue by the moron that is Canadian Prime Minister Justin Trudeau. Why all the hysteria to get everyone 'vaccinated' if lockdowns and

not 'vaccines' made the difference? None of it makes sense on the face of it. Oh, but it does. The Cult wants lockdowns *and* the 'vaccine' and if the 'vaccine' is allowed to be seen as the total answer lockdowns would no longer be justified when there are still livelihoods to destroy. 'Variants' and renewed upward manipulation of PCR amplification are planned to instigate never-ending lockdown *and* more 'vaccines'.

You *must* have it – we're desperate

Israel, where the Jewish and Arab population are ruled by the Sabbatian Cult, was the front-runner in imposing the DNA-manipulating 'vaccine' on its people to such an extent that Jewish refusers began to liken what was happening to the early years of Nazi Germany. This would seem to be a fantastic claim. Why would a government of Jewish people be acting like the Nazis did? If you realise that the Sabbatian Cult was behind the Nazis and that Sabbatians hate Jews the pieces start to fit and the question of why a 'Jewish' government would treat Jews with such callous disregard for their lives and freedom finds an answer. Those controlling the government of Israel *aren't Jewish* – they're Sabbatian. Israeli lawyer Tamir Turgal was one who made the Nazi comparison in comments to German lawyer Reiner Fuellmich who is leading a class action lawsuit against the psychopaths for crimes against humanity. Turgal described how the Israeli government was vaccinating children and pregnant women on the basis that there was no evidence that this was dangerous when they had no evidence that it *wasn't* dangerous either. They just had no evidence. This was medical experimentation and Turgal said this breached the Nuremberg Code about medical experimentation and procedures requiring informed consent and choice. Think about that. A Nuremberg Code developed because of Nazi experimentation on Jews and others in concentration camps by people like the evil-beyond-belief Josef Mengele is being breached by the *Israeli* government; but when you know that it's a *Sabbatian* government along with its intelligence and military agencies like Mossad, Shin Bet and the Israeli Defense Forces, and that Sabbatians

were the force behind the Nazis, the kaleidoscope comes into focus. What have we come to when Israeli Jews are suing their government for violating the Nuremberg Code by essentially making Israelis subject to a medical experiment using the controversial 'vaccines'? It's a shocker that this has to be done in the light of what happened in Nazi Germany. The Anshe Ha-Emet, or 'People of the Truth', made up of Israeli doctors, lawyers, campaigners and public, have launched a lawsuit with the International Criminal Court. It says:

> When the heads of the Ministry of Health as well as the prime minister presented the vaccine in Israel and began the vaccination of Israeli residents, the vaccinated were not advised, that, in practice, they are taking part in a medical experiment and that their consent is required for this under the Nuremberg Code.

The irony is unbelievable, but easily explained in one word: Sabbatians. The foundation of Israeli 'Covid' apartheid is the 'green pass' or 'green passport' which allows Jews and Arabs who have had the DNA-manipulating 'vaccine' to go about their lives – to work, fly, travel in general, go to shopping malls, bars, restaurants, hotels, concerts, gyms, swimming pools, theatres and sports venues, while non-'vaccinated' are banned from all those places and activities. Israelis have likened the 'green pass' to the yellow stars that Jews in Nazi Germany were forced to wear – the same as the yellow stickers that a branch of UK supermarket chain Morrisons told exempt mask-wears they had to display when shopping. How very sensitive. The Israeli system is blatant South African-style apartheid on the basis of compliance or non-compliance to fascism rather than colour of the skin. How appropriate that the Sabbatian Israeli government was so close to the pre-Mandela apartheid regime in Pretoria. The Sabbatian-instigated 'vaccine passport' in Israel is planned for everywhere. Sabbatians struck a deal with Pfizer that allowed them to lead the way in the percentage of a national population infused with synthetic material and the result was catastrophic. Israeli freedom activist Shai Dannon told me how chairs were appearing on beaches that said 'vaccinated only'. Health Minister Yuli Edelstein said that anyone unwilling or unable to get

the jabs that 'confer immunity' will be 'left behind'. The man's a liar. Not even the makers claim the 'vaccines' confer immunity. When you see those figures of 'vaccine' deaths these psychopaths were saying that you must take the chance the 'vaccine' will kill you or maim you while knowing it will change your DNA or lockdown for you will be permanent. That's fascism. The Israeli parliament passed a law to allow personal information of the non-vaccinated to be shared with local and national authorities for three months. This was claimed by its supporters to be a way to 'encourage' people to be vaccinated. Hadas Ziv from Physicians for Human Rights described this as a 'draconian law which crushed medical ethics and the patient rights'. But that's the idea, the Sabbatians would reply.

Your papers, please

Sabbatian Israel was leading what has been planned all along to be a global 'vaccine pass' called a 'green passport' without which you would remain in permanent lockdown restriction and unable to do anything. This is how badly – *desperately* – the Cult is to get everyone 'vaccinated'. The term and colour 'green' was not by chance and related to the psychology of fusing the perception of the green climate hoax with the 'Covid' hoax and how the 'solution' to both is the same Great Reset. Lying politicians, health officials and psychologists denied there were any plans for mandatory vaccinations or restrictions based on vaccinations, but they knew that was exactly what was meant to happen with governments of all countries reaching agreements to enforce a global system. 'Free' Denmark and 'free' Sweden unveiled digital vaccine certification. Cyprus, Czech Republic, Estonia, Greece, Hungary, Iceland, Italy, Poland, Portugal, Slovakia, and Spain have all committed to a vaccine passport system and the rest including the whole of the EU would follow. The satanic UK government will certainly go this way despite mendacious denials and at the time of writing it is trying to manipulate the public into having the 'vaccine' so they could go abroad on a summer holiday. How would that work without something to prove you had the synthetic toxicity injected into you?

Documents show that the EU's European Commission was moving towards 'vaccine certificates' in 2018 and 2019 before the 'Covid' hoax began. They knew what was coming. Abracadabra – Ursula von der Leyen, the German President of the Commission, announced in March, 2021, an EU 'Digital Green Certificate' – green again – to track the public's 'Covid status'. The passport sting is worldwide and the Far East followed the same pattern with South Korea ruling that only those with 'vaccination' passports – again the *green* pass – would be able to 'return to their daily lives'.

Bill Gates has been preparing for this 'passport' with other Cult operatives for years and beyond the paper version is a Gates-funded 'digital tattoo' to identify who has been vaccinated and who hasn't. The 'tattoo' is reported to include a substance which is externally readable to confirm who has been vaccinated. This is a bio-luminous light-generating enzyme (think fireflies) called … *Luciferase*. Yes, named after the Cult 'god' Lucifer the 'light bringer' of whom more to come. Gates said he funded the readable tattoo to ensure children in the developing world were vaccinated and no one was missed out. He cares so much about poor kids as we know. This was just the cover story to develop a vaccine tagging system for everyone on the planet. Gates has been funding the ID2020 'alliance' to do just that in league with other lovely people at Microsoft, GAVI, the Rockefeller Foundation, Accenture and IDEO.org. He said in interviews in March, 2020, before any 'vaccine' publicly existed, that the world must have a globalised digital certificate to track the 'virus' and who had been vaccinated. Gates knew from the start that the mRNA vaccines were coming and when they would come and that the plan was to tag the 'vaccinated' to marginalise the intelligent and stop them doing anything including travel. Evil just doesn't suffice. Gates was exposed for offering a $10 million bribe to the Nigerian House of Representatives to invoke compulsory 'Covid' vaccination of all Nigerians. Sara Cunial, a member of the Italian Parliament, called Gates a 'vaccine criminal'. She urged the Italian President to hand him over to the International Criminal Court for crimes against

humanity and condemned his plans to 'chip the human race' through ID2020.

You know it's a long-planned agenda when war criminal and Cult gofer Tony Blair is on the case. With the scale of arrogance only someone as dark as Blair can muster he said: 'Vaccination in the end is going to be your route to liberty.' Blair is a disgusting piece of work and he confirms that again. The media has given a lot of coverage to a bloke called Charlie Mullins, founder of London's biggest independent plumbing company, Pimlico Plumbers, who has said he won't employ anyone who has not been vaccinated or have them go to any home where people are not vaccinated. He said that if he had his way no one would be allowed to walk the streets if they have not been vaccinated. Gates was cheering at the time while I was alerting the white coats. The plan is that people will qualify for 'passports' for having the first two doses and then to keep it they will have to have all the follow ups and new ones for invented 'variants' until human genetics is transformed and many are dead who can't adjust to the changes. Hollywood celebrities – the usual propaganda stunt – are promoting something called the WELL Health-Safety Rating to verify that a building or space has 'taken the necessary steps to prioritize the health and safety of their staff, visitors and other stakeholders'. They included Lady Gaga, Jennifer Lopez, Michael B. Jordan, Robert DeNiro, Venus Williams, Wolfgang Puck, Deepak Chopra and 17th Surgeon General Richard Carmona. Yawn. WELL Health-Safety has big connections with China. Parent company Delos is headed by former Goldman Sachs partner Paul Scialla. This is another example – and we will see so many others – of using the excuse of 'health' to dictate the lives and activities of the population. I guess one confirmation of the 'safety' of buildings is that only 'vaccinated' people can go in, right?

Electronic concentration camps

I wrote decades ago about the plans to restrict travel and here we are for those who refuse to bow to tyranny. This can be achieved in one go with air travel if the aviation industry makes a blanket decree.

The 'vaccine' and guaranteed income are designed to be part of a global version of China's social credit system which tracks behaviour 24/7 and awards or deletes 'credits' based on whether your behaviour is supported by the state or not. I mean your entire lifestyle – what you do, eat, say, everything. Once your credit score falls below a certain level consequences kick in. In China tens of millions have been denied travel by air and train because of this. All the locations and activities denied to refusers by the 'vaccine' passports will be included in one big mass ban on doing almost anything for those that don't bow their head to government. It's beyond fascist and a new term is required to describe its extremes – I guess fascist technocracy will have to do. The way the Chinese system of technological – technocratic – control is sweeping the West can be seen in the Los Angeles school system and is planned to be expanded worldwide. Every child is required to have a 'Covid'-tracking app scanned daily before they can enter the classroom. The so-called Daily Pass tracking system is produced by Gates' Microsoft which I'm sure will shock you rigid. The pass will be scanned using a barcode (one step from an inside-the-body barcode) and the information will include health checks, 'Covid' tests and vaccinations. Entry codes are for one specific building only and access will only be allowed if a student or teacher has a negative test with a test not testing for the 'virus', has no symptoms of anything alleged to be related to 'Covid' (symptoms from a range of other illness), and has a temperature under 100 degrees. No barcode, no entry, is planned to be the case for everywhere and not only schools.

Kids are being psychologically prepared to accept this as 'normal' their whole life which is why what they can impose in schools is so important to the Cult and its gofers. Long-time American freedom campaigner John Whitehead of the Rutherford Institute was not exaggerating when he said: 'Databit by databit, we are building our own electronic concentration camps.' Canada under its Cult gofer prime minister Justin Trudeau has taken a major step towards the real thing with people interned against their will if they test positive with a test not testing for the 'virus' when they arrive at a Canadian

airport. They are jailed in internment hotels often without food or water for long periods and with many doors failing to lock there have been sexual assaults. The interned are being charged sometimes $2,000 for the privilege of being abused in this way. Trudeau is fully on board with the Cult and says the 'Covid pandemic' has provided an opportunity for a global 'reset' to permanently change Western civilisation. His number two, Deputy Prime Minister Chrystia Freeland, is a trustee of the World Economic Forum and a Rhodes Scholar. The Trudeau family have long been servants of the Cult. See *The Biggest Secret* and Cathy O'Brien's book *Trance-Formation of America* for the horrific background to Trudeau's father Pierre Trudeau another Canadian prime minister. Hide your fascism behind the façade of a heart-on-the-sleeve liberal. It's a well-honed Cult technique.

What can the 'vaccine' *really* do?

We have a 'virus' never shown to exist and 'variants' of the 'virus' that have also never been shown to exist except, like the 'original', as computer-generated fictions. Even if you believe there's a 'virus' the 'case' to 'death' rate is in the region of 0.23 to 0.15 percent and those 'deaths' are concentrated among the very old around the same average age that people die anyway. In response to this lack of threat (in truth none) psychopaths and idiots, knowingly and unknowingly answering to Gates and the Cult, are seeking to 'vaccinate' every man, woman and child on Planet Earth. Clearly the 'vaccine' is not about 'Covid' – none of this ever has been. So what is it all about *really*? Why the desperation to infuse genetically-manipulating synthetic material into everyone through mRNA fraudulent 'vaccines' with the intent of doing this over and over with the excuses of 'variants' and other 'virus' inventions? Dr Sherri Tenpenny, an osteopathic medical doctor in the United States, has made herself an expert on vaccines and their effects as a vehement campaigner against their use. Tenpenny was board certified in emergency medicine, the director of a level two trauma centre for 12 years, and moved to Cleveland in 1996 to start an integrative

medicine practice which has treated patients from all 50 states and some 17 other countries. Weaning people off pharmaceutical drugs is a speciality.

She became interested in the consequences of vaccines after attending a meeting at the National Vaccine Information Center in Washington DC in 2000 where she 'sat through four days of listening to medical doctors and scientists and lawyers and parents of vaccine injured kids' and asked: 'What's going on?' She had never been vaccinated and never got ill while her father was given a list of vaccines to be in the military and was 'sick his entire life'. The experience added to her questions and she began to examine vaccine documents from the Centers for Disease Control (CDC). After reading the first one, the 1998 version of *The General Recommendations of Vaccination*, she thought: 'This is it?' The document was poorly written and bad science and Tenpenny began 20 years of research into vaccines that continues to this day. She began her research into 'Covid vaccines' in March, 2020, and she describes them as 'deadly'. For many, as we have seen, they already have been. Tenpenny said that in the first 30 days of the 'vaccine' rollout in the United States there had been more than 40,000 adverse events reported to the vaccine adverse event database. A document had been delivered to her the day before that was 172 pages long. 'We have over 40,000 adverse events; we have over 3,100 cases of [potentially deadly] anaphylactic shock; we have over 5,000 neurological reactions.' Effects ranged from headaches to numbness, dizziness and vertigo, to losing feeling in hands or feet and paraesthesia which is when limbs 'fall asleep' and people have the sensation of insects crawling underneath their skin. All this happened in the first 30 days and remember that only about *ten percent* (or far less) of adverse reactions and vaccine-related deaths are estimated to be officially reported. Tenpenny said:

So can you think of one single product in any industry, any industry, for as long as products have been made on the planet that within 30 days we have 40,000 people complaining of side effects that not only is still on the market but … we've got paid actors telling us how great

they are for getting their vaccine. We're offering people $500 if they will just get their vaccine and we've got nurses and doctors going; 'I got the vaccine, I got the vaccine'.

Tenpenny said they were not going to be 'happy dancing folks' when they began to suffer Bell's palsy (facial paralysis), neuropathies, cardiac arrhythmias and autoimmune reactions that kill through a blood disorder. 'They're not going to be so happy, happy then, but we're never going to see pictures of those people' she said. Tenpenny described the 'vaccine' as 'a well-designed killing tool'.

No off-switch

Bad as the initial consequences had been Tenpenny said it would be maybe 14 months before we began to see the 'full ravage' of what is going to happen to the 'Covid vaccinated' with full-out consequences taking anything between two years and 20 years to show. You can understand why when you consider that variations of the 'Covid vaccine' use mRNA (messenger RNA) to in theory activate the immune system to produce protective antibodies without using the actual 'virus'. How can they when it's a computer program and they've never isolated what they claim is the 'real thing'? Instead they use *synthetic* mRNA. They are inoculating synthetic material into the body which through a technique known as the Trojan horse is absorbed into cells to change the nature of DNA. Human DNA is changed by an infusion of messenger RNA and with each new 'vaccine' of this type it is changed even more. Say so and you are banned by Cult Internet platforms. The contempt the contemptuous Mark Zuckerberg has for the truth and human health can be seen in an internal Facebook video leaked to the Project Veritas investigative team in which he said of the 'Covid vaccines': '… I share some caution on this because we just don't know the long term side-effects of basically modifying people's DNA and RNA.' At the same time this disgusting man's Facebook was censoring and banning anyone saying exactly the same. He must go before a Nuremberg trial for crimes against humanity when he *knows* that he

is censoring legitimate concerns and denying the right of informed consent on behalf of the Cult that owns him. People have been killed and damaged by the very 'vaccination' technique he cast doubt on himself when they may not have had the 'vaccine' with access to information that he denied them. The plan is to have at least annual 'Covid vaccinations', add others to deal with invented 'variants', and change all other vaccines into the mRNA system. Pfizer executives told shareholders at a virtual Barclays Global Healthcare Conference in March, 2021, that the public may need a third dose of 'Covid vaccine', plus regular yearly boosters and the company planned to hike prices to milk the profits in a 'significant opportunity for our vaccine'. These are the professional liars, cheats and opportunists who are telling you their 'vaccine' is safe. Given this volume of mRNA planned to be infused into the human body and its ability to then replicate we will have a transformation of human genetics from biological to synthetic biological – exactly the long-time Cult plan for reasons we'll see – and many will die. Sherri Tenpenny said of this replication:

> It's like having an on-button but no off-button and that whole mechanism … they actually give it a name and they call it the Trojan horse mechanism, because it allows that [synthetic] virus and that piece of that [synthetic] virus to get inside of your cells, start to replicate and even get inserted into other parts of your DNA as a Trojan-horse.

Ask the overwhelming majority of people who have the 'vaccine' what they know about the contents and what they do and they would reply: 'The government says it will stop me getting the virus.' Governments give that false impression on purpose to increase take-up. You can read Sherri Tenpenny's detailed analysis of the health consequences in her blog at Vaxxter.com, but in summary these are some of them. She highlights the statement by Bill Gates about how human beings can become their own 'vaccine manufacturing machine'. The man is insane. ['Vaccine'-generated] 'antibodies' carry synthetic messenger RNA into the cells and the damage starts, Tenpenny contends, and she says that lungs can be adversely affected through varying degrees of pus and bleeding which

obviously affects breathing and would be dubbed 'Covid-19'. Even more sinister was the impact of 'antibodies' on macrophages, a white blood cell of the immune system. They consist of Type 1 and Type 2 which have very different functions. She said Type 1 are 'hyper-vigilant' white blood cells which 'gobble up' bacteria etc. However, in doing so, this could cause inflammation and in extreme circumstances be fatal. She says these affects are mitigated by Type 2 macrophages which kick in to calm down the system and stop it going rogue. They clear up dead tissue debris and reduce inflammation that the Type 1 'fire crews' have caused. Type 1 kills the infection and Type 2 heals the damage, she says. This is her punchline with regard to 'Covid vaccinations': She says that mRNA 'antibodies' block Type 2 macrophages by attaching to them and deactivating them. This meant that when the Type 1 response was triggered by infection there was nothing to stop that getting out of hand by calming everything down. There's an on-switch, but no off-switch, she says. What follows can be 'over and out, see you when I see you'.

Genetic suicide

Tenpenny also highlights the potential for autoimmune disease – the body attacking itself – which has been associated with vaccines since they first appeared. Infusing a synthetic foreign substance into cells could cause the immune system to react in a panic believing that the body is being overwhelmed by an invader (it is) and the consequences can again be fatal. There is an autoimmune response known as a 'cytokine storm' which I have likened to a homeowner panicked by an intruder and picking up a gun to shoot randomly in all directions before turning the fire on himself. The immune system unleashes a storm of inflammatory response called cytokines to a threat and the body commits hara-kiri. The lesson is that you mess with the body's immune response at your peril and these 'vaccines' seriously – fundamentally – mess with immune response. Tenpenny refers to a consequence called anaphylactic shock which is a severe and highly dangerous allergic reaction when the immune system

floods the body with chemicals. She gives the example of having a bee sting which primes the immune system and makes it sensitive to those chemicals. When people are stung again maybe years later the immune response can be so powerful that it leads to anaphylactic shock. Tenpenny relates this 'shock' with regard to the 'Covid vaccine' to something called polyethylene glycol or PEG. Enormous numbers of people have become sensitive to this over decades of use in a whole range of products and processes including food, drink, skin creams and 'medicine'. Studies have claimed that some 72 percent of people have antibodies triggered by PEG compared with two percent in the 1960s and allergic hypersensitive reactions to this become a gathering cause for concern. Tenpenny points out that the 'mRNA vaccine' is coated in a 'bubble' of polyethylene glycol which has the potential to cause anaphylactic shock through immune sensitivity. Many reports have appeared of people reacting this way after having the 'Covid vaccine'. What do we think is going to happen as humanity has more and more of these 'vaccines'? Tenpenny said: 'All these pictures we have seen with people with these rashes … these weepy rashes, big reactions on their arms and things like that – it's an acute allergic reaction most likely to the polyethylene glycol that you've been previously primed and sensitised to.'

Those who have not studied the conspiracy and its perpetrators at length might think that making the population sensitive to PEG and then putting it in these 'vaccines' is just a coincidence. It is not. It is instead testament to how carefully and coldly-planned current events have been and the scale of the conspiracy we are dealing with. Tenpenny further explains that the 'vaccine' mRNA procedure can breach the blood-brain barrier which protects the brain from toxins and other crap that will cause malfunction. In this case they could make two proteins corrupt brain function to cause Amyotrophic lateral sclerosis (ALS) , a progressive nervous system disease leading to loss of muscle control, and frontal lobe degeneration – Alzheimer's and dementia. Immunologist J. Bart Classon published a paper connecting mRNA 'vaccines' to prion

disease which can lead to Alzheimer's and other forms of neurogenerative disease while others have pointed out the potential to affect the placenta in ways that make women infertile. This will become highly significant in the next chapter when I will discuss other aspects of this non-vaccine that relate to its nanotechnology and transmission from the injected to the uninjected.

Qualified in idiocy

Tenpenny describes how research has confirmed that these 'vaccine'-generated antibodies can interact with a range of other tissues in the body and attack many other organs including the lungs. 'This means that if you have a hundred people standing in front of you that all got this shot they could have a hundred different symptoms.' Anyone really think that Cult gofers like the Queen, Tony Blair, Christopher Whitty, Anthony Fauci, and all the other psychopaths have really had this 'vaccine' in the pictures we've seen? Not a bloody chance. Why don't doctors all tell us about all these dangers and consequences of the 'Covid vaccine'? Why instead do they encourage and pressure patients to have the shot? Don't let's think for a moment that doctors and medical staff can't be stupid, lazy, and psychopathic and that's without the financial incentives to give the jab. Tenpenny again:

> Some people are going to die from the vaccine directly but a large number of people are going to start to get horribly sick and get all kinds of autoimmune diseases 42 days to maybe a year out. What are they going to do, these stupid doctors who say; 'Good for you for getting that vaccine.' What are they going to say; 'Oh, it must be a mutant, we need to give an extra dose of that vaccine.'
>
> Because now the vaccine, instead of one dose or two doses we need three or four because the stupid physicians aren't taking the time to learn anything about it. If I can learn this sitting in my living room reading a 19 page paper and several others so can they. There's nothing special about me, I just take the time to do it.

Remember how Sara Kayat, the NHS and TV doctor, said that the 'Covid vaccine' would '100 percent prevent hospitalisation and death'. Doctors can be idiots like every other profession and they

should not be worshipped as infallible. They are not and far from it. Behind many medical and scientific 'experts' lies an uninformed prat trying to hide themselves from you although in the 'Covid' era many have failed to do so as with UK narrative-repeating 'TV doctor' Hilary Jones. Pushing back against the minority of proper doctors and scientists speaking out against the 'vaccine' has been the entire edifice of the Cult global state in the form of governments, medical systems, corporations, mainstream media, Silicon Valley, and an army of compliant doctors, medical staff and scientists willing to say anything for money and to enhance their careers by promoting the party line. If you do that you are an 'expert' and if you won't you are an 'anti-vaxxer' and 'Covidiot'. The pressure to be 'vaccinated' is incessant. We have even had reports claiming that the 'vaccine' can help cure cancer and Alzheimer's and make the lame walk. I am waiting for the announcement that it can bring you coffee in the morning and cook your tea. Just as the symptoms of 'Covid' seem to increase by the week so have the miracles of the 'vaccine'. American supermarket giant Kroger Co. offered nearly 500,000 employees in 35 states a $100 bonus for having the 'vaccine' while donut chain Krispy Kreme promised 'vaccinated' customers a free glazed donut every day for the rest of 2021. Have your DNA changed and you will get a doughnut although we might not have to give you them for long. Such offers and incentives confirm the desperation.

Perhaps the worse vaccine-stunt of them all was UK 'Health' Secretary Matt-the-prat Hancock on live TV after watching a clip of someone being 'vaccinated' when the roll-out began. Hancock faked tears so badly it was embarrassing. Brain-of-Britain Piers Morgan, the lockdown-supporting, 'vaccine' supporting, 'vaccine' passport-supporting, TV host played along with Hancock – 'You're quite emotional about that' he said in response to acting so atrocious it would have been called out at a school nativity which will presumably today include Mary and Jesus in masks, wise men keeping their camels six feet apart, and shepherds under tent arrest. System-serving Morgan tweeted this: 'Love the idea of covid vaccine passports for everywhere: flights, restaurants, clubs, football, gyms,

shops etc. It's time covid-denying, anti-vaxxer loonies had their bullsh*t bluff called & bar themselves from going anywhere that responsible citizens go.' If only I could aspire to his genius. To think that Morgan, who specialises in shouting over anyone he disagrees with, was lauded as a free speech hero when he lost his job after storming off the set of his live show like a child throwing his dolly out of the pram. If he is a free speech hero we are in real trouble. I have no idea what 'bullsh*t' means, by the way, the * throws me completely.

The Cult is desperate to infuse its synthetic DNA-changing concoction into everyone and has been using every lie, trick and intimidation to do so. The question of '*Why*?' we shall now address.

CHAPTER TEN

Human 2.0

I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted –
Alan Turing (1912-1954), the 'Father of artificial intelligence'

I have been exposing for decades the plan to transform the human body from a biological to a synthetic-biological state. The new human that I will call Human 2.0 is planned to be connected to artificial intelligence and a global AI 'Smart Grid' that would operate as one global system in which AI would control everything from your fridge to your heating system to your car to your mind. Humans would no longer be 'human', but post-human and sub-human, with their thinking and emotional processes replaced by AI.

What I said sounded crazy and beyond science fiction and I could understand that. To any balanced, rational, mind it *is* crazy. Today, however, that world is becoming reality and it puts the 'Covid vaccine' into its true context. Ray Kurzweil is the ultra-Zionist 'computer scientist, inventor and futurist' and co-founder of the Singularity University. Singularity refers to the merging of humans with machines or 'transhumanism'. Kurzweil has said humanity would be connected to the cyber 'cloud' in the period of the ever-recurring year of 2030:

Our thinking … will be a hybrid of biological and non-biological thinking … humans will be able to extend their limitations and 'think in the cloud' … We're going to put gateways to the

cloud in our brains ... We're going to gradually merge and enhance ourselves ... In my view, that's the nature of being human – we transcend our limitations. As the technology becomes vastly superior to what we are then the small proportion that is still human gets smaller and smaller and smaller until it's just utterly negligible.

They are trying to sell this end-of-humanity-as-we-know-it as the next stage of 'evolution' when we become super-human and 'like the gods'. They are lying to you. Shocked, eh? The population, and again especially the young, have been manipulated into addiction to technologies designed to enslave them for life. First they induced an addiction to smartphones (holdables); next they moved to technology on the body (wearables); and then began the invasion of the body (implantables). I warned way back about the plan for microchipped people and we are now entering that era. We should not be diverted into thinking that this refers only to chips we can see. Most important are the nanochips known as smart dust, neural dust and nanobots which are far too small to be seen by the human eye. Nanotechnology is everywhere, increasingly in food products, and released into the atmosphere by the geoengineering of the skies funded by Bill Gates to 'shut out the Sun' and 'save the planet from global warming'. Gates has been funding a project to spray millions of tonnes of chalk (calcium carbonate) into the stratosphere over Sweden to 'dim the Sun' and cool the Earth. Scientists warned the move could be disastrous for weather systems in ways no one can predict and opposition led to the Swedish space agency announcing that the 'experiment' would not be happening as planned in the summer of 2021; but it shows where the Cult is going with dimming the impact of the Sun and there's an associated plan to change the planet's atmosphere. Who gives psychopath Gates the right to dictate to the entire human race and dismantle planetary systems? The world will not be safe while this man is at large.

The global warming hoax has made the Sun, like the gas of life, something to fear when both are essential to good health and human survival (more inversion). The body transforms sunlight into vital vitamin D through a process involving … *cholesterol*. This is the cholesterol we are also told to fear. We are urged to take Big Pharma

statin drugs to reduce cholesterol and it's all systematic. Reducing cholesterol means reducing vitamin D uptake with all the multiple health problems that will cause. At least if you take statins long term it saves the government from having to pay you a pension. The delivery system to block sunlight is widely referred to as chemtrails although these have a much deeper agenda, too. They appear at first to be contrails or condensation trails streaming from aircraft into cold air at high altitudes. Contrails disperse very quickly while chemtrails do not and spread out across the sky before eventually their content falls to earth. Many times I have watched aircraft cross-cross a clear blue sky releasing chemtrails until it looks like a cloudy day. Chemtrails contain many things harmful to humans and the natural world including toxic heavy metals, aluminium (see Alzheimer's) and nanotechnology. Ray Kurzweil reveals the reason without actually saying so: 'Nanobots will infuse all the matter around us with information. Rocks, trees, everything will become these intelligent creatures.' How do you deliver that? *From the sky*. Self-replicating nanobots would connect everything to the Smart Grid. The phenomenon of Morgellons disease began in the chemtrail era and the correlation has led to it being dubbed the 'chemtrail disease'. Self-replicating fibres appear in the body that can be pulled out through the skin. Morgellons fibres continue to grow outside the body and have a form of artificial intelligence. I cover this at greater length in *Phantom Self*.

'Vaccine' operating system

'Covid vaccines' with their self-replicating synthetic material are also designed to make the connection between humanity and Kurzweil's 'cloud'. American doctor and dedicated campaigner for truth, Carrie Madej, an Internal Medicine Specialist in Georgia with more than 20 years medical experience, has highlighted the nanotechnology aspect of the fake 'vaccines'. She explains how one of the components in at least the Moderna and Pfizer synthetic potions are 'lipid nanoparticles' which are 'like little tiny computer bits' – a 'sci-fi substance' known as nanobots and hydrogel which can be 'triggered

at any moment to deliver its payload' and act as 'biosensors'. The synthetic substance had 'the ability to accumulate data from your body like your breathing, your respiration, thoughts and emotions, all kind of things' and each syringe could carry a *million* nanobots:

This substance because it's like little bits of computers in your body, crazy, but it's true, it can do that, [and] obviously has the ability to act through Wi-Fi. It can receive and transmit energy, messages, frequencies or impulses. That issue has never been addressed by these companies. What does that do to the human?

Just imagine getting this substance in you and it can react to things all around you, the 5G, your smart device, your phones, what is happening with that? What if something is triggering it, too, like an impulse, a frequency? We have something completely foreign in the human body.

Madej said her research revealed that electromagnetic (EMF) frequencies emitted by phones and other devices had increased dramatically in the same period of the 'vaccine' rollout and she was seeing more people with radiation problems as 5G and other electromagnetic technology was expanded and introduced to schools and hospitals. She said she was 'floored with the EMF coming off' the devices she checked. All this makes total sense and syncs with my own work of decades when you think that Moderna refers in documents to its mRNA 'vaccine' as an 'operating system':

Recognizing the broad potential of mRNA science, we set out to create an mRNA technology platform that functions very much like an operating system on a computer. It is designed so that it can plug and play interchangeably with different programs. In our case, the 'program' or 'app' is our mRNA drug – the unique mRNA sequence that codes for a protein …

… Our MRNA Medicines – 'The 'Software Of Life': When we have a concept for a new mRNA medicine and begin research, fundamental components are already in place. Generally, the only thing that changes from one potential mRNA medicine to another is the coding region – the actual genetic code that instructs ribosomes to make protein. Utilizing these instruction sets gives our investigational mRNA medicines a software-like quality. We also have the ability to combine different mRNA sequences encoding for different proteins in a single mRNA investigational medicine.

Who needs a real 'virus' when you can create a computer version to justify infusing your operating system into the entire human race on the road to making living, breathing people into cyborgs? What is missed with the 'vaccines' is the *digital* connection between synthetic material and the body that I highlighted earlier with the study that hacked a computer with human DNA. On one level the body is digital, based on mathematical codes, and I'll have more about that in the next chapter. Those who ridiculously claim that mRNA 'vaccines' are not designed to change human genetics should explain the words of Dr Tal Zaks, chief medical officer at Moderna, in a 2017 TED talk. He said that over the last 30 years 'we've been living this phenomenal digital scientific revolution, and I'm here today to tell you, that we are actually *hacking the software of life*, and that it's changing the way we think about prevention and treatment of disease':

In every cell there's this thing called messenger RNA, or mRNA for short, that transmits the critical information from the DNA in our genes to the protein, which is really the stuff we're all made out of. This is the critical information that determines what the cell will do. So we think about it as an operating system. So if you could change that, if you could introduce a line of code, or change a line of code, it turns out, that has profound implications for everything, from the flu to cancer.

Zaks should more accurately have said that this has profound implications for the human genetic code and the nature of DNA. Communications within the body go both ways and not only one. But, hey, no, the 'Covid vaccine' will not affect your genetics. Cult fact-checkers say so even though the man who helped to develop the mRNA technique says that it does. Zaks said in 2017:

If you think about what it is we're trying to do. We've taken information and our understanding of that information and how that information is transmitted in a cell, and we've taken our understanding of medicine and how to make drugs, and we're fusing the two. We think of it as information therapy.

I have been writing for decades that the body is an information field communicating with itself and the wider world. This is why

radiation which is information can change the information field of body and mind through phenomena like 5G and change their nature and function. 'Information therapy' means to change the body's information field and change the way it operates. DNA is a receiver-transmitter of information and can be mutated by information like mRNA synthetic messaging. Technology to do this has been ready and waiting in the underground bases and other secret projects to be rolled out when the 'Covid' hoax was played. 'Trials' of such short and irrelevant duration were only for public consumption. When they say the 'vaccine' is 'experimental' that is not true. It may appear to be 'experimental' to those who don't know what's going on, but the trials have already been done to ensure the Cult gets the result it desires. Zaks said that it took decades to sequence the human genome, completed in 2003, but now they could do it in a week. By 'they' he means scientists operating in the public domain. In the secret projects they were sequencing the genome in a week long before even 2003.

Deluge of mRNA

Highly significantly the Moderna document says the guiding premise is that if using mRNA as a medicine works for one disease then it should work for many diseases. They were leveraging the flexibility afforded by their platform and the fundamental role mRNA plays in protein synthesis to pursue mRNA medicines for a broad spectrum of diseases. Moderna is confirming what I was saying through 2020 that multiple 'vaccines' were planned for 'Covid' (and later invented 'variants') and that previous vaccines would be converted to the mRNA system to infuse the body with massive amounts of genetically-manipulating synthetic material to secure a transformation to a synthetic-biological state. The 'vaccines' are designed to kill stunning numbers as part of the long-exposed Cult depopulation agenda and transform the rest. Given this is the goal you can appreciate why there is such hysterical demand for every human to be 'vaccinated' for an alleged 'disease' that has an estimated 'infection' to 'death' ratio of 0.23-0.15 percent. As I write

children are being given the 'vaccine' in trials (their parents are a disgrace) and ever-younger people are being offered the vaccine for a 'virus' that even if you believe it exists has virtually zero chance of harming them. Horrific effects of the 'trials' on a 12-year-old girl were revealed by a family member to be serious brain and gastric problems that included a bowel obstruction and the inability to swallow liquids or solids. She was unable to eat or drink without throwing up, had extreme pain in her back, neck and abdomen, and was paralysed from the waist down which stopped her urinating unaided. When the girl was first taken to hospital doctors said it was all in her mind. She was signed up for the 'trial' by her parents for whom no words suffice. None of this 'Covid vaccine' insanity makes any sense unless you see what the 'vaccine' really is – a body-changer. Synthetic biology or 'SynBio' is a fast-emerging and expanding scientific discipline which includes everything from genetic and molecular engineering to electrical and computer engineering. Synthetic biology is defined in these ways:

- A multidisciplinary area of research that seeks to create new biological parts, devices, and systems, or to redesign systems that are already found in nature.

- The use of a mixture of physical engineering and genetic engineering to create new (and therefore synthetic) life forms.

- An emerging field of research that aims to combine the knowledge and methods of biology, engineering and related disciplines in the design of chemically-synthesized DNA to create organisms with novel or enhanced characteristics and traits (synthetic organisms including humans).

We now have synthetic blood, skin, organs and limbs being developed along with synthetic body parts produced by 3D printers. These are all elements of the synthetic human programme and this comment by Kurzweil's co-founder of the Singularity University,

Peter Diamandis, can be seen in a whole new light with the 'Covid' hoax and the sanctions against those that refuse the 'vaccine':

Anybody who is going to be resisting the progress forward [to transhumanism] is going to be resisting evolution and, fundamentally, they will die out. It's not a matter of whether it's good or bad. It's going to happen.

'Resisting evolution'? What absolute bollocks. The arrogance of these people is without limit. His 'it's going to happen' mantra is another way of saying 'resistance is futile' to break the spirit of those pushing back and we must not fall for it. Getting this genetically-transforming 'vaccine' into everyone is crucial to the Cult plan for total control and the desperation to achieve that is clear for anyone to see. Vaccine passports are a major factor in this and they, too, are a form of resistance is futile. It's NOT. The paper funded by the Rockefeller Foundation for the 2013 'health conference' in China said:

We will interact more with artificial intelligence. The use of robotics, bio-engineering to augment human functioning is already well underway and will advance. Re-engineering of humans into potentially separate and unequal forms through genetic engineering or mixed human-robots raises debates on ethics and equality.

A new demography is projected to emerge after 2030 [that year again] of technologies (robotics, genetic engineering, nanotechnology) producing robots, engineered organisms, 'nanobots' and artificial intelligence (AI) that can self-replicate. Debates will grow on the implications of an impending reality of human designed life.

What is happening today is so long planned. The world army enforcing the will of the world government is intended to be a robot army, not a human one. Today's military and its technologically 'enhanced' troops, pilotless planes and driverless vehicles are just stepping stones to that end. Human soldiers are used as Cult fodder and its time they woke up to that and worked for the freedom of the population instead of their own destruction and their family's destruction – the same with the police. Join us and let's sort this out. The phenomenon of enforce my own destruction is widespread in the 'Covid' era with Woker 'luvvies' in the acting and entertainment

industries supporting 'Covid' rules which have destroyed their profession and the same with those among the public who put signs on the doors of their businesses 'closed due to Covid – stay safe' when many will never reopen. It's a form of masochism and most certainly insanity.

Transgender = transhumanism

When something explodes out of nowhere and is suddenly everywhere it is always the Cult agenda and so it is with the tidal wave of claims and demands that have infiltrated every aspect of society under the heading of 'transgenderism'. The term 'trans' is so 'in' and this is the dictionary definition:

> A prefix meaning 'across', 'through', occurring … in loanwords from Latin, used in particular for denoting movement or conveyance from place to place (transfer; transmit; transplant) or complete change (transform; transmute), or to form adjectives meaning 'crossing', 'on the other side of', or 'going beyond' the place named (transmontane; transnational; trans-Siberian).

Transgender means to go beyond gender and transhuman means to go beyond human. Both are aspects of the Cult plan to transform the human body to a synthetic state with *no gender*. Human 2.0 is not designed to procreate and would be produced technologically with no need for parents. The new human would mean the end of parents and so men, and increasingly women, are being targeted for the deletion of their rights and status. Parental rights are disappearing at an ever-quickening speed for the same reason. The new human would have no need for men or women when there is no procreation and no gender. Perhaps the transgender movement that appears to be in a permanent state of frenzy might now contemplate on how it is being used. This was never about transgender rights which are only the interim excuse for confusing gender, particularly in the young, on the road to *fusing* gender. Transgender activism is not an end; it is a *means* to an end. We see again the technique of creative destruction in which you destroy the status quo to 'build back better' in the form that you want. The gender status quo had to be

destroyed by persuading the Cult-created Woke mentality to believe that you can have 100 genders or more. A programme for 9 to 12 year olds produced by the Cult-owned BBC promoted the 100 genders narrative. The very idea may be the most monumental nonsense, but it is not what is true that counts, only what you can make people *believe* is true. Once the gender of 2 + 2 = 4 has been dismantled through indoctrination, intimidation and 2 + 2 = 5 then the new no-gender normal can take its place with Human 2.0. Aldous Huxley revealed the plan in his prophetic *Brave New World* in 1932:

> Natural reproduction has been done away with and children are created, decanted', and raised in 'hatcheries and conditioning centres'. From birth, people are genetically designed to fit into one of five castes, which are further split into 'Plus' and 'Minus' members and designed to fulfil predetermined positions within the social and economic strata of the World State.

How could Huxley know this in 1932? For the same reason George Orwell knew about the Big Brother state in 1948, Cult insiders I have quoted knew about it in 1969, and I have known about it since the early 1990s. If you are connected to the Cult or you work your balls off to uncover the plan you can predict the future. The process is simple. If there is a plan for the world and nothing intervenes to stop it then it will happen. Thus if you communicate the plan ahead of time you are perceived to have predicted the future, but you haven't. You have revealed the plan which without intervention will become the human future. The whole reason I have done what I have is to alert enough people to inspire an intervention and maybe at last that time has come with the Cult and its intentions now so obvious to anyone with a brain in working order.

The future is here

Technological wombs that Huxley described to replace parent procreation are already being developed and they are only the projects we know about in the public arena. Israeli scientists told *The Times of Israel* in March, 2021, that they have grown 250-cell embryos

into mouse foetuses with fully formed organs using artificial wombs in a development they say could pave the way for gestating humans outside the womb. Professor Jacob Hanna of the Weizmann Institute of Science said:

> We took mouse embryos from the mother at day five of development, when they are just of 250 cells, and had them in the incubator from day five until day 11, by which point they had grown all their organs.
>
> By day 11 they make their own blood and have a beating heart, a fully developed brain. Anybody would look at them and say, 'this is clearly a mouse foetus with all the characteristics of a mouse.' It's gone from being a ball of cells to being an advanced foetus.

A special liquid is used to nourish embryo cells in a laboratory dish and they float on the liquid to duplicate the first stage of embryonic development. The incubator creates all the right conditions for its development, Hanna said. The liquid gives the embryo 'all the nutrients, hormones and sugars they need' along with a custom-made electronic incubator which controls gas concentration, pressure and temperature. The cutting-edge in the underground bases and other secret locations will be light years ahead of that, however, and this was reported by the London *Guardian* in 2017:

> We are approaching a biotechnological breakthrough. Ectogenesis, the invention of a complete external womb, could completely change the nature of human reproduction. In April this year, researchers at the Children's Hospital of Philadelphia announced their development of an artificial womb.

The article was headed 'Artificial wombs could soon be a reality. What will this mean for women?' What would it mean for children is an even bigger question. No mother to bond with only a machine in preparation for a life of soulless interaction and control in a world governed by machines (see the *Matrix* movies). Now observe the calculated manipulations of the 'Covid' hoax as human interaction and warmth has been curtailed by distancing, isolation and fear with people communicating via machines on a scale never seen before.

These are all dots in the same picture as are all the personal assistants, gadgets and children's toys through which kids and adults communicate with AI as if it is human. The AI 'voice' on Sat-Nav should be included. All these things are psychological preparation for the Cult endgame. Before you can make a physical connection with AI you have to make a psychological connection and that is what people are being conditioned to do with this ever gathering human-AI interaction. Movies and TV programmes depicting the transhuman, robot dystopia relate to a phenomenon known as 'pre-emptive programming' in which the world that is planned is portrayed everywhere in movies, TV and advertising. This is conditioning the conscious and subconscious mind to become familiar with the planned reality to dilute resistance when it happens for real. What would have been a shock such is the change is made less so. We have young children put on the road to transgender transition surgery with puberty blocking drugs at an age when they could never be able to make those life-changing decisions.

Rachel Levine, a professor of paediatrics and psychiatry who believes in treating children this way, became America's highest-ranked openly-transgender official when she was confirmed as US Assistant Secretary at the Department of Health and Human Services after being nominated by Joe Biden (the Cult). Activists and governments press for laws to deny parents a say in their children's transition process so the kids can be isolated and manipulated into agreeing to irreversible medical procedures. A Canadian father Robert Hoogland was denied bail by the Vancouver Supreme Court in 2021 and remained in jail for breaching a court order that he stay silent over his young teenage daughter, a minor, who was being offered life-changing hormone therapy without parental consent. At the age of 12 the girl's 'school counsellor' said she may be transgender, referred her to a doctor and told the school to treat her like a boy. This is another example of state-serving schools imposing ever more control over children's lives while parents have ever less.

Contemptible and extreme child abuse is happening all over the world as the Cult gender-fusion operation goes into warp-speed.

Why the war on men – and now women?

The question about what artificial wombs mean for women should rightly be asked. The answer can be seen in the deletion of women's rights involving sport, changing rooms, toilets and status in favour of people in male bodies claiming to identify as women. I can identify as a mountain climber, but it doesn't mean I can climb a mountain any more than a biological man can be a biological woman. To believe so is a triumph of belief over factual reality which is the very perceptual basis of everything Woke. Women's sport is being destroyed by allowing those with male bodies who say they identify as female to 'compete' with girls and women. Male body 'women' dominate 'women's' competition with their greater muscle mass, bone density, strength and speed. With that disadvantage sport for women loses all meaning. To put this in perspective nearly 300 American high school boys can run faster than the quickest woman sprinter in the world. Women are seeing their previously protected spaces invaded by male bodies simply because they claim to identify as women. That's all they need to do to access all women's spaces and activities under the Biden 'Equality Act' that destroys equality for women with the usual Orwellian Woke inversion. Male sex offenders have already committed rapes in women's prisons after claiming to identify as women to get them transferred. Does this not matter to the Woke 'equality' hypocrites? Not in the least. What matters to Cult manipulators and funders behind transgender activists is to advance gender fusion on the way to the no-gender 'human'. When you are seeking to impose transparent nonsense like this, or the 'Covid' hoax, the only way the nonsense can prevail is through censorship and intimidation of dissenters, deletion of factual information, and programming of the unquestioning, bewildered and naive. You don't have to scan the world for long to see that all these things are happening.

Many women's rights organisations have realised that rights and status which took such a long time to secure are being eroded and that it is systematic. Kara Dansky of the global Women's Human Rights Campaign said that Biden's transgender executive order immediately he took office, subsequent orders, and Equality Act legislation that followed 'seek to erase women and girls in the law as a category'. *Exactly*. I said during the long ago-started war on men (in which many women play a crucial part) that this was going to turn into a war on them. The Cult is phasing out *both* male and female genders. To get away with that they are brought into conflict so they are busy fighting each other while the Cult completes the job with no unity of response. Unity, people, *unity*. We need unity everywhere. Transgender is the only show in town as the big step towards the no-gender human. It's not about rights for transgender people and never has been. Woke political correctness is deleting words relating to genders to the same end. Wokers believe this is to be 'inclusive' when the opposite is true. They are deleting words describing gender because gender *itself* is being deleted by Human 2.0. Terms like 'man', 'woman', 'mother' and 'father' are being deleted in the universities and other institutions to be replaced by the *no*-gender, not trans-gender, 'individuals' and 'guardians'. Women's rights campaigner Maria Keffler of Partners for Ethical Care said: 'Children are being taught from kindergarten upward that some boys have a vagina, some girls have a penis, and that kids can be any gender they want to be.' Do we really believe that suddenly countries all over the world at the same time had the idea of having drag queens go into schools or read transgender stories to very young children in the local library? It's coldly-calculated confusion of gender on the way to the fusion of gender. Suzanne Vierling, a psychologist from Southern California, made another important point:

> Yesterday's slave woman who endured gynecological medical experiments is today's girl-child being butchered in a booming gender-transitioning sector. Ovaries removed, pushing her into menopause and osteoporosis, uncharted territory, and parents' rights and authority decimated.

The erosion of parental rights is a common theme in line with the Cult plans to erase the very concept of parents and 'ovaries removed, pushing her into menopause' means what? Those born female lose the ability to have children – another way to discontinue humanity as we know it.

Eliminating Human 1.0 (before our very eyes)

To pave the way for Human 2.0 you must phase out Human 1.0. This is happening through plummeting sperm counts and making women infertile through an onslaught of chemicals, radiation (including smartphones in pockets of men) and mRNA 'vaccines'. Common agriculture pesticides are also having a devastating impact on human fertility. I have been tracking collapsing sperm counts in the books for a long time and in 2021 came a book by fertility scientist and reproductive epidemiologist Shanna Swan, *Count Down: How Our Modern World Is Threatening Sperm Counts, Altering Male and Female Reproductive Development and Imperiling the Future of the Human Race*. She reports how the global fertility rate dropped by *half* between 1960 and 2016 with America's birth rate 16 percent below where it needs to be to sustain the population. Women are experiencing declining egg quality, more miscarriages, and more couples suffer from infertility. Other findings were an increase in erectile dysfunction, infant boys developing more genital abnormalities, male problems with conception, and plunging levels of the male hormone testosterone which would explain why so many men have lost their backbone and masculinity. This has been very evident during the 'Covid' hoax when women have been prominent among the Pushbackers and big strapping blokes have bowed their heads, covered their faces with a nappy and quietly submitted. Mind control expert Cathy O'Brien also points to how global education introduced the concept of 'we're all winners' in sport and classrooms: 'Competition was defused, and it in turn defused a sense of fighting back.' This is another version of the 'equity' doctrine in which you drive down rather than raise up. What a contrast in Cult-controlled China with its global ambitions

where the government published plans in January, 2021, to 'cultivate masculinity' in boys from kindergarten through to high school in the face of a 'masculinity crisis'. A government adviser said boys would be soon become 'delicate, timid and effeminate' unless action was taken. Don't expect any similar policy in the targeted West. A 2006 study showed that a 65-year-old man in 2002 had testosterone levels *15 percent* lower than a 65-year-old man in 1987 while a 2020 study found a similar story with young adults and adolescents. Men are getting prescriptions for testosterone replacement therapy which causes an even greater drop in sperm count with up to 99 percent seeing sperm counts drop to zero during the treatment. More sperm is defective and malfunctioning with some having two heads or not pursuing an egg.

A class of *synthetic* chemicals known as phthalates are being blamed for the decline. These are found everywhere in plastics, shampoos, cosmetics, furniture, flame retardants, personal care products, pesticides, canned foods and even receipts. Why till receipts? Everyone touches them. Let no one delude themselves that all this is not systematic to advance the long-time agenda for human body transformation. Phthalates mimic hormones and disrupt the hormone balance causing testosterone to fall and genital birth defects in male infants. Animals and fish have been affected in the same way due to phthalates and other toxins in rivers. When fish turn gay or change sex through chemicals in rivers and streams it is a pointer to why there has been such an increase in gay people and the sexually confused. It doesn't matter to me what sexuality people choose to be, but if it's being affected by chemical pollution and consumption then we need to know. Does anyone really think that this is not connected to the transgender agenda, the war on men and the condemnation of male 'toxic masculinity'? You watch this being followed by 'toxic femininity'. It's already happening. When breastfeeding becomes 'chest-feeding', pregnant women become pregnant people along with all the other Woke claptrap you know that the world is going insane and there's a Cult scam in progress. Transgender activists are promoting the Cult agenda while Cult

billionaires support and fund the insanity as they laugh themselves to sleep at the sheer stupidity for which humans must be infamous in galaxies far, far away.

'Covid vaccines' and female infertility

We can now see why the 'vaccine' has been connected to potential infertility in women. Dr Michael Yeadon, former Vice President and Chief Scientific Advisor at Pfizer, and Dr Wolfgang Wodarg in Germany, filed a petition with the European Medicines Agency in December, 2020, urging them to stop trials for the Pfizer/BioNTech shot and all other mRNA trials until further studies had been done. They were particularly concerned about possible effects on fertility with 'vaccine'-produced antibodies attacking the protein Syncytin-1 which is responsible for developing the placenta. The result would be infertility 'of indefinite duration' in women who have the 'vaccine' with the placenta failing to form. Section 10.4.2 of the Pfizer/BioNTech trial protocol says that pregnant women or those who might become so should not have mRNA shots. Section 10.4 warns men taking mRNA shots to 'be abstinent from heterosexual intercourse' and not to donate sperm. The UK government said that it *did not know* if the mRNA procedure had an effect on fertility. *Did not know?* These people have to go to jail. UK government advice did not recommend at the start that pregnant women had the shot and said they should avoid pregnancy for at least two months after 'vaccination'. The 'advice' was later updated to pregnant women should only have the 'vaccine' if the benefits outweighed the risks to mother and foetus. What the hell is that supposed to mean? Then 'spontaneous abortions' began to appear and rapidly increase on the adverse reaction reporting schemes which include only a fraction of adverse reactions. Thousands and ever-growing numbers of 'vaccinated' women are describing changes to their menstrual cycle with heavier blood flow, irregular periods and menstruating again after going through the menopause – all links to reproduction effects. Women are passing blood clots and the lining of their uterus while men report erectile dysfunction and blood effects. Most

significantly of all *un*vaccinated women began to report similar menstrual changes after interaction with *'vaccinated'* people and men and children were also affected with bleeding noses, blood clots and other conditions. 'Shedding' is when vaccinated people can emit the content of a vaccine to affect the unvaccinated, but this is different. 'Vaccinated' people were not shedding a 'live virus' allegedly in 'vaccines' as before because the fake 'Covid vaccines' involve synthetic material and other toxicity. Doctors exposing what is happening prefer the term 'transmission' to shedding. Somehow those that have had the shots are transmitting effects to those that haven't. Dr Carrie Madej said the nano-content of the 'vaccines' can 'act like an antenna' to others around them which fits perfectly with my own conclusions. This 'vaccine' transmission phenomenon was becoming known as the book went into production and I deal with this further in the Postscript.

Vaccine effects on sterility are well known. The World Health Organization was accused in 2014 of sterilising millions of women in Kenya with the evidence confirmed by the content of the vaccines involved. The same WHO behind the 'Covid' hoax admitted its involvement for more than ten years with the vaccine programme. Other countries made similar claims. Charges were lodged by Tanzania, Nicaragua, Mexico, and the Philippines. The Gardasil vaccine claimed to protect against a genital 'virus' known as HPV has also been linked to infertility. Big Pharma and the WHO (same thing) are criminal and satanic entities. Then there's the Bill Gates Foundation which is connected through funding and shared interests with 20 pharmaceutical giants and laboratories. He stands accused of directing the policy of United Nations Children's Fund (UNICEF), vaccine alliance GAVI, and other groupings, to advance the vaccine agenda and silence opposition at great cost to women and children. At the same time Gates wants to reduce the global population. Coincidence?

Great Reset = Smart Grid = new human

The Cult agenda I have been exposing for 30 years is now being openly promoted by Cult assets like Gates and Klaus Schwab of the World Economic Forum under code-terms like the 'Great Reset', 'Build Back Better' and 'a rare but narrow window of opportunity to reflect, reimagine, and reset our world'. What provided this 'rare but narrow window of opportunity'? The 'Covid' hoax did. Who created that? *They* did. My books from not that long ago warned about the planned 'Internet of Things' (IoT) and its implications for human freedom. This was the plan to connect all technology to the Internet and artificial intelligence and today we are way down that road with an estimated 36 billion devices connected to the World Wide Web and that figure is projected to be 76 billion by 2025. I further warned that the Cult planned to go beyond that to the Internet of *Everything* when the human brain was connected via AI to the Internet and Kurzweil's 'cloud'. Now we have Cult operatives like Schwab calling for precisely that under the term 'Internet of Bodies', a fusion of the physical, digital and biological into one centrally-controlled Smart Grid system which the Cult refers to as the 'Fourth Industrial Revolution'. They talk about the 'biological', but they really mean the synthetic-biological which is required to fully integrate the human body and brain into the Smart Grid and artificial intelligence planned to replace the human mind. We have everything being synthetically manipulated including the natural world through GMO and smart dust, the food we eat and the human body itself with synthetic 'vaccines'. I said in *The Answer* that we would see the Cult push for synthetic meat to replace animals and in February, 2021, the so predictable psychopath Bill Gates called for the introduction of synthetic meat to save us all from 'climate change'. The climate hoax just keeps on giving like the 'Covid' hoax. The war on meat by vegan activists is a carbon (oops, sorry) copy of the manipulation of transgender activists. They have no idea (except their inner core) that they are being used to promote and impose the agenda of the Cult or that they are only the *vehicle* and not the *reason*. This is not to say those who choose not to eat meat shouldn't be respected and supported in that right, but there are ulterior motives

for those in power. A *Forbes* article in December, 2019, highlighted the plan so beloved of Schwab and the Cult under the heading: 'What Is The Internet of Bodies? And How Is It Changing Our World?' The article said the human body is the latest data platform (remember 'our vaccine is an operating system'). *Forbes* described the plan very accurately and the words could have come straight out of my books from long before:

> The Internet of Bodies (IoB) is an extension of the IoT and basically connects the human body to a network through devices that are ingested, implanted, or connected to the body in some way. Once connected, data can be exchanged, and the body and device can be remotely monitored and controlled.

They were really describing a human hive mind with human perception centrally-dictated via an AI connection as well as allowing people to be 'remotely monitored and controlled'. Everything from a fridge to a human mind could be directed from a central point by these insane psychopaths and 'Covid vaccines' are crucial to this. Forbes explained the process I mentioned earlier of holdable and wearable technology followed by implantable. The article said there were three generations of the Internet of Bodies that include:

- Body external: These are wearable devices such as Apple Watches or Fitbits that can monitor our health.

- Body internal: These include pacemakers, cochlear implants, and digital pills that go inside our bodies to monitor or control various aspects of health.

- Body embedded: The third generation of the Internet of Bodies is embedded technology where technology and the human body are melded together and have a real-time connection to a remote machine.

Forbes noted the development of the Brain Computer Interface (BCI) which merges the brain with an external device for monitoring and controlling in real-time. 'The ultimate goal is to help restore function to individuals with disabilities by using brain signals rather than conventional neuromuscular pathways.' Oh, do fuck off. The goal of brain interface technology is controlling human thought and emotion from the central point in a hive mind serving its masters wishes. Many people are now agreeing to be chipped to open doors without a key. You can recognise them because they'll be wearing a mask, social distancing and lining up for the 'vaccine'. The Cult plans a Great Reset money system after they have completed the demolition of the global economy in which 'money' will be exchanged through communication with body operating systems. Rand Corporation, a Cult-owned think tank, said of the Internet of Bodies or IoB:

> Internet of Bodies technologies fall under the broader IoT umbrella. But as the name suggests, IoB devices introduce an even more intimate interplay between humans and gadgets. IoB devices monitor the human body, collect health metrics and other personal information, and transmit those data over the Internet. Many devices, such as fitness trackers, are already in use … IoB devices … and those in development can track, record, and store users' whereabouts, bodily functions, and what they see, hear, and even think.

Schwab's World Economic Forum, a long-winded way of saying 'fascism' or 'the Cult', has gone full-on with the Internet of Bodies in the 'Covid' era. 'We're entering the era of the Internet of Bodies', it declared, 'collecting our physical data via a range of devices that can be implanted, swallowed or worn'. The result would be a huge amount of health-related data that could improve human wellbeing around the world, and prove crucial in fighting the 'Covid-19 pandemic'. Does anyone think these clowns care about 'human wellbeing' after the death and devastation their pandemic hoax has purposely caused? Schwab and co say we should move forward with the Internet of Bodies because 'Keeping track of symptoms could help us stop the spread of infection, and quickly detect new cases'. How wonderful, but keeping track' is all they are really bothered

about. Researchers were investigating if data gathered from smartwatches and similar devices could be used as viral infection alerts by tracking the user's heart rate and breathing. Schwab said in his 2018 book *Shaping the Future of the Fourth Industrial Revolution*:

> The lines between technologies and beings are becoming blurred and not just by the ability to create lifelike robots or synthetics. Instead it is about the ability of new technologies to literally become part of us. Technologies already influence how we understand ourselves, how we think about each other, and how we determine our realities. As the technologies … give us deeper access to parts of ourselves, we may begin to integrate digital technologies into our bodies.

You can see what the game is. Twenty-four hour control and people – if you could still call them that – would never know when something would go ping and take them out of circulation. It's the most obvious rush to a global fascist dictatorship and the complete submission of humanity and yet still so many are locked away in their Cult-induced perceptual coma and can't see it.

Smart Grid control centres

The human body is being transformed by the 'vaccines' and in other ways into a synthetic cyborg that can be attached to the global Smart Grid which would be controlled from a central point and other sub-locations of Grid manipulation. Where are these planned to be? Well, China for a start which is one of the Cult's biggest centres of operation. The technological control system and technocratic rule was incubated here to be unleashed across the world after the 'Covid' hoax came out of China in 2020. Another Smart Grid location that will surprise people new to this is Israel. I have exposed in *The Trigger* how Sabbatian technocrats, intelligence and military operatives were behind the horrors of 9/11 and not '19 Arab hijackers' who somehow manifested the ability to pilot big passenger airliners when instructors at puddle-jumping flying schools described some of them as a joke. The 9/11 attacks were made possible through control of civilian and military air computer systems and those of the White House, Pentagon and connected agencies. See *The Trigger* – it

will blow your mind. The controlling and coordinating force were the Sabbatian networks in Israel and the United States which by then had infiltrated the entire US government, military and intelligence system. The real name of the American Deep State is 'Sabbatian State'. Israel is a tiny country of only nine million people, but it is one of the global centres of cyber operations and fast catching Silicon Valley in importance to the Cult. Israel is known as the 'start-up nation' for all the cyber companies spawned there with the Sabbatian specialisation of 'cyber security' that I mentioned earlier which gives those companies access to computer systems of their clients in real time through 'backdoors' written into the coding when security software is downloaded. The Sabbatian centre of cyber operations outside Silicon Valley is the Israeli military Cyber Intelligence Unit, the biggest infrastructure project in Israel's history, headquartered in the desert-city of Beersheba and involving some 20,000 'cyber soldiers'. Here are located a literal army of Internet trolls scanning social media, forums and comment lists for anyone challenging the Cult agenda. The UK military has something similar with its 77th Brigade and associated operations. The Beersheba complex includes research and development centres for other Cult operations such as Intel, Microsoft, IBM, Google, Apple, Hewlett-Packard, Cisco Systems, Facebook and Motorola. Techcrunch.com ran an article about the Beersheba global Internet technology centre headlined 'Israel's desert city of Beersheba is turning into a cybertech oasis':

> The military's massive relocation of its prestigious technology units, the presence of multinational and local companies, a close proximity to Ben Gurion University and generous government subsidies are turning Beersheba into a major global cybertech hub. Beersheba has all of the ingredients of a vibrant security technology ecosystem, including Ben Gurion University with its graduate program in cybersecurity and Cyber Security Research Center, and the presence of companies such as EMC, Deutsche Telekom, PayPal, Oracle, IBM, and Lockheed Martin. It's also the future home of the INCB (Israeli National Cyber Bureau); offers a special income tax incentive for cyber security companies, and was the site for the relocation of the army's intelligence corps units.

Sabbatians have taken over the cyber world through the following process: They scan the schools for likely cyber talent and develop them at Ben Gurion University and their period of conscription in the Israeli Defense Forces when they are stationed at the Beersheba complex. When the cyber talented officially leave the army they are funded to start cyber companies with technology developed by themselves or given to them by the state. Much of this is stolen through backdoors of computer systems around the world with America top of the list. Others are sent off to Silicon Valley to start companies or join the major ones and so we have many major positions filled by apparently 'Jewish' but really Sabbatian operatives. Google, YouTube and Facebook are all run by 'Jewish' CEOs while Twitter is all but run by ultra-Zionist hedge-fund shark Paul Singer. At the centre of the Sabbatian global cyber web is the Israeli army's Unit 8200 which specialises in hacking into computer systems of other countries, inserting viruses, gathering information, instigating malfunction, and even taking control of them from a distance. A long list of Sabbatians involved with 9/11, Silicon Valley and Israeli cyber security companies are operatives of Unit 8200. This is not about Israel. It's about the Cult. Israel is planned to be a Smart Grid hub as with China and what is happening at Beersheba is not for the benefit of Jewish people who are treated disgustingly by the Sabbatian elite that control the country. A glance at the Nuremberg Codes will tell you that.

The story is much bigger than 'Covid', important as that is to where we are being taken. Now, though, it's time to really strap in. There's more … much more …

CHAPTER ELEVEN

Who controls the Cult?

Awake, arise or be forever fall'n
John Milton, *Paradise Lost*

I have exposed this far the level of the Cult conspiracy that operates in the world of the seen and within the global secret society and satanic network which operates in the shadows one step back from the seen. The story, however, goes much deeper than that.

The 'Covid' hoax is major part of the Cult agenda, but only part, and to grasp the biggest picture we have to expand our attention beyond the realm of human sight and into the infinity of possibility that we cannot see. It is from here, ultimately, that humanity is being manipulated into a state of total control by the force which dictates the actions of the Cult. How much of reality can we see? Next to damn all is the answer. We may appear to see all there is to see in the 'space' our eyes survey and observe, but little could be further from the truth. The human 'world' is only a tiny band of frequency that the body's visual and perceptual systems can decode into *perception* of a 'world'. According to mainstream science the electromagnetic spectrum is 0.005 percent of what exists in the Universe (Fig 10). The maximum estimate I have seen is 0.5 percent and either way it's miniscule. I say it is far, far, smaller even than 0.005 percent when you compare reality we see with the totality of reality that we don't. Now get this if you are new to such information: Visible light, the only band of frequency that we can see, is a *fraction* of the 0.005

percent (Fig 11 overleaf). Take this further and realise that our universe is one of infinite universes and that universes are only a fragment of overall reality – *infinite* reality. Then compare that with the almost infinitesimal frequency band of visible light or human sight. You see that humans are as near blind as it is possible to be without actually being so. Artist and filmmaker, Sergio Toporek, said:

Figure 10: Humans can perceive such a tiny band of visual reality it's laughable.

Figure 11: We can see a smear of the 0.005 percent electromagnetic spectrum, but we still know it all. Yep, makes sense.

Consider that you can see less than 1% of the electromagnetic spectrum and hear less than 1% of the acoustic spectrum. 90% of the cells in your body carry their own microbial DNA and are not 'you'. The atoms in your body are 99.9999999999999999% empty space and none of them are the ones you were born with ... Human beings have 46 chromosomes, two less than a potato.

The existence of the rainbow depends on the conical photoreceptors in your eyes; to animals without cones, the rainbow does not exist. So you don't just look at a rainbow, you create it. This is pretty amazing, especially considering that all the beautiful colours you see represent less than 1% of the electromagnetic spectrum.

Suddenly the 'world' of humans looks a very different place. Take into account, too, that Planet Earth when compared with the projected size of this single universe is the equivalent of a billionth of a pinhead. Imagine the ratio that would be when compared to infinite reality. To think that Christianity once insisted that Earth and humanity were the centre of everything. This background is vital if we are going to appreciate the nature of 'human' and how we can be manipulated by an unseen force. To human visual reality virtually *everything* is unseen and yet the prevailing perception within the institutions and so much of the public is that if we can't see it, touch it, hear it, taste it and smell it then it cannot exist. Such perception is indoctrinated and encouraged by the Cult and its agents because it isolates believers in the strictly limited, village-idiot, realm of the five senses where perceptions can be firewalled and information controlled. Most of those perpetuating the 'this-world-is-all-there-is' insanity are themselves indoctrinated into believing the same delusion. While major players and influencers know that official reality is laughable most of those in science, academia and medicine really believe the nonsense they peddle and teach succeeding generations. Those who challenge the orthodoxy are dismissed as nutters and freaks to protect the manufactured illusion from exposure. Observe the dynamic of the 'Covid' hoax and you will see how that takes the same form. The inner-circle psychopaths knows it's a gigantic scam, but almost the entirety of those imposing their fascist rules believe that 'Covid' is all that they're told it is.

Stolen identity

Ask people who they are and they will give you their name, place of birth, location, job, family background and life story. Yet that is not who they are – it is what they are *experiencing*. The difference is *absolutely crucial*. The true 'I', the eternal, infinite 'I', is consciousness,

a state of being aware. Forget 'form'. That is a vehicle for a brief experience. Consciousness does not come *from* the brain, but *through* the brain and even that is more symbolic than literal. We are awareness, pure awareness, and this is what withdraws from the body at what we call 'death' to continue our eternal beingness, *isness*, in other realms of reality within the limitlessness of infinity or the Biblical 'many mansions in my father's house'. Labels of a human life, man, woman, transgender, black, white, brown, nationality, circumstances and income are not who we are. They are what we are – awareness – is *experiencing* in a brief connection with a band of frequency we call 'human'. The labels are not the self; they are, to use the title of one of my books, a *Phantom* Self. I am not David Icke born in Leicester, England, on April 29th, 1952. I am the consciousness *having that experience*. The Cult and its non-human masters seek to convince us through the institutions of 'education', science, medicine, media and government that what we are *experiencing* is who we *are*. It's so easy to control and direct perception locked away in the bewildered illusions of the five senses with no expanded radar. Try, by contrast, doing the same with a humanity aware of its true self and its true power to consciously create its reality and experience. How is it possible to do this? We do it all day every day. If you perceive yourself as 'little me' with no power to impact upon your life and the world then your life experience will reflect that. You will hand the power you don't think you have to authority in all its forms which will use it to control your experience. This, in turn, will appear to confirm your perception of 'little me' in a self-fulfilling feedback loop. But that is what 'little me' really is – a *perception*. We are all 'big-me', infinite me, and the Cult has to make us forget that if its will is to prevail. We are therefore manipulated and pressured into self-identifying with human labels and not the consciousness/awareness *experiencing* those human labels.

The phenomenon of identity politics is a Cult-instigated manipulation technique to sub-divide previous labels into even smaller ones. A United States university employs this list of letters to

describe student identity: LGBTTQQFAGPBDSM or lesbian, gay, bisexual, transgender, transsexual, queer, questioning, flexual, asexual, gender-fuck, polyamorous, bondage/discipline, dominance/submission and sadism/masochism. I'm sure other lists are even longer by now as people feel the need to self-identity the 'I' with the minutiae of race and sexual preference. Wokers programmed by the Cult for generations believe this is about 'inclusivity' when it's really the Cult locking them away into smaller and smaller versions of Phantom Self while firewalling them from the influence of their true self, the infinite, eternal 'I'. You may notice that my philosophy which contends that we are all unique points of attention/awareness within the same infinite whole or Oneness is the ultimate non-racism. The very sense of Oneness makes the judgement of people by their body-type, colour or sexuality utterly ridiculous and confirms that racism has no understanding of reality (including anti-white racism). Yet despite my perception of life Cult agents and fast-asleep Wokers label me racist to discredit my information while they are themselves phenomenally racist and sexist. All they see is race and sexuality and they judge people as good or bad, demons or untouchables, by their race and sexuality. All they see is *Phantom Self* and perceive themselves in terms of Phantom Self. They are pawns and puppets of the Cult agenda to focus attention and self-identity in the five senses and play those identities against each other to divide and rule. Columbia University has introduced segregated graduations in another version of social distancing designed to drive people apart and teach them that different racial and cultural groups have nothing in common with each other. The last thing the Cult wants is unity. Again the pump-primers of this will be Cult operatives in the knowledge of what they are doing, but the rest are just the Phantom Self blind leading the Phantom Self blind. We *do* have something in common – we are all *the same consciousness* having different temporary experiences.

What is this 'human'?

Yes, what *is* 'human'? That is what we are supposed to be, right? I mean 'human'? True, but 'human' is the experience not the 'I'. Break it down to basics and 'human' is the way that information is processed. If we are to experience and interact with this band of frequency we call the 'world' we must have a vehicle that operates within that band of frequency. Our consciousness in its prime form cannot do that; it is way beyond the frequency of the human realm. My consciousness or awareness could not tap these keys and pick up the cup in front of me in the same way that radio station A cannot interact with radio station B when they are on different frequencies. The human body is the means through which we have that interaction. I have long described the body as a biological computer which processes information in a way that allows consciousness to experience this reality. The body is a receiver, transmitter and processor of information in a particular way that we call human. We visually perceive only the world of the five senses in a wakened state – that is the limit of the body's visual decoding system. In truth it's not even visual in the way we experience 'visual reality' as I will come to in a moment. We are 'human' because the body processes the information sources of human into a reality and behaviour system that we *perceive* as human. Why does an elephant act like an elephant and not like a human or a duck? The elephant's biological computer is a different information field and processes information according to that program into a visual and behaviour type we call an elephant. The same applies to everything in our reality. These body information fields are perpetuated through procreation (like making a copy of a software program). The Cult wants to break that cycle and intervene technologically to transform the human information field into one that will change what we call humanity. If it can change the human information field it will change the way that field processes information and change humanity both 'physically' and psychologically. Hence the *messenger* (information) RNA 'vaccines' and so much more that is targeting human genetics by changing the body's information – *messaging* – construct through food, drink, radiation, toxicity and other means.

Reality that we experience is nothing like reality as it really is in the same way that the reality people experience in virtual reality games is not the reality they are really living in. The game is only a decoded source of information that appears to be a reality. Our world is also an information construct – a *simulation* (more later). In its base form our reality is a wavefield of information much the same in theme as Wi-Fi. The five senses decode wavefield information into electrical information which they communicate to the brain to decode into holographic (illusory 'physical') information. Different parts of the brain specialise in decoding different senses and the information is fused into a reality that appears to be outside of us but is really inside the brain and the genetic structure in general (Fig 12 overleaf). DNA is a receiver-transmitter of information and a vital part of this decoding process and the body's connection to other realities. Change DNA and you change the way we decode and connect with reality – see 'Covid vaccines'. Think of computers decoding Wi-Fi. You have information encoded in a radiation field and the computer decodes that information into a very different form on the screen. You can't see the Wi-Fi until its information is made manifest on the screen and the information on the screen is inside the computer and not outside. I have just described how we decode the 'human world'. All five senses decode the waveform 'Wi-Fi' field into electrical signals and the brain (computer) constructs reality inside the brain and not outside – 'You don't just look at a rainbow, you create it'. Sound is a simple example. We don't hear sound until the brain decodes it. Waveform sound waves are picked up by the hearing sense and communicated to the brain in an electrical form to be decoded into the sounds that we hear. Everything we hear is inside the brain along with everything we see, feel, smell and taste. Words and language are waveform fields generated by our vocal chords which pass through this process until they are decoded by the brain into words that we hear. Different languages are different frequency fields or sound waves generated by vocal chords. Late British philosopher Alan Watts said:

Figure 12: The brain receives information from the five senses and constructs from that our perceived reality.

[Without the brain] the world is devoid of light, heat, weight, solidity, motion, space, time or any other imaginable feature. All these phenomena are interactions, or transactions, of vibrations with a certain arrangement of neurons.

That's exactly what they are and scientist Robert Lanza describes in his book, *Biocentrism*, how we decode electromagnetic waves and energy into visual and 'physical' experience. He uses the example of a flame emitting photons, electromagnetic energy, each pulsing electrically and magnetically:

… these … invisible electromagnetic waves strike a human retina, and if (and only if) the waves happen to measure between 400 and 700 nano meters in length from crest to crest, then their energy is just right to deliver a stimulus to the 8 million cone-shaped cells in the retina.

Each in turn send an electrical pulse to a neighbour neuron, and on up the line this goes, at 250 mph, until it reaches the … occipital lobe of the brain, in the back of the head. There, a cascading complex of neurons fire from the incoming stimuli, and we subjectively perceive this experience as a yellow brightness occurring in a place we have been conditioned to call the 'external world'.

You hear what you decode

If a tree falls or a building collapses they make no noise unless someone is there to decode the energetic waves generated by the disturbance into what we call sound. Does a falling tree make a noise? Only if you hear it – *decode* it. Everything in our reality is a frequency field of information operating within the overall 'Wi-Fi' field that I call The Field. A vibrational disturbance is generated in The Field by the fields of the falling tree or building. These disturbance waves are what we decode into the sound of them falling. If no one is there to do that then neither will make any noise. Reality is created by the observer – *decoder* – and the *perceptions* of the observer affect the decoding process. For this reason different people – different *perceptions* – will perceive the same reality or situation in a different way. What one may perceive as a nightmare another will see as an opportunity. The question of why the Cult is so focused on controlling human perception now answers itself. All experienced reality is the act of decoding and we don't experience Wi-Fi until it is decoded on the computer screen. The sight and sound of an Internet video is encoded in the Wi-Fi all around us, but we don't see or hear it until the computer decodes that information. Taste, smell and touch are all phenomena of the brain as a result of the same process. We don't taste, smell or feel anything except in the brain and there are pain relief techniques that seek to block the signal from the site of discomfort to the brain because if the brain doesn't decode that signal we don't feel pain. Pain is in the brain and only appears to be at the point of impact thanks to the feedback loop between them. We don't see anything until electrical information from the sight senses is decoded in an area at the back of the brain. If that area is damaged we can go blind when our eyes are perfectly okay. So why do we go blind if we damage an eye? We damage the information processing between the waveform visual information and the visual decoding area of the brain. If information doesn't reach the brain in a form it can decode then we can't see the visual reality that it represents. What's more the brain is decoding only a fraction of the information it receives and the rest is absorbed by the

sub-conscious mind. This explanation is from the science magazine, *Wonderpedia*:

> Every second, 11 million sensations crackle along these [brain] pathways ... The brain is confronted with an alarming array of images, sounds and smells which it rigorously filters down until it is left with a manageable list of around 40. Thus 40 sensations per second make up what we perceive as reality.

The 'world' is not what people are told to believe that is it and the inner circles of the Cult *know that*.

Illusory 'physical' reality

We can only see a smear of 0.005 percent of the Universe which is only one of a vast array of universes – 'mansions' – within infinite reality. Even then the brain decodes only 40 pieces of information ('sensations') from a potential *11 million* that we receive every second. Two points strike you from this immediately: The sheer breathtaking stupidity of believing we know anything so rigidly that there's nothing more to know; and the potential for these processes to be manipulated by a malevolent force to control the reality of the population. One thing I can say for sure with no risk of contradiction is that when you can perceive an almost indescribable fraction of infinite reality there is always more to know as in tidal waves of it. Ancient Greek philosopher Socrates was so right when he said that wisdom is to know how little we know. How obviously true that is when you think that we are experiencing a physical world of solidity that is neither physical nor solid and a world of apartness when everything is connected. Cult-controlled 'science' dismisses the so-called 'paranormal' and all phenomena related to that when the 'para'-normal is perfectly normal and explains the alleged 'great mysteries' which dumbfound scientific minds. There is a reason for this. A 'scientific mind' in terms of the mainstream is a material mind, a five-sense mind imprisoned in see it, touch it, hear it, smell it and taste it. Phenomena and happenings that can't be explained that way leave the 'scientific mind' bewildered and the rule is that if they

can't account for why something is happening then it can't, by definition, be happening. I beg to differ. Telepathy is thought waves passing through The Field (think wave disturbance again) to be decoded by someone able to connect with that wavelength (information). For example: You can pick up the thought waves of a friend at any distance and at the very least that will bring them to mind. A few minutes later the friend calls you. 'My god', you say, 'that's incredible – I was just thinking of you.' Ah, but *they* were thinking of *you* before they made the call and that's what you decoded. Native peoples not entrapped in five-sense reality do this so well it became known as the 'bush telegraph'. Those known as psychics and mediums (genuine ones) are doing the same only across dimensions of reality. 'Mind over matter' comes from the fact that matter and mind are the *same*. The state of one influences the state of the other. Indeed one *and* the other are illusions. They are aspects of the same field. Paranormal phenomena are all explainable so why are they still considered 'mysteries' or not happening? Once you go down this road of understanding you begin to expand awareness beyond the five senses and that's the nightmare for the Cult.

Figure 13: Holograms are not solid, but the best ones appear to be.

Figure 14: How holograms are created by capturing a waveform version of the subject image.

Holographic 'solidity'

Our reality is not solid, it is holographic. We are now well aware of holograms which are widely used today. Two-dimensional information is decoded into a three-dimensional reality that is not solid although can very much appear to be (Fig 13). Holograms are created with a laser divided into two parts. One goes directly onto a holographic photographic print ('reference beam') and the other takes a waveform image of the subject ('working beam') before being directed onto the print where it 'collides' with the other half of the laser (Fig 14). This creates a *waveform* interference pattern which contains the wavefield information of whatever is being photographed (Fig 15 overleaf). The process can be likened to dropping pebbles in a pond. Waves generated by each one spread out across the water to collide with the others and create a wave representation of where the stones fell and at what speed, weight and distance. A waveform interference pattern of a hologram is akin to the waveform information in The Field which the five senses decode into electrical signals to be decoded by the brain into a holographic illusory 'physical' reality. In the same way when a laser (think human attention) is directed at the waveform interference pattern a three-dimensional version of the subject is projected into apparently 'solid' reality (Fig 16). An amazing trait of holograms reveals more 'paranormal mysteries'. Information of the *whole*

hologram is encoded in waveform in every part of the interference pattern by the way they are created. This means that every *part* of a hologram is a smaller version of the whole. Cut the interference wave-pattern into four and you won't get four parts of the image. You get quarter-sized versions of the *whole* image. The body is a hologram and the same applies. Here we have the basis of acupuncture, reflexology and other forms of healing which identify representations of the whole body in all of the parts, hands, feet, ears, everywhere. Skilled palm readers can do what they do because the information of whole body is encoded in the hand. The concept of as above, so below, comes from this.

Figure 15: A waveform interference pattern that holds the information that transforms into a hologram.

Figure 16: Holographic people including 'Elvis' holographically inserted to sing a duet with Celine Dion.

The question will be asked of why, if solidity is illusory, we can't just walk through walls and each other. The resistance is not solid against solid; it is electromagnetic field against electromagnetic field and we decode this into the *experience* of solid against solid. We should also not underestimate the power of belief to dictate reality. What you believe is impossible *will be*. Your belief impacts on your decoding processes and they won't decode what you think is impossible. What we believe we perceive and what we perceive we experience. 'Can't dos' and 'impossibles' are like a firewall in a computer system that won't put on the screen what the firewall blocks. How vital that is to understanding how human experience has been hijacked. I explain in *The Answer*, *Everything You Need To Know But Have Never Been Told* and other books a long list of 'mysteries' and 'paranormal' phenomena that are not mysterious and perfectly normal once you realise what reality is and how it works. 'Ghosts' can be seen to pass through 'solid' walls because the walls are not solid and the ghost is a discarnate entity operating on a frequency so different to that of the wall that it's like two radio stations sharing the same space while never interfering with each other. I have seen ghosts do this myself. The apartness of people and objects is also an illusion. Everything is connected by the Field like all sea life is connected by the sea. It's just that within the limits of our visual reality we only 'see' holographic information and not the field of information that connects everything and from which the holographic world is made manifest. If you can only see holographic 'objects' and not the field that connects them they will appear to you as unconnected to each other in the same way that we see the computer while not seeing the Wi-Fi.

What you don't know *can* hurt you

Okay, we return to those 'two worlds' of human society and the Cult with its global network of interconnecting secret societies and satanic groups which manipulate through governments, corporations, media, religions, etc. The fundamental difference between them is *knowledge*. The idea has been to keep humanity

ignorant of the plan for its total enslavement underpinned by a crucial ignorance of reality – who we are and where we are – and how we interact with it. 'Human' should be the interaction between our expanded eternal consciousness and the five-sense body experience. We are meant to be *in* this world in terms of the five senses but not *of* this world in relation to our greater consciousness and perspective. In that state we experience the small picture of the five senses within the wider context of the big picture of awareness beyond the five senses. Put another way the five senses see the dots and expanded awareness connects them into pictures and patterns that give context to the apparently random and unconnected. Without the context of expanded awareness the five senses see only apartness and randomness with apparently no meaning. The Cult and its other-dimensional controllers seek to intervene in the frequency realm where five-sense reality is supposed to connect with expanded reality and to keep the two apart (more on this in the final chapter). When that happens five-sense mental and emotional processes are no longer influenced by expanded awareness, or the True 'I', and instead are driven by the isolated perceptions of the body's decoding systems. They are in the world *and* of it. Here we have the human plight and why humanity with its potential for infinite awareness can be so easily manipulatable and descend into such extremes of stupidity.

Once the Cult isolates five-sense mind from expanded awareness it can then program the mind with perceptions and beliefs by controlling information that the mind receives through the 'education' system of the formative years and the media perceptual bombardment and censorship of an entire lifetime. Limit perception and a sense of the possible through limiting knowledge by limiting and skewing information while censoring and discrediting that which could set people free. As the title of another of my books says *… And The Truth Shall Set You Free*. For this reason the last thing the Cult wants in circulation is the truth about anything – especially the reality of the eternal 'I' – and that's why it is desperate to control information. The Cult knows that information becomes perception

which becomes behaviour which, collectively, becomes human society. Cult-controlled and funded mainstream 'science' denies the existence of an eternal 'I' and seeks to dismiss and trash all evidence to the contrary. Cult-controlled mainstream religion has a version of 'God' that is little more than a system of control and dictatorship that employs threats of damnation in an afterlife to control perceptions and behaviour in the here and now through fear and guilt. Neither is true and it's the 'neither' that the Cult wishes to suppress. This 'neither' is that everything is an expression, a point of attention, within an infinite state of consciousness which is the real meaning of the term 'God'.

Perceptual obsession with the 'physical body' and five-senses means that 'God' becomes personified as a bearded bloke sitting among the clouds or a raging bully who loves us if we do what 'he' wants and condemns us to the fires of hell if we don't. These are no more than a 'spiritual' fairy tales to control and dictate events and behaviour through fear of this 'God' which has bizarrely made 'God-fearing' in religious circles a state to be desired. I would suggest that fearing *anything* is not to be encouraged and celebrated, but rather deleted. You can see why 'God fearing' is so beneficial to the Cult and its religions when *they* decide what 'God' wants and what 'God' demands (the Cult demands) that everyone do. As the great American comedian Bill Hicks said satirising a Christian zealot: 'I think what God meant to say.' How much of this infinite awareness ('God') that we access is decided by how far we choose to expand our perceptions, self-identity and sense of the possible. The scale of self-identity reflects itself in the scale of awareness that we can connect with and are influenced by – how much knowing and insight we have instead of programmed perception. You cannot expand your awareness into the infinity of possibility when you believe that you are little me Peter the postman or Mary in marketing and nothing more. I'll deal with this in the concluding chapter because it's crucial to how we turnaround current events.

Where the Cult came from

When I realised in the early 1990s there was a Cult network behind global events I asked the obvious question: When did it start? I took it back to ancient Rome and Egypt and on to Babylon and Sumer in Mesopotamia, the 'Land Between Two Rivers', in what we now call Iraq. The two rivers are the Tigris and Euphrates and this region is of immense historical and other importance to the Cult, as is the land called Israel only 550 miles away by air. There is much more going with deep esoteric meaning across this whole region. It's not only about 'wars for oil'. Priceless artefacts from Mesopotamia were stolen or destroyed after the American and British invasion of Iraq in 2003 justified by the lies of Boy Bush and Tony Blair (their Cult masters) about non-existent 'weapons of mass destruction'. Mesopotamia was the location of Sumer (about 5,400BC to 1,750BC), and Babylon (about 2,350BC to 539BC). Sabbatians may have become immensely influential in the Cult in modern times but they are part of a network that goes back into the mists of history. Sumer is said by historians to be the 'cradle of civilisation'. I disagree. I say it was the re-start of what we call human civilisation after cataclysmic events symbolised in part as the 'Great Flood' destroyed the world that existed before. These fantastic upheavals that I have been describing in detail in the books since the early1990s appear in accounts and legends of ancient cultures across the world and they are supported by geological and biological evidence. Stone tablets found in Iraq detailing the Sumer period say the cataclysms were caused by non-human 'gods' they call the Anunnaki. These are described in terms of extraterrestrial visitations in which knowledge supplied by the Anunnaki is said to have been the source of at least one of the world's oldest writing systems and developments in astronomy, mathematics and architecture that were way ahead of their time. I have covered this subject at length in *The Biggest Secret* and *Children of the Matrix* and the same basic 'Anunnaki' story can be found in Zulu accounts in South Africa where the late and very great Zulu high shaman Credo Mutwa told me that the Sumerian Anunnaki were known by Zulus as the Chitauri or 'children of the serpent'. See my six-hour video interview with Credo on this subject entitled *The*

Reptilian Agenda recorded at his then home near Johannesburg in 1999 which you can watch on the Ickonic media platform.

The Cult emerged out of Sumer, Babylon and Egypt (and elsewhere) and established the Roman Empire before expanding with the Romans into northern Europe from where many empires were savagely imposed in the form of Cult-controlled societies all over the world. Mass death and destruction was their calling card. The Cult established its centre of operations in Europe and European Empires were Cult empires which allowed it to expand into a global force. Spanish and Portuguese colonialists headed for Central and South America while the British and French targeted North America. Africa was colonised by Britain, France, Belgium, the Netherlands, Portugal, Spain, Italy, and Germany. Some like Britain and France moved in on the Middle East. The British Empire was by far the biggest for a simple reason. By now Britain was the headquarters of the Cult from which it expanded to form Canada, the United States, Australia and New Zealand. The Sun never set on the British Empire such was the scale of its occupation. London remains a global centre for the Cult along with Rome and the Vatican although others have emerged in Israel and China. It is no accident that the 'virus' is alleged to have come out of China while Italy was chosen as the means to terrify the Western population into compliance with 'Covid' fascism. Nor that Israel has led the world in 'Covid' fascism and mass 'vaccination'.

You would think that I would mention the United States here, but while it has been an important means of imposing the Cult's will it is less significant than would appear and is currently in the process of having what power it does have deleted. The Cult in Europe has mostly loaded the guns for the US to fire. America has been controlled from Europe from the start through Cult operatives in Britain and Europe. The American Revolution was an illusion to make it appear that America was governing itself while very different forces were pulling the strings in the form of Cult families such as the Rothschilds through the Rockefellers and other subordinates. The Rockefellers are extremely close to Bill Gates and

established both scalpel and drug 'medicine' and the World Health Organization. They play a major role in the development and circulation of vaccines through the Rockefeller Foundation on which Bill Gates said his Foundation is based. Why wouldn't this be the case when the Rockefellers and Gates are on the same team? Cult infiltration of human society goes way back into what we call history and has been constantly expanding and centralising power with the goal of establishing a global structure to dictate everything. Look how this has been advanced in great leaps with the 'Covid' hoax.

The non-human dimension

I researched and observed the comings and goings of Cult operatives through the centuries and even thousands of years as they were born, worked to promote the agenda within the secret society and satanic networks, and then died for others to replace them. Clearly there had to be a coordinating force that spanned this entire period while operatives who would not have seen the end goal in their lifetimes came and went advancing the plan over millennia. I went in search of that coordinating force with the usual support from the extraordinary synchronicity of my life which has been an almost daily experience since 1990. I saw common themes in religious texts and ancient cultures about a non-human force manipulating human society from the hidden. Christianity calls this force Satan, the Devil and demons; Islam refers to the Jinn or Djinn; Zulus have their Chitauri (spelt in other ways in different parts of Africa); and the Gnostic people in Egypt in the period around and before 400AD referred to this phenomena as the 'Archons', a word meaning rulers in Greek. Central American cultures speak of the 'Predators' among other names and the same theme is everywhere. I will use 'Archons' as a collective name for all of them. When you see how their nature and behaviour is described all these different sources are clearly talking about the same force. Gnostics described the Archons in terms of 'luminous fire' while Islam relates the Jinn to 'smokeless fire'. Some refer to beings in form that could occasionally be seen, but the most common of common theme is that they operate from

unseen realms which means almost all existence to the visual processes of humans. I had concluded that this was indeed the foundation of human control and that the Cult was operating within the human frequency band on behalf of this hidden force when I came across the writings of Gnostics which supported my conclusions in the most extraordinary way.

A sealed earthen jar was found in 1945 near the town of Nag Hammadi about 75-80 miles north of Luxor on the banks of the River Nile in Egypt. Inside was a treasure trove of manuscripts and texts left by the Gnostic people some 1,600 years earlier. They included 13 leather-bound papyrus codices (manuscripts) and more than 50 texts written in Coptic Egyptian estimated to have been hidden in the jar in the period of 400AD although the source of the information goes back much further. Gnostics oversaw the Great or Royal Library of Alexandria, the fantastic depository of ancient texts detailing advanced knowledge and accounts of human history. The Library was dismantled and destroyed in stages over a long period with the death-blow delivered by the Cult-established Roman Church in the period around 415AD. The Church of Rome was the Church of Babylon relocated as I said earlier. Gnostics were not a race. They were a way of perceiving reality. Whenever they established themselves and their information circulated the terrorists of the Church of Rome would target them for destruction. This happened with the Great Library and with the Gnostic Cathars who were burned to death by the psychopaths after a long period of oppression at the siege of the Castle of Monségur in southern France in 1244. The Church has always been terrified of Gnostic information which demolishes the official Christian narrative although there is much in the Bible that supports the Gnostic view if you read it in another way. To anyone studying the texts of what became known as the Nag Hammadi Library it is clear that great swathes of Christian and Biblical belief has its origin with Gnostics sources going back to Sumer. Gnostic themes have been twisted to manipulate the perceived reality of Bible believers. Biblical texts have been in the open for centuries where they could be changed while Gnostic

documents found at Nag Hammadi were sealed away and untouched for 1,600 years. What you see is what they wrote.

Use your *pneuma* not your *nous*

Gnosticism and Gnostic come from 'gnosis' which means knowledge, or rather *secret* knowledge, in the sense of spiritual awareness – knowledge about reality and life itself. The desperation of the Cult's Church of Rome to destroy the Gnostics can be understood when the knowledge they were circulating was the last thing the Cult wanted the population to know. Sixteen hundred years later the same Cult is working hard to undermine and silence me for the same reason. The dynamic between knowledge and ignorance is a constant. 'Time' appears to move on, but essential themes remain the same. We are told to 'use your nous', a Gnostic word for head/brain/intelligence. They said, however, that spiritual awakening or 'salvation' could only be secured by expanding awareness *beyond* what they called *nous* and into *pneuma* or Infinite Self. Obviously as I read these texts the parallels with what I have been saying since 1990 were fascinating to me. There is a universal truth that spans human history and in that case why wouldn't we be talking the same language 16 centuries apart? When you free yourself from the perception program of the five senses and explore expanded realms of consciousness you are going to connect with the same information no matter what the perceived 'era' within a manufactured timeline of a single and tiny range of manipulated frequency. Humans working with 'smart' technology or knocking rocks together in caves is only a timeline appearing to operate within the human frequency band. Expanded awareness and the knowledge it holds have always been there whether the era be Stone Age or computer age. We can only access that knowledge by opening ourselves to its frequency which the five-sense prison cell is designed to stop us doing. Gates, Fauci, Whitty, Vallance, Zuckerberg, Brin, Page, Wojcicki, Bezos, and all the others behind the 'Covid' hoax clearly have a long wait before their range of frequency can make that connection given that an open heart is

crucial to that as we shall see. Instead of accessing knowledge directly through expanded awareness it is given to Cult operatives by the secret society networks of the Cult where it has been passed on over thousands of years outside the public arena. Expanded realms of consciousness is where great artists, composers and writers find their inspiration and where truth awaits anyone open enough to connect with it. We need to go there fast.

Archon hijack

A fifth of the Nag Hammadi texts describe the existence and manipulation of the Archons led by a 'Chief Archon' they call 'Yaldabaoth', or the 'Demiurge', and this is the Christian 'Devil', 'Satan', 'Lucifer', and his demons. Archons in Biblical symbolism are the 'fallen ones' which are also referred to as fallen angels after the angels expelled from heaven according to the Abrahamic religions of Judaism, Christianity and Islam. These angels are claimed to tempt humans to 'sin' ongoing and you will see how accurate that symbolism is during the rest of the book. The theme of 'original sin' is related to the 'Fall' when Adam and Eve were 'tempted by the serpent' and fell from a state of innocence and 'obedience' (connection) with God into a state of disobedience (disconnection). The Fall is said to have brought sin into the world and corrupted everything including human nature. Yaldabaoth, the 'Lord Archon', is described by Gnostics as a 'counterfeit spirit', 'The Blind One', 'The Blind God', and 'The Foolish One'. The Jewish name for Yaldabaoth in Talmudic writings is Samael which translates as 'Poison of God', or 'Blindness of God'. You see the parallels. Yaldabaoth in Islamic belief is the Muslim Jinn devil known as Shaytan – Shaytan is Satan as the same themes are found all over the world in every religion and culture. The 'Lord God' of the Old Testament is the 'Lord Archon' of Gnostic manuscripts and that's why he's such a bloodthirsty bastard. Satan is known by Christians as 'the Demon of Demons' and Gnostics called Yaldabaoth the 'Archon of Archons'. Both are known as 'The Deceiver'. We are talking about the same 'bloke' for sure and these common themes

using different names, storylines and symbolism tell a common tale of the human plight.

Archons are referred to in Nag Hammadi documents as mind parasites, inverters, guards, gatekeepers, detainers, judges, pitiless ones and deceivers. The 'Covid' hoax alone is a glaring example of all these things. The Biblical 'God' is so different in the Old and New Testaments because they are not describing the same phenomenon. The vindictive, angry, hate-filled, 'God' of the Old Testament, known as Yahweh, is Yaldabaoth who is depicted in Cult-dictated popular culture as the 'Dark Lord', 'Lord of Time', Lord (Darth) Vader and Dormammu, the evil ruler of the 'Dark Dimension' trying to take over the 'Earth Dimension' in the Marvel comic movie, *Dr Strange*. Yaldabaoth is both the Old Testament 'god' and the Biblical 'Satan'. Gnostics referred to Yaldabaoth as the 'Great Architect of the Universe'and the Cult-controlled Freemason network calls their god 'the 'Great Architect of the Universe' (also Grand Architect). The 'Great Architect' Yaldabaoth is symbolised by the Cult as the all-seeing eye at the top of the pyramid on the Great Seal of the United States and the dollar bill. Archon is encoded in *arch*-itect as it is in *arch*-angels and *arch*-bishops. All religions have the theme of a force for good and force for evil in some sort of spiritual war and there is a reason for that – the theme is true. The Cult and its non-human masters are quite happy for this to circulate. They present themselves as the force for good fighting evil when they are really the force of evil (absence of love). The whole foundation of Cult modus operandi is inversion. They promote themselves as a force for good and anyone challenging them in pursuit of peace, love, fairness, truth and justice is condemned as a satanic force for evil. This has been the game plan throughout history whether the Church of Rome inquisitions of non-believers or 'conspiracy theorists' and 'anti-vaxxers' of today. The technique is the same whatever the timeline era.

Yaldabaoth is revolting (true)

Yaldabaoth and the Archons are said to have revolted against God with Yaldabaoth claiming to *be* God – the *All That Is*. The Old Testament 'God' (Yaldabaoth) demanded to be worshipped as such: '*I am* the LORD, and there is none else, there is no God beside me' (Isaiah 45:5). I have quoted in other books a man who said he was the unofficial son of the late Baron Philippe de Rothschild of the Mouton-Rothschild wine producing estates in France who died in 1988 and he told me about the Rothschild 'revolt from God'. The man said he was given the name Phillip Eugene de Rothschild and we shared long correspondence many years ago while he was living under another identity. He said that he was conceived through 'occult incest' which (within the Cult) was 'normal and to be admired'. 'Phillip' told me about his experience attending satanic rituals with rich and famous people whom he names and you can see them and the wider background to Cult Satanism in my other books starting with *The Biggest Secret*. Cult rituals are interactions with Archontic 'gods'. 'Phillip' described Baron Philippe de Rothschild as 'a master Satanist and hater of God' and he used the same term 'revolt from God' associated with Yaldabaoth/Satan/Lucifer/the Devil in describing the Sabbatian Rothschild dynasty. 'I played a key role in my family's revolt from God', he said. That role was to infiltrate in classic Sabbatian style the Christian Church, but eventually he escaped the mind-prison to live another life. The Cult has been targeting religion in a plan to make worship of the Archons the global one-world religion. Infiltration of Satanism into modern 'culture', especially among the young, through music videos, stage shows and other means, is all part of this.

Nag Hammadi texts describe Yaldabaoth and the Archons in their prime form as energy – consciousness – and say they can take form if they choose in the same way that consciousness takes form as a human. Yaldabaoth is called 'formless' and represents a deeply inverted, distorted and chaotic state of consciousness which seeks to attached to humans and turn them into a likeness of itself in an attempt at assimilation. For that to happen it has to manipulate

humans into low frequency mental and emotional states that match its own. Archons can certainly appear in human form and this is the origin of the psychopathic personality. The energetic distortion Gnostics called Yaldabaoth *is* psychopathy. When psychopathic Archons take human form that human will be a psychopath as an expression of Yaldabaoth consciousness. Cult psychopaths are Archons in human form. The principle is the same as that portrayed in the 2009 *Avatar* movie when the American military travelled to a fictional Earth-like moon called Pandora in the Alpha Centauri star system to infiltrate a society of blue people, or Na'vi, by hiding within bodies that looked like the Na'vi. Archons posing as humans have a particular hybrid information field, part human, part Archon, (the ancient 'demigods') which processes information in a way that manifests behaviour to match their psychopathic evil, lack of empathy and compassion, and stops them being influenced by the empathy, compassion and love that a fully-human information field is capable of expressing. Cult bloodlines interbreed, be they royalty or dark suits, for this reason and you have their obsession with incest. Interbreeding with full-blown humans would dilute the Archontic energy field that guarantees psychopathy in its representatives in the human realm.

Gnostic writings say the main non-human forms that Archons take are *serpentine* (what I have called for decades 'reptilian' amid unbounded ridicule from the Archontically-programmed) and what Gnostics describe as 'an unborn baby or foetus with grey skin and dark, unmoving eyes'. This is an excellent representation of the ET 'Greys' of UFO folklore which large numbers of people claim to have seen and been abducted by – Zulu shaman Credo Mutwa among them. I agree with those that believe in extraterrestrial or interdimensional visitations today and for thousands of years past. No wonder with their advanced knowledge and technological capability they were perceived and worshipped as gods for technological and other 'miracles' they appeared to perform. Imagine someone arriving in a culture disconnected from the modern world with a smartphone and computer. They would be

seen as a 'god' capable of 'miracles'. The Renegade Mind, however, wants to know the source of everything and not only the way that source manifests as human or non-human. In the same way that a Renegade Mind seeks the original source material for the 'Covid virus' to see if what is claimed is true. The original source of Archons in form is consciousness – the distorted state of consciousness known to Gnostics as Yaldabaoth.

'Revolt from God' is energetic disconnection

Where I am going next will make a lot of sense of religious texts and ancient legends relating to 'Satan', Lucifer' and the 'gods'. Gnostic descriptions sync perfectly with the themes of my own research over the years in how they describe a consciousness distortion seeking to impose itself on human consciousness. I've referred to the core of infinite awareness in previous books as Infinite Awareness in Awareness of Itself. By that I mean a level of awareness that knows that it is all awareness and is aware of all awareness. From here comes the frequency of love in its true sense and balance which is what love is on one level – the balance of all forces into a single whole called Oneness and Isness. The more we disconnect from this state of love that many call 'God' the constituent parts of that Oneness start to unravel and express themselves as a part and not a whole. They become individualised as intellect, mind, selfishness, hatred, envy, desire for power over others, and such like. This is not a problem in the greater scheme in that 'God', the *All That Is*, can experience all these possibilities through different expressions of itself including humans. What we as expressions of the whole experience the *All That Is* experiences. We are the *All That Is* experiencing itself. As we withdraw from that state of Oneness we disconnect from its influence and things can get very unpleasant and very stupid. Archontic consciousness is at the extreme end of that. It has so disconnected from the influence of Oneness that it has become an inversion of unity and love, an inversion of everything, an inversion of life itself. Evil is appropriately live written backwards. Archontic consciousness is obsessed with death, an inversion of life,

and so its manifestations in Satanism are obsessed with death. They use inverted symbols in their rituals such as the inverted pentagram and cross. Sabbatians as Archontic consciousness incarnate invert Judaism and every other religion and culture they infiltrate. They seek disunity and chaos and they fear unity and harmony as they fear love like garlic to a vampire. As a result the Cult, Archons incarnate, act with such evil, psychopathy and lack of empathy and compassion disconnected as they are from the source of love. How could Bill Gates and the rest of the Archontic psychopaths do what they have to human society in the 'Covid' era with all the death, suffering and destruction involved and have no emotional consequence for the impact on others? Now you know. Why have Zuckerberg, Brin, Page, Wojcicki and company callously censored information warning about the dangers of the 'vaccine' while thousands have been dying and having severe, sometimes life-changing reactions? Now you know. Why have Tedros, Fauci, Whitty, Vallance and their like around the world been using case and death figures they're aware are fraudulent to justify lockdowns and all the deaths and destroyed lives that have come from that? Now you know. Why did Christian Drosten produce and promote a 'testing' protocol that he knew couldn't test for infectious disease which led to a global human catastrophe. Now you know. The Archontic mind doesn't give a shit (Fig 17). I personally think that Gates and major Cult insiders are a form of AI cyborg that the Archons want humans to become.

Figure 17: Artist Neil Hague's version of the 'Covid' hierarchy.

Human batteries

A state of such inversion does have its consequences, however. The level of disconnection from the Source of All means that you withdraw from that source of energetic sustenance and creativity. This means that you have to find your own supply of energetic power and it has – *us*. When the Morpheus character in the first *Matrix* movie held up a battery he spoke a profound truth when he said: 'The Matrix is a computer-generated dream world built to keep us under control in order to change the human being into one of

these.' The statement was true in all respects. We do live in a technologically-generated virtual reality simulation (more very shortly) and we have been manipulated to be an energy source for Archontic consciousness. The Disney-Pixar animated movie *Monsters, Inc*. in 2001 symbolised the dynamic when monsters in their world had no energy source and they would enter the human world to terrify children in their beds, catch the child's scream, terror (low-vibrational frequencies), and take that energy back to power the monster world. The lead character you might remember was a single giant eye and the symbolism of the Cult's all-seeing eye was obvious. Every thought and emotion is broadcast as a frequency unique to that thought and emotion. Feelings of love and joy, empathy and compassion, are high, quick, frequencies while fear, depression, anxiety, suffering and hate are low, slow, dense frequencies. Which kind do you think Archontic consciousness can connect with and absorb? In such a low and dense frequency state there's no way it can connect with the energy of love and joy. Archons can only feed off energy compatible with their own frequency and they and their Cult agents want to delete the human world of love and joy and manipulate the transmission of low vibrational frequencies through low-vibrational human mental and emotional states. *We are their energy source*. Wars are energetic banquets to the Archons – a world war even more so – and think how much low-frequency mental and emotional energy has been generated from the consequences for humanity of the 'Covid' hoax orchestrated by Archons incarnate like Gates.

The ancient practice of human sacrifice 'to the gods', continued in secret today by the Cult, is based on the same principle. 'The gods' are Archontic consciousness in different forms and the sacrifice is induced into a state of intense terror to generate the energy the Archontic frequency can absorb. Incarnate Archons in the ritual drink the blood which contains an adrenaline they crave which floods into the bloodstream when people are terrorised. Most of the sacrifices, ancient and modern, are children and the theme of 'sacrificing young virgins to the gods' is just code for children. They

have a particular pre-puberty energy that Archons want more than anything and the energy of the young in general is their target. The California Department of Education wants students to chant the names of Aztec gods (Archontic gods) once worshipped in human sacrifice rituals in a curriculum designed to encourage them to 'challenge racist, bigoted, discriminatory, imperialist/colonial beliefs', join 'social movements that struggle for social justice', and 'build new possibilities for a post-racist, post-systemic racism society'. It's the usual Woke crap that inverts racism and calls it anti-racism. In this case solidarity with 'indigenous tribes' is being used as an excuse to chant the names of 'gods' to which people were sacrificed (and still are in secret). What an example of Woke's inability to see beyond black and white, us and them, They condemn the colonisation of these tribal cultures by Europeans (quite right), but those cultures sacrificing people including children to their 'gods', and mass murdering untold numbers as the Aztecs did, is just fine. One chant is to the Aztec god Tezcatlipoca who had a man sacrificed to him in the 5th month of the Aztec calendar. His heart was cut out and he was eaten. Oh, that's okay then. Come on children … after three … Other sacrificial 'gods' for the young to chant their allegiance include Quetzalcoatl, Huitzilopochtli and Xipe Totec. The curriculum says that 'chants, affirmations, and energizers can be used to bring the class together, build unity around ethnic studies principles and values, and to reinvigorate the class following a lesson that may be emotionally taxing or even when student engagement may appear to be low'. Well, that's the cover story, anyway. Chanting and mantras are the repetition of a particular frequency generated from the vocal cords and chanting the names of these Archontic 'gods' tunes you into their frequency. That is the last thing you want when it allows for energetic synchronisation, attachment and perceptual influence. Initiates chant the names of their 'Gods' in their rituals for this very reason.

Vampires of the Woke

Paedophilia is another way that Archons absorb the energy of children. Paedophiles possessed by Archontic consciousness are used as the conduit during sexual abuse for discarnate Archons to vampire the energy of the young they desire so much. Stupendous numbers of children disappear every year never to be seen again although you would never know from the media. Imagine how much low-vibrational energy has been generated by children during the 'Covid' hoax when so many have become depressed and psychologically destroyed to the point of killing themselves. Shocking numbers of children are now taken by the state from loving parents to be handed to others. I can tell you from long experience of researching this since 1996 that many end up with paedophiles and assets of the Cult through corrupt and Cult-owned social services which in the reframing era has hired many psychopaths and emotionless automatons to do the job. Children are even stolen to order using spurious reasons to take them by the corrupt and secret (because they're corrupt) 'family courts'. I have written in detail in other books, starting with *The Biggest Secret* in 1997, about the ubiquitous connections between the political, corporate, government, intelligence and military elites (Cult operatives) and Satanism and paedophilia. If you go deep enough both networks have an interlocking leadership. The Woke mentality has been developed by the Cult for many reasons: To promote almost every aspect of its agenda; to hijack the traditional political left and turn it fascist; to divide and rule; and to target agenda pushbackers. But there are other reasons which relate to what I am describing here. How many happy and joyful Wokers do you ever see especially at the extreme end? They are a mental and psychological mess consumed by emotional stress and constantly emotionally cocked for the next explosion of indignation at someone referring to a female as a female. They are walking, talking, batteries as Morpheus might say emitting frequencies which both enslave them in low-vibrational bubbles of perceptual limitation and feed the Archons. Add to this the hatred claimed to be love; fascism claimed to 'anti-fascism', racism claimed to be 'anti-racism';

exclusion claimed to inclusion; and the abuse-filled Internet trolling. You have a purpose-built Archontic energy system with not a wind turbine in sight and all founded on Archontic *inversion*. We have whole generations now manipulated to serve the Archons with their actions and energy. They will be doing so their entire adult lives unless they snap out of their Archon-induced trance. Is it really a surprise that Cult billionaires and corporations put so much money their way? Where is the energy of joy and laughter, including laughing at yourself which is confirmation of your own emotional security? Mark Twain said: 'The human race has one really effective weapon, and that is laughter.' We must use it all the time. Woke has destroyed comedy because it has no humour, no joy, sense of irony, or self-deprecation. Its energy is dense and intense. *Mmmmm*, lunch says the Archontic frequency. Rudolf Steiner (1861-1925) was the Austrian philosopher and famous esoteric thinker who established Waldorf education or Steiner schools to treat children like unique expressions of consciousness and not minds to be programmed with the perceptions determined by authority. I'd been writing about this energy vampiring for decades when I was sent in 2016 a quote by Steiner. He was spot on:

> There are beings in the spiritual realms for whom anxiety and fear emanating from human beings offer welcome food. When humans have no anxiety and fear, then these creatures starve. If fear and anxiety radiates from people and they break out in panic, then these creatures find welcome nutrition and they become more and more powerful. These beings are hostile towards humanity. Everything that feeds on negative feelings, on anxiety, fear and superstition, despair or doubt, are in reality hostile forces in super-sensible worlds, launching cruel attacks on human beings, while they are being fed ... These are exactly the feelings that belong to contemporary culture and materialism; because it estranges people from the spiritual world, it is especially suited to evoke hopelessness and fear of the unknown in people, thereby calling up the above mentioned hostile forces against them.

Pause for a moment from this perspective and reflect on what has happened in the world since the start of 2020. Not only will pennies drop, but billion dollar bills. We see the same theme from Don Juan Matus, a Yaqui Indian shaman in Mexico and the information source for Peruvian-born writer, Carlos Castaneda, who wrote a series of

books from the 1960s to 1990s. Don Juan described the force manipulating human society and his name for the Archons was the predator:

> We have a predator that came from the depths of the cosmos and took over the rule of our lives. Human beings are its prisoners. The predator is our lord and master. It has rendered us docile, helpless. If we want to protest, it suppresses our protest. If we want to act independently, it demands that we don't do so ... indeed we are held prisoner!
>
> They took us over because we are food to them, and they squeeze us mercilessly because we are their sustenance. Just as we rear chickens in coops, the predators rear us in human coops, humaneros. Therefore, their food is always available to them.
>
> Different cultures, different eras, same recurring theme.

The 'ennoia' dilemma

Nag Hammadi Gnostic manuscripts say that Archon consciousness has no 'ennoia'. This is directly translated as 'intentionality', but I'll use the term 'creative imagination'. The *All That Is* in awareness of itself is the source of all creativity – all possibility – and the more disconnected you are from that source the more you are subsequently denied 'creative imagination'. Given that Archon consciousness is almost entirely disconnected it severely lacks creativity and has to rely on far more mechanical processes of thought and exploit the creative potential of those that do have 'ennoia'. You can see cases of this throughout human society. Archon consciousness almost entirely dominates the global banking system and if we study how that system works you will appreciate what I mean. Banks manifest 'money' out of nothing by issuing lines of 'credit' which is 'money' that has never, does not, and will never exist except in theory. It's a confidence trick. If you think 'credit' figures-on-a-screen 'money' is worth anything you accept it as payment. If you don't then the whole system collapses through lack of confidence in the value of that 'money'. Archontic bankers with no 'ennoia' are 'lending' 'money' that doesn't exist to humans that *do* have creativity – those that have the inspired ideas and create businesses and products. Archon banking feeds off human creativity

which it controls through 'money' creation and debt. Humans have the creativity and Archons exploit that for their own benefit and control while having none themselves. Archon Internet platforms like Facebook claim joint copyright of everything that creative users post and while Archontic minds like Zuckerberg may officially head that company it will be human creatives on the staff that provide the creative inspiration. When you have limitless 'money' you can then buy other companies established by creative humans. Witness the acquisition record of Facebook, Google and their like. Survey the Archon-controlled music industry and you see non-creative dark suit executives making their fortune from the human creativity of their artists. The cases are endless. Research the history of people like Gates and Zuckerberg and how their empires were built on exploiting the creativity of others. Archon minds cannot create out of nothing, but they are skilled (because they have to be) in what Gnostic texts call 'countermimicry'. They can imitate, but not innovate. Sabbatians trawl the creativity of others through backdoors they install in computer systems through their cybersecurity systems. Archon-controlled China is globally infamous for stealing intellectual property and I remember how Hong Kong, now part of China, became notorious for making counterfeit copies of the creativity of others – 'countermimicry'. With the now pervasive and all-seeing surveillance systems able to infiltrate any computer you can appreciate the potential for Archons to vampire the creativity of humans. Author John Lamb Lash wrote in his book about the Nag Hammadi texts, *Not In His Image*:

> Although they cannot originate anything, because they lack the divine factor of ennoia (intentionality), Archons can imitate with a vengeance. Their expertise is simulation (HAL, virtual reality). The Demiurge [Yaldabaoth] fashions a heaven world copied from the fractal patterns [of the original] ... His construction is celestial kitsch, like the fake Italianate villa of a Mafia don complete with militant angels to guard every portal.

This brings us to something that I have been speaking about since the turn of the millennium. Our reality is a simulation; a virtual reality that we think is real. No, I'm not kidding.

Human reality? Well, virtually

I had pondered for years about whether our reality is 'real' or some kind of construct. I remembered being immensely affected on a visit as a small child in the late 1950s to the then newly-opened Planetarium on the Marylebone Road in London which is now closed and part of the adjacent Madame Tussauds wax museum. It was in the middle of the day, but when the lights went out there was the night sky projected in the Planetarium's domed ceiling and it appeared to be so real. The experience never left me and I didn't know why until around the turn of the millennium when I became certain that our 'night sky' and entire reality is a projection, a virtual reality, akin to the illusory world portrayed in the *Matrix* movies. I looked at the sky one day in this period and it appeared to me like the domed roof of the Planetarium. The release of the first *Matrix* movie in 1999 also provided a synchronistic and perfect visual representation of where my mind had been going for a long time. I hadn't come across the Gnostic Nag Hammadi texts then. When I did years later the correlation was once again astounding. As I read Gnostic accounts from 1,600 years and more earlier it was clear that they were describing the same simulation phenomenon. They tell how the Yaldabaoth 'Demiurge' and Archons created a 'bad copy' of original reality to rule over all that were captured by its illusions and the body was a prison to trap consciousness in the 'bad copy' fake reality. Read how Gnostics describe the 'bad copy' and update that to current times and they are referring to what we would call today a virtual reality simulation.

Author John Lamb Lash said 'the Demiurge fashions a heaven world copied from the fractal patterns' of the original through expertise in 'HAL' or virtual reality simulation. Fractal patterns are part of the energetic information construct of our reality, a sort of blueprint. If these patterns were copied in computer terms it would indeed give you a copy of a 'natural' reality in a non-natural frequency and digital form. The principle is the same as making a copy of a website. The original website still exists, but now you can change the copy version to make it whatever you like and it can

become very different to the original website. Archons have done this with our reality, a *synthetic* copy of prime reality that still exists beyond the frequency walls of the simulation. Trapped within the illusions of this synthetic Matrix, however, were and are human consciousness and other expressions of prime reality and this is why the Archons via the Cult are seeking to make the human body synthetic and give us synthetic AI minds to complete the job of turning the entire reality synthetic including what we perceive to be the natural world. To quote Kurzweil: 'Nanobots will infuse all the matter around us with information. Rocks, trees, everything will become these intelligent creatures.' Yes, *synthetic* 'creatures' just as 'Covid' and other genetically-manipulating 'vaccines' are designed to make the human body synthetic. From this perspective it is obvious why Archons and their Cult are so desperate to infuse synthetic material into every human with their 'Covid' scam.

Let there be (electromagnetic) light

Yaldabaoth, the force that created the simulation, or Matrix, makes sense of the Gnostic reference to 'The Great Architect' and its use by Cult Freemasonry as the name of its deity. The designer of the Matrix in the movies is called 'The Architect' and that trilogy is jam-packed with symbolism relating to these subjects. I have contended for years that the angry Old Testament God (Yaldabaoth) is the 'God' being symbolically 'quoted' in the opening of Genesis as 'creating the world'. This is not the creation of prime reality – it's the creation of the *simulation*. The Genesis 'God' says: 'Let there be Light: and there was light.' But what is this 'Light'? I have said for decades that the speed of light (186,000 miles per second) is not the fastest speed possible as claimed by mainstream science and is in fact the frequency walls or outer limits of the Matrix. You can't have a fastest or slowest anything within all possibility when everything is possible. The human body is encoded to operate within the speed of light or *within the simulation* and thus we see only the tiny frequency band of visible *light*. Near-death experiencers who perceive reality outside the body during temporary 'death' describe a very different

form of light and this is supported by the Nag Hammadi texts. Prime reality beyond the simulation ('Upper Aeons' to the Gnostics) is described as a realm of incredible beauty, bliss, love and harmony – a realm of 'watery light' that is so powerful 'there are no shadows'. Our false reality of Archon control, which Gnostics call the 'Lower Aeons', is depicted as a realm with a different kind of 'light' and described in terms of chaos, 'Hell', 'the Abyss' and 'Outer Darkness', where trapped souls are tormented and manipulated by demons (relate that to the 'Covid' hoax alone). The watery light theme can be found in near-death accounts and it is not the same as *simulation* 'light' which is electromagnetic or radiation light within the speed of light – the 'Lower Aeons'. Simulation 'light' is the 'luminous fire' associated by Gnostics with the Archons. The Bible refers to Yaldabaoth as 'that old serpent, called the Devil, and Satan, which deceiveth the whole world' (Revelation 12:9). I think that making a simulated copy of prime reality ('countermimicry') and changing it dramatically while all the time manipulating humanity to believe it to be real could probably meet the criteria of deceiving the whole world. Then we come to the Cult god Lucifer – the *Light Bringer*. Lucifer is symbolic of Yaldabaoth, the bringer of radiation light that forms the bad copy simulation within the speed of light. 'He' is symbolised by the lighted torch held by the Statue of Liberty and in the name 'Illuminati'. Sabbatian-Frankism declares that Lucifer is the true god and Lucifer is the real god of Freemasonry honoured as their 'Great or Grand Architect of the Universe' (simulation).

I would emphasise, too, the way Archontic technologically-generated luminous fire of radiation has deluged our environment since I was a kid in the 1950s and changed the nature of The Field with which we constantly interact. Through that interaction technological radiation is changing us. The Smart Grid is designed to operate with immense levels of communication power with 5G expanding across the world and 6G, 7G, in the process of development. Radiation is the simulation and the Archontic manipulation system. Why wouldn't the Archon Cult wish to unleash radiation upon us to an ever-greater extreme to form

Kurzweil's 'cloud'? The plan for a synthetic human is related to the need to cope with levels of radiation beyond even anything we've seen so far. Biological humans would not survive the scale of radiation they have in their script. The Smart Grid is a technological sub-reality within the technological simulation to further disconnect five-sense perception from expanded consciousness. It's a technological prison of the mind.

Infusing the 'spirit of darkness'

A recurring theme in religion and native cultures is the manipulation of human genetics by a non-human force and most famously recorded as the biblical 'sons of god' (the god*s* plural in the original) who interbred with the daughters of men. The Nag Hammadi *Apocryphon of John* tells the same story this way:

> He [Yaldabaoth] sent his angels [Archons/demons] to the daughters of men, that they might take some of them for themselves and raise offspring for their enjoyment. And at first they did not succeed. When they had no success, they gathered together again and they made a plan together ... And the angels changed themselves in their likeness into the likeness of their mates, filling them with the spirit of darkness, which they had mixed for them, and with evil ... And they took women and begot children out of the darkness according to the likeness of their spirit.

Possession when a discarnate entity takes over a human body is an age-old theme and continues today. It's very real and I've seen it. Satanic and secret society rituals can create an energetic environment in which entities can attach to initiates and I've heard many stories of how people have changed their personality after being initiated even into lower levels of the Freemasons. I have been inside three Freemasonic temples, one at a public open day and two by just walking in when there was no one around to stop me. They were in Ryde, the town where I live, Birmingham, England, when I was with a group, and Boston, Massachusetts. They all felt the same energetically – dark, dense, low-vibrational and sinister. Demonic attachment can happen while the initiate has no idea what is going on. To them it's just a ritual to get in the Masons and do a bit of good

business. In the far more extreme rituals of Satanism human possession is even more powerful and they are designed to make possession possible. The hierarchy of the Cult is dictated by the power and perceived status of the possessing Archon. In this way the Archon hierarchy becomes the Cult hierarchy. Once the entity has attached it can influence perception and behaviour and if it attaches to the extreme then so much of its energy (information) infuses into the body information field that the hologram starts to reflect the nature of the possessing entity. This is the *Exorcist* movie type of possession when facial features change and it's known as shapeshifting. Islam's Jinn are said to be invisible tricksters who change shape, 'whisper', confuse and take human form. These are all traits of the Archons and other versions of the same phenomenon. Extreme possession could certainty infuse the 'spirit of darkness' into a partner during sex as the Nag Hammadi texts appear to describe. Such an infusion can change genetics which is also energetic information. Human genetics is information and the 'spirit of darkness' is information. Mix one with the other and change must happen. Islam has the concept of a 'Jinn baby' through possession of the mother and by Jinn taking human form. There are many ways that human genetics can be changed and remember that Archons have been aware all along of advanced techniques to do this. What is being done in human society today – and far more – was known about by Archons at the time of the 'fallen ones' and their other versions described in religions and cultures.

Archons and their human-world Cult are obsessed with genetics as we see today and they know this dictates how information is processed into perceived reality during a human life. They needed to produce a human form that would decode the simulation and this is symbolically known as 'Adam and Eve' who left the 'garden' (prime reality) and 'fell' into Matrix reality. The simulation is not a 'physical' construct (there is no 'physical'); it is a source of information. Think Wi-Fi again. The simulation is an energetic field encoded with information and body-brain systems are designed to decode that information encoded in wave or frequency form which

is transmitted to the brain as electrical signals. These are decoded by the brain to construct our sense of reality – an illusory 'physical' world that only exists in the brain or the mind. Virtual reality games mimic this process using the same sensory decoding system. Information is fed to the senses to decode a virtual reality that can appear so real, but isn't (Figs 18 and 19). Some scientists believe – and I agree with them – that what we perceive as 'physical' reality only exists when we are looking or observing. The act of perception or focus triggers the decoding systems which turn waveform information into holographic reality. When we are not observing something our reality reverts from a holographic state to a waveform state. This relates to the same principle as a falling tree not making a noise unless someone is there to hear it or decode it. The concept makes sense from the simulation perspective. A computer is not decoding all the information in a Wi-Fi field all the time and only decodes or brings into reality on the screen that part of Wi-Fi that it's decoding – focusing upon – at that moment.

Figure 18: Virtual reality technology 'hacks' into the body's five-sense decoding system.

Figure 19: The result can be experienced as very 'real'.

Interestingly, Professor Donald Hoffman at the Department of Cognitive Sciences at the University of California, Irvine, says that our experienced reality is like a computer interface that shows us only the level with which we interact while hiding all that exists beyond it: 'Evolution shaped us with a user interface that hides the truth. Nothing that we see is the truth – the very language of space and time and objects is the wrong language to describe reality.' He is correct in what he says on so many levels. Space and time are not a universal reality. They are a phenomenon of decoded *simulation* reality as part of the process of enslaving our sense of reality. Near-death experiencers report again and again how space and time did not exist as we perceive them once they were free of the body – body decoding systems. You can appreciate from this why Archons and their Cult are so desperate to entrap human attention in the five senses where we are in the Matrix and of the Matrix. Opening your mind to expanded states of awareness takes you beyond the information confines of the simulation and you become aware of knowledge and insights denied to you before. This is what we call 'awakening' – *awakening from the Matrix* – and in the final chapter I will relate this to current events.

Where are the 'aliens'?

A simulation would explain the so-called 'Fermi Paradox' named after Italian physicist Enrico Fermi (1901-1954) who created the first nuclear reactor. He considered the question of why there is such a lack of extraterrestrial activity when there are so many stars and planets in an apparently vast universe; but what if the night sky that we see, or think we do, is a simulated projection as I say? If you control the simulation and your aim is to hold humanity fast in essential ignorance would you want other forms of life including advanced life coming and going sharing information with humanity? Or would you want them to believe they were isolated and apparently alone? Themes of human isolation and apartness are common whether they be the perception of a lifeless universe or the fascist isolation laws of the 'Covid' era. Paradoxically the very

existence of a simulation means that we are not alone when some force had to construct it. My view is that experiences that people have reported all over the world for centuries with Reptilians and Grey entities are Archon phenomena as Nag Hammadi texts describe; and that benevolent 'alien' interactions are non-human groups that come in and out of the simulation by overcoming Archon attempts to keep them out. It should be highlighted, too, that Reptilians and Greys are obsessed with *genetics* and *technology* as related by cultural accounts and those who say they have been abducted by them. Technology is their way of overcoming some of the limitations in their creative potential and our technology-driven and controlled human society of today is *arch*etypical Archon-Reptilian-Grey modus operandi. Technocracy is really *Archon*tocracy. The Universe does not have to be as big as it appears with a simulation. There is no space or distance only information decoded into holographic reality. What we call 'space' is only the absence of holographic 'objects' and that 'space' is The Field of energetic information which connects everything into a single whole. The same applies with the artificially-generated information field of the simulation. The Universe is not big or small as a physical reality. It is decoded information, that's all, and its perceived size is decided by the way the simulation is encoded to make it appear. The entire night sky as we perceive it only exists in our brain and so where are those 'millions of light years'? The 'stars' on the ceiling of the Planetarium looked a vast distance away.

There's another point to mention about 'aliens'. I have been highlighting since the 1990s the plan to stage a fake 'alien invasion' to justify the centralisation of global power and a world military. Nazi scientist Werner von Braun, who was taken to America by Operation Paperclip after World War Two to help found NASA, told his American assistant Dr Carol Rosin about the Cult agenda when he knew he was dying in 1977. Rosin said that he told her about a sequence that would lead to total human control by a one-world government. This included threats from terrorism, rogue nations, meteors and asteroids before finally an 'alien invasion'. All of these

things, von Braun said, would be bogus and what I would refer to as a No-Problem-Reaction-Solution. Keep this in mind when 'the aliens are coming' is the new mantra. The aliens are not coming – they are *already here* and they have infiltrated human society while looking human. French-Canadian investigative journalist Serge Monast said in 1994 that he had uncovered a NASA/military operation called Project Blue Beam which fits with what Werner von Braun predicted. Monast died of a 'heart attack' in 1996 the day after he was arrested and spent a night in prison. He was 51. He said Blue Beam was a plan to stage an alien invasion that would include religious figures beamed holographically into the sky as part of a global manipulation to usher in a 'new age' of worshipping what I would say is the Cult 'god' Yaldabaoth in a one-world religion. Fake holographic asteroids are also said to be part of the plan which again syncs with von Braun. How could you stage an illusory threat from asteroids unless they were holographic inserts? This is pretty straightforward given the advanced technology outside the public arena and the fact that our 'physical' reality is holographic anyway. Information fields would be projected and we would decode them into the illusion of a 'physical' asteroid. If they can sell a global 'pandemic' with a 'virus' that doesn't exist what will humans not believe if government and media tell them?

All this is particularly relevant as I write with the Pentagon planning to release in June, 2021, information about 'UFO sightings'. I have been following the UFO story since the early 1990s and the common theme throughout has been government and military denials and cover up. More recently, however, the Pentagon has suddenly become more talkative and apparently open with Air Force pilot radar images released of unexplained craft moving and changing direction at speeds well beyond anything believed possible with human technology. Then, in March, 2021, former Director of National Intelligence John Ratcliffe said a Pentagon report months later in June would reveal a great deal of information about UFO sightings unknown to the public. He said the report would have 'massive implications'. The order to do this was included bizarrely

in a $2.3 trillion 'coronavirus' relief and government funding bill passed by the Trump administration at the end of 2020. I would add some serious notes of caution here. I have been pointing out since the 1990s that the US military and intelligence networks have long had craft – 'flying saucers' or anti-gravity craft – which any observer would take to be extraterrestrial in origin. Keeping this knowledge from the public allows craft flown by *humans* to be perceived as alien visitations. I am not saying that 'aliens' do not exist. I would be the last one to say that, but we have to be streetwise here. President Ronald Reagan told the UN General Assembly in 1987: 'I occasionally think how quickly our differences worldwide would vanish if we were facing an alien threat from outside this world.' That's the idea. Unite against a common 'enemy' with a common purpose behind your 'saviour force' (the Cult) as this age-old technique of mass manipulation goes global.

Science moves this way ...

I could find only one other person who was discussing the simulation hypothesis publicly when I concluded it was real. This was Nick Bostrom, a Swedish-born philosopher at the University of Oxford, who has explored for many years the possibility that human reality is a computer simulation although his version and mine are not the same. Today the simulation and holographic reality hypothesis have increasingly entered the scientific mainstream. Well, the more open-minded mainstream, that is. Here are a few of the ever-gathering examples. American nuclear physicist Silas Beane led a team of physicists at the University of Bonn in Germany pursuing the question of whether we live in a simulation. They concluded that we probably do and it was likely based on a lattice of cubes. They found that cosmic rays align with that specific pattern. The team highlighted the Greisen–Zatsepin–Kuzmin (GZK) limit which refers to cosmic ray particle interaction with cosmic background radiation that creates an apparent boundary for cosmic ray particles. They say in a paper entitled 'Constraints on the Universe as a Numerical Simulation' that this 'pattern of constraint' is exactly what you

would find with a computer simulation. They also made the point that a simulation would create its own 'laws of physics' that would limit possibility. I've been making the same point for decades that the *perceived* laws of physics relate only to this reality, or what I would later call the simulation. When designers write codes to create computer and virtual reality games they are the equivalent of the laws of physics for that game. Players interact within the limitations laid out by the coding. In the same way those who wrote the codes for the simulation decided the laws of physics that would apply. These can be overridden by expanded states of consciousness, but not by those enslaved in only five-sense awareness where simulation codes rule. Overriding the codes is what people call 'miracles'. They are not. They are bypassing the encoded limits of the simulation. A population caught in simulation perception would have no idea that this was their plight. As the Bonn paper said: 'Like a prisoner in a pitch-black cell we would not be able to see the "walls" of our prison,' That's true if people remain mesmerised by the five senses. Open to expanded awareness and those walls become very clear. The main one is the speed of light.

American theoretical physicist James Gates is another who has explored the simulation question and found considerable evidence to support the idea. Gates was Professor of Physics at the University of Maryland, Director of The Center for String and Particle Theory, and on Barack Obama's Council of Advisors on Science and Technology. He and his team found *computer codes* of digital data embedded in the fabric of our reality. They relate to on-off electrical charges of 1 and 0 in the binary system used by computers. 'We have no idea what they are doing there', Gates said. They found within the energetic fabric mathematical sequences known as error-correcting codes or block codes that 'reboot' data to its original state or 'default settings' when something knocks it out of sync. Gates was asked if he had found a set of equations embedded in our reality indistinguishable from those that drive search engines and browsers and he said: 'That is correct.' Rich Terrile, director of the Centre for Evolutionary Computation and Automated Design at NASA's Jet

Propulsion Laboratory, has said publicly that he believes the Universe is a digital hologram that must have been created by a form of intelligence. I agree with that in every way. Waveform information is delivered electrically by the senses to the brain which constructs a *digital* holographic reality that we call the 'world'. This digital level of reality can be read by the esoteric art of numerology. Digital holograms are at the cutting edge of holographics today. We have digital technology everywhere designed to access and manipulate our digital level of perceived reality. Synthetic mRNA in 'Covid vaccines' has a digital component to manipulate the body's digital 'operating system'.

Reality is numbers

How many know that our reality can be broken down to numbers and codes that are the same as computer games? Max Tegmark, a physicist at the Massachusetts Institute of Technology (MIT), is the author of *Our Mathematical Universe* in which he lays out how reality can be entirely described by numbers and maths in the way that a video game is encoded with the 'physics' of computer games. Our world and computer virtual reality are essentially the same. Tegmark imagines the perceptions of characters in an advanced computer game when the graphics are so good they don't know they are in a game. They think they can bump into real objects (electromagnetic resistance in our reality), fall in love and feel emotions like excitement. When they began to study the apparently 'physical world' of the video game they would realise that everything was made of pixels (which have been found in our energetic reality as must be the case when on one level our world is digital). What computer game characters thought was physical 'stuff', Tegmark said, could actually be broken down into numbers:

And we're exactly in this situation in our world. We look around and it doesn't seem that mathematical at all, but everything we see is made out of elementary particles like quarks and electrons. And what properties does an electron have? Does it have a smell or a colour or a texture? No! ... We physicists have come up with geeky names for [Electron] properties, like

electric charge, or spin, or lepton number, but the electron doesn't care what we call it, the properties are just numbers.

This is the illusory reality Gnostics were describing. This is the simulation. The A, C, G, and T codes of DNA have a binary value – A and C = 0 while G and T = 1. This has to be when the simulation is digital and the body must be digital to interact with it. Recurring mathematical sequences are encoded throughout reality and the body. They include the Fibonacci sequence in which the two previous numbers are added to get the next one, as in ... 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, etc. The sequence is encoded in the human face and body, proportions of animals, DNA, seed heads, pine cones, trees, shells, spiral galaxies, hurricanes and the number of petals in a flower. The list goes on and on. There are fractal patterns – a 'never-ending pattern that is infinitely complex and self-similar across all scales in the as above, so below, principle of holograms. These and other famous recurring geometrical and mathematical sequences such as Phi, Pi, Golden Mean, Golden Ratio and Golden Section are *computer codes* of the simulation. I had to laugh and give my head a shake the day I finished this book and it went into the production stage. I was sent an article in *Scientific American* published in April, 2021, with the headline 'Confirmed! We Live in a Simulation'. Two decades after I first said our reality is a simulation and the speed of light is it's outer limit the article suggested that we do live in a simulation and that the speed of light is its outer limit. I left school at 15 and never passed a major exam in my life while the writer was up to his eyes in qualifications. As I will explain in the final chapter *knowing* is far better than thinking and they come from very different sources. The article rightly connected the speed of light to the processing speed of the 'Matrix' and said what has been in my books all this time … 'If we are in a simulation, as it appears, then space is an abstract property written in code. It is not real'. No it's not and if we live in a simulation something created it and it wasn't *us*. 'That David Icke says we are manipulated by aliens' – he's crackers.'

Wow ...

The reality that humanity thinks is so real is an illusion. Politicians, governments, scientists, doctors, academics, law enforcement, media, school and university curriculums, on and on, are all founded on a world that *does not exist* except as a simulated prison cell. Is it such a stretch to accept that 'Covid' doesn't exist when our entire 'physical' reality doesn't exist? Revealed here is the knowledge kept under raps in the Cult networks of compartmentalised secrecy to control humanity's sense of reality by inducing the population to believe in a reality that's not real. If it wasn't so tragic in its experiential consequences the whole thing would be hysterically funny. None of this is new to Renegade Minds. Ancient Greek philosopher Plato (about 428 to about 347BC) was a major influence on Gnostic belief and he described the human plight thousands of years ago with his Allegory of the Cave. He told the symbolic story of prisoners living in a cave who had never been outside. They were chained and could only see one wall of the cave while behind them was a fire that they could not see. Figures walked past the fire casting shadows on the prisoners' wall and those moving shadows became their sense of reality. Some prisoners began to study the shadows and were considered experts on them (today's academics and scientists), but what they studied was only an illusion (today's academics and scientists). A prisoner escaped from the cave and saw reality as it really is. When he returned to report this revelation they didn't believe him, called him mad and threatened to kill him if he tried to set them free. Plato's tale is not only a brilliant analogy of the human plight and our illusory reality. It describes, too, the dynamics of the 'Covid' hoax. I have only skimmed the surface of these subjects here. The aim of this book is to crisply connect all essential dots to put what is happening today into its true context. All subject areas and their connections in this chapter are covered in great evidential detail in *Everything You Need To Know, But Have Never Been Told* and *The Answer*.

They say that bewildered people 'can't see the forest for the trees'. Humanity, however, can't see the forest for the *twigs*. The five senses

see only twigs while Renegade Minds can see the forest and it's the forest where the answers lie with the connections that reveals. Breaking free of perceptual programming so the forest can be seen is the way we turn all this around. Not breaking free is how humanity got into this mess. The situation may seem hopeless, but I promise you it's not. We are a perceptual heartbeat from paradise if only we knew.

CHAPTER TWELVE

Escaping Wetiko

Life is simply a vacation from the infinite
Dean Cavanagh

Renegade Minds weave the web of life and events and see common themes in the apparently random. They are always there if you look for them and their pursuit is aided by incredible synchronicity that comes when your mind is open rather than mesmerised by what it thinks it can see.

Infinite awareness is infinite possibility and the more of infinite possibility that we access the more becomes infinitely possible. That may be stating the apparently obvious, but it is a devastatingly-powerful fact that can set us free. We are a point of attention within an infinity of consciousness. The question is how much of that infinity do we choose to access? How much knowledge, insight, awareness, wisdom, do we want to connect with and explore? If your focus is only in the five senses you will be influenced by a fraction of infinite awareness. I mean a range so tiny that it gives new meaning to infinitesimal. Limitation of self-identity and a sense of the possible limit accordingly your range of consciousness. We are what we think we are. Life is what we think it is. The dream is the dreamer and the dreamer is the dream. Buddhist philosophy puts it this way: 'As a thing is viewed, so it appears.' Most humans live in the realm of touch, taste, see, hear, and smell and that's the limit of their sense of the possible and sense of self. Many will follow a religion and speak of a God in his heaven, but their lives are still

dominated by the five senses in their perceptions and actions. The five senses become the arbiter of everything. When that happens all except a smear of infinity is sealed away from influence by the rigid, unyielding, reality bubbles that are the five-sense human or Phantom Self. Archon Cult methodology is to isolate consciousness within five-sense reality – the simulation – and then program that consciousness with a sense of self and the world through a deluge of life-long information designed to instil the desired perception that allows global control. Efforts to do this have increased dramatically with identity politics as identity bubbles are squeezed into the minutiae of five-sense detail which disconnect people even more profoundly from the infinite 'I'.

Five-sense focus and self-identity are like a firewall that limits access to the infinite realms. You only perceive one radio or television station and no other. We'll take that literally for a moment. Imagine a vast array of stations giving different information and angles on reality, but you only ever listen to one. Here we have the human plight in which the population is overwhelmingly confined to CultFM. This relates only to the frequency range of CultFM and limits perception and insight to that band – limits *possibility* to that band. It means you are connecting with an almost imperceptibly minuscule range of possibility and creative potential within the infinite Field. It's a world where everything seems apart from everything else and where synchronicity is rare. Synchronicity is defined in the dictionary as 'the happening by chance of two or more related or similar events at the same time'. Use of 'by chance' betrays a complete misunderstanding of reality. Synchronicity is not 'by chance'. As people open their minds, or 'awaken' to use the term, they notice more and more coincidences in their lives, bits of 'luck', apparently miraculous happenings that put them in the right place at the right time with the right people. Days become peppered with 'fancy meeting you here' and 'what are the chances of that?' My entire life has been lived like this and ever more so since my own colossal awakening in 1990 and 91 which transformed my sense of reality. Synchronicity is not 'by chance'; it is by accessing expanded

realms of possibility which allow expanded potential for manifestation. People broadcasting the same vibe from the same openness of mind tend to be drawn 'by chance' to each other through what I call frequency magnetism and it's not only people. In the last more than 30 years incredible synchronicity has also led me through the Cult maze to information in so many forms and to crucial personal experiences. These 'coincidences' have allowed me to put the puzzle pieces together across an enormous array of subjects and situations. Those who have breached the bubble of five-sense reality will know exactly what I mean and this escape from the perceptual prison cell is open to everyone whenever they make that choice. This may appear super-human when compared with the limitations of 'human', but it's really our natural state. 'Human' as currently experienced is consciousness in an unnatural state of induced separation from the infinity of the whole. I'll come to how this transformation into unity can be made when I have described in more detail the force that holds humanity in servitude by denying this access to infinite self.

The Wetiko factor

I have been talking and writing for decades about the way five-sense mind is systematically barricaded from expanded awareness. I have used the analogy of a computer (five-sense mind) and someone at the keyboard (expanded awareness). Interaction between the computer and the operator is symbolic of the interaction between five-sense mind and expanded awareness. The computer directly experiences the Internet and the operator experiences the Internet via the computer which is how it's supposed to be – the two working as one. Archons seek to control that point where the operator connects with the computer to stop that interaction (Fig 20). Now the operator is banging the keyboard and clicking the mouse, but the computer is not responding and this happens when the computer is taken over – *possessed* – by an appropriately-named computer 'virus'. The operator has lost all influence over the computer which goes its own way making decisions under the control of the 'virus'. I have

just described the dynamic through which the force known to Gnostics as Yaldabaoth and Archons disconnects five-sense mind from expanded awareness to imprison humanity in perceptual servitude.

Figure 20: The mind 'virus' I have been writing about for decades seeks to isolate five-sense mind (the computer) from the true 'I'. (Image by Neil Hague).

About a year ago I came across a Native American concept of Wetiko which describes precisely the same phenomenon. Wetiko is the spelling used by the Cree and there are other versions including wintiko and windigo used by other tribal groups. They spell the name with lower case, but I see Wetiko as a proper noun as with Archons and prefer a capital. I first saw an article about Wetiko by writer and researcher Paul Levy which so synced with what I had been writing about the computer/operator disconnection and later the Archons. I then read his book, the fascinating *Dispelling Wetiko, Breaking the Spell of Evil*. The parallels between what I had concluded long before and the Native American concept of Wetiko were so clear and obvious that it was almost funny. For Wetiko see the Gnostic Archons for sure and the Jinn, the Predators, and every other name for a force of evil, inversion and chaos. Wetiko is the Native American name for the force that divides the computer from

the operator (Fig 21). Indigenous author Jack D. Forbes, a founder of the Native American movement in the 1960s, wrote another book about Wetiko entitled *Columbus And Other Cannibals – The Wetiko Disease of Exploitation, Imperialism, and Terrorism* which I also read. Forbes says that Wetiko refers to an evil person or spirit 'who terrorizes other creatures by means of terrible acts, including cannibalism'. Zulu shaman Credo Mutwa told me that African accounts tell how cannibalism was brought into the world by the Chitauri 'gods' – another manifestation of Wetiko. The distinction between 'evil person or spirit' relates to Archons/Wetiko possessing a human or acting as pure consciousness. Wetiko is said to be a sickness of the soul or spirit and a state of being that takes but gives nothing back – the Cult and its operatives perfectly described. Black Hawk, a Native American war leader defending their lands from confiscation, said European invaders had 'poisoned hearts' – Wetiko hearts – and that this would spread to native societies. Mention of the heart is very significant as we shall shortly see. Forbes writes: 'Tragically, the history of the world for the past 2,000 years is, in great part, the story of the epidemiology of the wetiko disease.' Yes, and much longer. Forbes is correct when he says: 'The wetikos destroyed Egypt and Babylon and Athens and Rome and Tenochtitlan [capital of the Aztec empire] and perhaps now they will destroy the entire earth.' Evil, he said, is the number one export of a Wetiko culture – see its globalisation with 'Covid'. Constant war, mass murder, suffering of all kinds, child abuse, Satanism, torture and human sacrifice are all expressions of Wetiko and the Wetiko possessed. The world is Wetiko made manifest, *but it doesn't have to be*. There is a way out of this even now.

Figure 21: The mind 'virus' is known to Native Americans as 'Wetiko'. (Image by Neil Hague).

Cult of Wetiko

Wetiko is the Yaldabaoth frequency distortion that seeks to attach to human consciousness and absorb it into its own. Once this connection is made Wetiko can drive the perceptions of the target which they believe to be coming from their own mind. All the horrors of history and today from mass killers to Satanists, paedophiles like Jeffrey Epstein and other psychopaths, are the embodiment of Wetiko and express its state of being in all its grotesqueness. The Cult is Wetiko incarnate, Yaldabaoth incarnate, and it seeks to facilitate Wetiko assimilation of humanity in totality into its distortion by manipulating the population into low frequency states that match its own. Paul Levy writes: 'Holographically enforced within the psyche of every human being the wetiko virus pervades and underlies the entire field of consciousness, and can therefore potentially manifest through any one of us at any moment if we are not mindful.' The 'Covid' hoax has achieved this with many people, but others have not fallen into Wetiko's frequency lair. Players in the 'Covid' human catastrophe including Gates, Schwab, Tedros, Fauci, Whitty, Vallance, Johnson, Hancock, Ferguson, Drosten, and all the rest, including the psychopath psychologists, are expressions of Wetiko. This is why

they have no compassion or empathy and no emotional consequence for what they do that would make them stop doing it. Observe all the people who support the psychopaths in authority against the Pushbackers despite the damaging impact the psychopaths have on their own lives and their family's lives. You are again looking at Wetiko possession which prevents them seeing through the lies to the obvious scam going on. *Why can't they see it?* Wetiko won't let them see it. The perceptual divide that has now become a chasm is between the Wetikoed and the non-Wetikoed.

Paul Levy describes Wetiko in the same way that I have long described the Archontic force. They are the same distorted consciousness operating across dimensions of reality: '… the subtle body of wetiko is not located in the third dimension of space and time, literally existing in another dimension … it is able to affect ordinary lives by mysteriously interpenetrating into our three-dimensional world.' Wetiko does this through its incarnate representatives in the Cult and by weaving itself into The Field which on our level of reality is the electromagnetic information field of the simulation or Matrix. More than that, the simulation *is* Wetiko / Yaldabaoth. Caleb Scharf, Director of Astrobiology at Columbia University, has speculated that 'alien life' could be so advanced that it has transcribed itself into the quantum realm to become what we call physics. He said intelligence indistinguishable from the fabric of the Universe would solve many of its greatest mysteries:

> Perhaps hyper-advanced life isn't just external. Perhaps it's already all around. It is embedded in what we perceive to be physics itself, from the root behaviour of particles and fields to the phenomena of complexity and emergence ... In other words, life might not just be in the equations. It might BE the equations [My emphasis].

Scharf said it is possible that 'we don't recognise advanced life because it forms an integral and unsuspicious part of what we've considered to be the natural world'. I agree. Wetiko/Yaldabaoth *is* the simulation. We are literally in the body of the beast. But that doesn't mean it has to control us. We all have the power to overcome Wetiko

influence and the Cult knows that. I doubt it sleeps too well because it knows that.

Which Field?

This, I suggest, is how it all works. There are two Fields. One is the fierce electromagnetic light of the Matrix within the speed of light; the other is the 'watery light' of The Field beyond the walls of the Matrix that connects with the Great Infinity. Five-sense mind and the decoding systems of the body attach us to the Field of Matrix light. They have to or we could not experience this reality. Five-sense mind sees only the Matrix Field of information while our expanded consciousness is part of the Infinity Field. When we open our minds, and most importantly our hearts, to the Infinity Field we have a mission control which gives us an expanded perspective, a road map, to understand the nature of the five-sense world. If we are isolated only in five-sense mind there is no mission control. We're on our own trying to understand a world that's constantly feeding us information to ensure we do not understand. People in this state can feel 'lost' and bewildered with no direction or radar. You can see ever more clearly those who are influenced by the Fields of Big Infinity or little five-sense mind simply by their views and behaviour with regard to the 'Covid' hoax. We have had this division throughout known human history with the mass of the people on one side and individuals who could see and intuit beyond the walls of the simulation – Plato's prisoner who broke out of the cave and saw reality for what it is. Such people have always been targeted by Wetiko/Archon-possessed authority, burned at the stake or demonised as mad, bad and dangerous. The Cult today and its global network of 'anti-hate', 'anti-fascist' Woke groups are all expressions of Wetiko attacking those exposing the conspiracy, 'Covid' lies and the 'vaccine' agenda.

Woke as a whole is Wetiko which explains its black and white mentality and how at one it is with the Wetiko-possessed Cult. Paul Levy said: 'To be in this paradigm is to still be under the thrall of a two-valued logic – where things are either true or false – of a

wetikoized mind.' Wetiko consciousness is in a permanent rage, therefore so is Woke, and then there is Woke inversion and contradiction. 'Anti-fascists' act like fascists because fascists *and* 'anti-fascists' are both Wetiko at work. Political parties act the same while claiming to be different for the same reason. Secret society and satanic rituals are attaching initiates to Wetiko and the cold, ruthless, psychopathic mentality that secures the positions of power all over the world is Wetiko. Reframing 'training programmes' have the same cumulative effect of attaching Wetiko and we have their graduates described as automatons and robots with a cold, psychopathic, uncaring demeanour. They are all traits of Wetiko possession and look how many times they have been described in this book and elsewhere with regard to personnel behind 'Covid' including the police and medical profession. Climbing the greasy pole in any profession in a Wetiko society requires traits of Wetiko to get there and that is particularly true of politics which is not about fair competition and pre-eminence of ideas. It is founded on how many backs you can stab and arses you can lick. This culminated in the global 'Covid' coordination between the Wetiko possessed who pulled it off in all the different countries without a trace of empathy and compassion for their impact on humans. Our sight sense can see only holographic form and not the Field which connects holographic form. Therefore we perceive 'physical' objects with 'space' in between. In fact that 'space' is energy/consciousness operating on multiple frequencies. One of them is Wetiko and that connects the Cult psychopaths, those who submit to the psychopaths, and those who serve the psychopaths in the media operations of the world. Wetiko is Gates. Wetiko is the mask-wearing submissive. Wetiko is the fake journalist and 'fact-checker'. The Wetiko Field is coordinating the whole thing. Psychopaths, gofers, media operatives, 'anti-hate' hate groups, 'fact-checkers' and submissive people work as one unit *even without human coordination* because they are attached to the *same* Field which is organising it all (Fig 22). Paul Levy is here describing how Wetiko-possessed people are drawn together and refuse to let any information breach their rigid

perceptions. He was writing long before 'Covid', but I think you will recognise followers of the 'Covid' religion *oh just a little bit*:

> People who are channelling the vibratory frequency of wetiko align with each other through psychic resonance to reinforce their unspoken shared agreement so as to uphold their deranged view of reality. Once an unconscious content takes possession of certain individuals, it irresistibly draws them together by mutual attraction and knits them into groups tied together by their shared madness that can easily swell into an avalanche of insanity.
>
> A psychic epidemic is a closed system, which is to say that it is insular and not open to any new information or informing influences from the outside world which contradict its fixed, limited, and limiting perspective.

There we have the Woke mind and the 'Covid' mind. Compatible resonance draws the awakening together, too, which is clearly happening today.

Figure 22: The Wetiko Field from which the Cult pyramid and its personnel are made manifest. (Image by Neil Hague).

Spiritual servitude

Wetiko doesn't care about humans. It's not human; it just possesses humans for its own ends and the effect (depending on the scale of

possession) can be anything from extreme psychopathy to unquestioning obedience. Wetiko's worst nightmare is for human consciousness to expand beyond the simulation. Everything is focussed on stopping that happening through control of information, thus perception, thus frequency. The 'education system', media, science, medicine, academia, are all geared to maintaining humanity in five-sense servitude as is the constant stimulation of low-vibrational mental and emotional states (see 'Covid'). Wetiko seeks to dominate those subconscious spaces between five-sense perception and expanded consciousness where the computer meets the operator. From these subconscious hiding places Wetiko speaks to us to trigger urges and desires that we take to be our own and manipulate us into anything from low-vibrational to psychopathic states. Remember how Islam describes the Jinn as invisible tricksters that 'whisper' and confuse. Wetiko is the origin of the 'trickster god' theme that you find in cultures all over the world. Jinn, like the Archons, are Wetiko which is terrified of humans awakening and reconnecting with our true self for then its energy source has gone. With that the feedback loop breaks between Wetiko and human perception that provides the energetic momentum on which its very existence depends as a force of evil. Humans are both its target and its source of survival, but only if we are operating in low-vibrational states of fear, hate, depression and the background anxiety that most people suffer. We are Wetiko's target because we are its key to survival. It needs us, not the other way round. Paul Levy writes:

> A vampire has no intrinsic, independent, substantial existence in its own right; it only exists in relation to us. The pathogenic, vampiric mind-parasite called wetiko is nothing in itself – not being able to exist from its own side – yet it has a 'virtual reality' such that it can potentially destroy our species …
>
> …The fact that a vampire is not reflected by a mirror can also mean that what we need to see is that there's nothing, no-thing to see, other than ourselves. The fact that wetiko is the expression of something inside of us means that the cure for wetiko is with us as well. The critical issue is finding this cure within us and then putting it into effect.

Evil begets evil because if evil does not constantly expand and find new sources of energetic sustenance its evil, its *distortion*, dies with the assimilation into balance and harmony. Love is the garlic to Wetiko's vampire. Evil, the absence of love, cannot exist in the presence of love. I think I see a way out of here. I have emphasised so many times over the decades that the Archons/Wetiko and their Cult are not all powerful. *They are not*. I don't care how it looks even now *they are not*. I have not called them little boys in short trousers for effect. I have said it because it is true. Wetiko's insatiable desire for power over others is not a sign of its omnipotence, but its insecurity. Paul Levy writes: 'Due to the primal fear which ultimately drives it and which it is driven to cultivate, wetiko's body politic has an intrinsic and insistent need for centralising power and control so as to create imagined safety for itself.' *Yeeeeeees!* Exactly! Why does Wetiko want humans in an ongoing state of fear? Wetiko itself *is* fear and it is petrified of love. As evil is an absence of love, so love is an absence of fear. Love conquers all and *especially* Wetiko which *is* fear. Wetiko brought fear into the world when it wasn't here before. *Fear* was the 'fall', the fall into low-frequency ignorance and illusion – fear is **F**alse **E**motion **A**ppearing **R**eal. The simulation is driven and energised by fear because Wetiko/Yaldabaoth (fear) *are* the simulation. Fear is the absence of love and Wetiko is the absence of love.

Wetiko today

We can now view current events from this level of perspective. The 'Covid' hoax has generated momentous amounts of ongoing fear, anxiety, depression and despair which have empowered Wetiko. No wonder people like Gates have been the instigators when they are Wetiko incarnate and exhibit every trait of Wetiko in the extreme. See how cold and unemotional these people are like Gates and his cronies, how dead of eye they are. That's Wetiko. Sabbatians are Wetiko and everything they control including the World Health Organization, Big Pharma and the 'vaccine' makers, national 'health'

hierarchies, corporate media, Silicon Valley, the banking system, and the United Nations with its planned transformation into world government. All are controlled and possessed by the Wetiko distortion into distorting human society in its image. We are with this knowledge at the gateway to understanding the world. Divisions of race, culture, creed and sexuality are diversions to hide the real division between those possessed and influenced by Wetiko and those that are not. The 'Covid' hoax has brought both clearly into view. Human behaviour is not about race. Tyrants and dictatorships come in all colours and creeds. What unites the US president bombing the innocent and an African tribe committing genocide against another as in Rwanda? What unites them? *Wetiko*. All wars are Wetiko, all genocide is Wetiko, all hunger over centuries in a world of plenty is Wetiko. Children going to bed hungry, including in the West, is Wetiko. Cult-generated Woke racial divisions that focus on the body are designed to obscure the reality that divisions in behaviour are manifestations of mind, not body. Obsession with body identity and group judgement is a means to divert attention from the real source of behaviour – mind and perception. Conflict sown by the Woke both within themselves and with their target groups are Wetiko providing lunch for itself through still more agents of the division, chaos, and fear on which it feeds. The Cult is seeking to assimilate the entirety of humanity and all children and young people into the Wetiko frequency by manipulating them into states of fear and despair. Witness all the suicide and psychological unravelling since the spring of 2020. Wetiko psychopaths want to impose a state of unquestioning obedience to authority which is no more than a conduit for Wetiko to enforce its will and assimilate humanity into itself. It needs us to believe that resistance is futile when it fears resistance and even more so the game-changing non-cooperation with its impositions. It can use violent resistance for its benefit. Violent impositions and violent resistance are *both* Wetiko. The Power of Love with its Power of No will sweep Wetiko from our world. Wetiko and its Cult know that. They just don't want us to know.

AI Wetiko

This brings me to AI or artificial intelligence and something else Wetikos don't want us to know. What is AI *really*? I know about computer code algorithms and AI that learns from data input. These, however, are more diversions, the expeditionary force, for the real AI that they want to connect to the human brain as promoted by Silicon Valley Wetikos like Kurzweil. What is this AI? It is the frequency of *Wetiko*, the frequency of the Archons. The connection of AI to the human brain is the connection of the Wetiko frequency to create a Wetiko hive mind and complete the job of assimilation. The hive mind is planned to be controlled from Israel and China which are both 100 percent owned by Wetiko Sabbatians. The assimilation process has been going on minute by minute in the 'smart' era which fused with the 'Covid' era. We are told that social media is scrambling the minds of the young and changing their personality. This is true, but what is social media? Look more deeply at how it works, how it creates divisions and conflict, the hostility and cruelty, the targeting of people until they are destroyed. That's Wetiko. Social media is manipulated to tune people to the Wetiko frequency with all the emotional exploitation tricks employed by platforms like Facebook and its Wetiko front man, Zuckerberg. Facebook's Instagram announced a new platform for children to overcome a legal bar on them using the main site. This is more Wetiko exploitation and manipulation of kids. Amnesty International likened the plan to foxes offering to guard the henhouse and said it was incompatible with human rights. Since when did Wetiko or Zuckerberg (I repeat myself) care about that? Would Brin and Page at Google, Wojcicki at YouTube, Bezos at Amazon and whoever the hell runs Twitter act as they do if they were not channelling Wetiko? Would those who are developing technologies for no other reason than human control? How about those designing and selling technologies to kill people and Big Pharma drug and 'vaccine' producers who know they will end or devastate lives? Quite a thought for these people to consider is that if you are Wetiko in a human life you are Wetiko on the 'other side' unless your frequency

changes and that can only change by a change of perception which becomes a change of behaviour. Where Gates is going does not bear thinking about although perhaps that's exactly where he wants to go. Either way, that's where he's going. His frequency will make it so.

The frequency lair

I have been saying for a long time that a big part of the addiction to smartphones and devices is that a frequency is coming off them that entraps the mind. People spend ages on their phones and sometimes even a minute or so after they put them down they pick them up again and it all repeats. 'Covid' lockdowns will have increased this addiction a million times for obvious reasons. Addictions to alcohol overindulgence and drugs are another way that Wetiko entraps consciousness to attach to its own. Both are symptoms of low-vibrational psychological distress which alcoholism and drug addiction further compound. Do we think it's really a coincidence that access to them is made so easy while potions that can take people into realms beyond the simulation are banned and illegal? I have explored smartphone addiction in other books, the scale is mind-blowing, and that level of addiction does not come without help. Tech companies that make these phones are Wetiko and they will have no qualms about destroying the minds of children. We are seeing again with these companies the Wetiko perceptual combination of psychopathic enforcers and weak and meek unquestioning compliance by the rank and file.

The global Smart Grid is the Wetiko Grid and it is crucial to complete the Cult endgame. The simulation is radiation and we are being deluged with technological radiation on a devastating scale. Wetiko frauds like Elon Musk serve Cult interests while occasionally criticising them to maintain his street-cred. 5G and other forms of Wi-Fi are being directed at the earth from space on a volume and scale that goes on increasing by the day. Elon Musk's (officially) SpaceX Starlink project is in the process of putting tens of thousands of satellites in low orbit to cover every inch of the planet with 5G and other Wi-Fi to create Kurzweil's global 'cloud' to which the

human mind is planned to be attached very soon. SpaceX has approval to operate 12,000 satellites with more than 1,300 launched at the time of writing and applications filed for *30,000* more. Other operators in the Wi-Fi, 5G, low-orbit satellite market include OneWeb (UK), Telesat (Canada), and AST & Science (US). Musk tells us that AI could be the end of humanity and then launches a company called Neuralink to connect the human brain to computers. Musk's (in theory) Tesla company is building electric cars and the driverless vehicles of the smart control grid. As frauds and bullshitters go Elon Musk in my opinion is Major League.

5G and technological radiation in general are destructive to human health, genetics and psychology and increasing the strength of artificial radiation underpins the five-sense perceptual bubbles which are themselves expressions of radiation or electromagnetism. Freedom activist John Whitehead was so right with his 'databit by databit, we are building our own electronic concentration camps'. The Smart Grid and 5G is a means to control the human mind and infuse perceptual information into The Field to influence anyone in sync with its frequency. You can change perception and behaviour en masse if you can manipulate the population into those levels of frequency and this is happening all around us today. The arrogance of Musk and his fellow Cult operatives knows no bounds in the way that we see with Gates. Musk's satellites are so many in number already they are changing the night sky when viewed from Earth. The astronomy community has complained about this and they have seen nothing yet. Some consequences of Musk's Wetiko hubris include: Radiation; visible pollution of the night sky; interference with astronomy and meteorology; ground and water pollution from intensive use of increasingly many spaceports; accumulating space debris; continual deorbiting and burning up of aging satellites, polluting the atmosphere with toxic dust and smoke; and ever-increasing likelihood of collisions. A collective public open letter of complaint to Musk said:

We are writing to you … because SpaceX is in process of surrounding the Earth with a network of thousands of satellites whose very purpose is to irradiate every square inch of the

Earth. SpaceX, like everyone else, is treating the radiation as if it were not there. As if the mitochondria in our cells do not depend on electrons moving undisturbed from the food we digest to the oxygen we breathe.

As if our nervous systems and our hearts are not subject to radio frequency interference like any piece of electronic equipment. As if the cancer, diabetes, and heart disease that now afflict a majority of the Earth's population are not metabolic diseases that result from interference with our cellular machinery. As if insects everywhere, and the birds and animals that eat them, are not starving to death as a result.

People like Musk and Gates believe in their limitless Wetiko arrogance that they can do whatever they like to the world because they own it. Consequences for humanity are irrelevant. It's absolutely time that we stopped taking this shit from these self-styled masters of the Earth when you consider where this is going.

Why is the Cult so anti-human?

I hear this question often: Why would they do this when it will affect them, too? Ah, but will it? Who is this *them*? Forget their bodies. They are just vehicles for Wetiko consciousness. When you break it all down to the foundations we are looking at a state of severely distorted consciousness targeting another state of consciousness for assimilation. The rest is detail. The simulation is the fly-trap in which unique sensations of the five senses create a cycle of addiction called reincarnation. Renegade Minds see that everything which happens in our reality is a smaller version of the whole picture in line with the holographic principle. Addiction to the radiation of smart technology is a smaller version of addiction to the whole simulation. Connecting the body/brain to AI is taking that addiction on a giant step further to total ongoing control by assimilating human incarnate consciousness into Wetiko. I have watched during the 'Covid' hoax how many are becoming ever more profoundly attached to Wetiko's perceptual calling cards of aggressive response to any other point of view ('There is no other god but me'), psychopathic lack of compassion and empathy, and servile submission to the narrative and will of authority. Wetiko is the psychopaths *and* subservience to psychopaths. The Cult of Wetiko is

so anti-human because it is *not* human. It embarked on a mission to destroy human by targeting everything that it means to be human and to survive as human. 'Covid' is not the end, just a means to an end. The Cult with its Wetiko consciousness is seeking to change Earth systems, including the atmosphere, to suit them, not humans. The gathering bombardment of 5G alone from ground and space is dramatically changing The Field with which the five senses interact. There is so much more to come if we sit on our hands and hope it will all go away. It is not meant to go away. It is meant to get ever more extreme and we need to face that while we still can – just.

Carbon dioxide is the gas of life. Without that human is over. Kaput, gone, history. No natural world, no human. The Cult has created a cock and bull story about carbon dioxide and climate change to justify its reduction to the point where Gates and the ignoramus Biden 'climate chief' John Kerry want to suck it out of the atmosphere. Kerry wants to do this because his master Gates does. Wetikos have made the gas of life a demon with the usual support from the Wokers of Extinction Rebellion and similar organisations and the bewildered puppet-child that is Greta Thunberg who was put on the world stage by Klaus Schwab and the World Economic Forum. The name Extinction Rebellion is both ironic and as always Wetiko inversion. The gas that we need to survive must be reduced to save us from extinction. The most basic need of human is oxygen and we now have billions walking around in face nappies depriving body and brain of this essential requirement of human existence. More than that 5G at 60 gigahertz interacts with the oxygen molecule to reduce the amount of oxygen the body can absorb into the bloodstream. The obvious knock-on consequences of that for respiratory and cognitive problems and life itself need no further explanation. Psychopaths like Musk are assembling a global system of satellites to deluge the human atmosphere with this insanity. The man should be in jail. Here we have two most basic of human needs, oxygen and carbon dioxide, being dismantled.

Two others, water and food, are getting similar treatment with the United Nations Agendas 21 and 2030 – the Great Reset – planning to

centrally control all water and food supplies. People will not even own rain water that falls on their land. Food is affected at the most basic level by reducing carbon dioxide. We have genetic modification or GMO infiltrating the food chain on a mass scale, pesticides and herbicides polluting the air and destroying the soil. Freshwater fish that provide livelihoods for 60 million people and feed hundreds of millions worldwide are being 'pushed to the brink' according the conservationists while climate change is the only focus. Now we have Gates and Schwab wanting to dispense with current food sources all together and replace them with a synthetic version which the Wetiko Cult would control in terms of production and who eats and who doesn't. We have been on the Totalitarian Tiptoe to this for more than 60 years as food has become ever more processed and full of chemical shite to the point today when it's not natural food at all. As Dr Tom Cowan says: 'If it has a label don't eat it.' Bill Gates is now the biggest owner of farmland in the United States and he does nothing without an ulterior motive involving the Cult. Klaus Schwab wrote: 'To feed the world in the next 50 years we will need to produce as much food as was produced in the last 10,000 years … food security will only be achieved, however, if regulations on genetically modified foods are adapted to reflect the reality that gene editing offers a precise, efficient and safe method of improving crops.' Liar. People and the world are being targeted with aluminium through vaccines, chemtrails, food, drink cans, and endless other sources when aluminium has been linked to many health issues including dementia which is increasing year after year. Insects, bees and wildlife essential to the food chain are being deleted by pesticides, herbicides and radiation which 5G is dramatically increasing with 6G and 7G to come. The pollinating bee population is being devastated while wildlife including birds, dolphins and whales are having their natural radar blocked by the effects of ever-increasing radiation. In the summer windscreens used to be splattered with insects so numerous were they. It doesn't happen now. Where have they gone?

Synthetic everything

The Cult is introducing genetically-modified versions of trees, plants and insects including a Gates-funded project to unleash hundreds of millions of genetically-modified, lab-altered and patented male mosquitoes to mate with wild mosquitoes and induce genetic flaws that cause them to die out. Clinically-insane Gates-funded Japanese researchers have developed mosquitos that spread vaccine and are dubbed 'flying vaccinators'. Gates is funding the modification of weather patterns in part to sell the myth that this is caused by carbon dioxide and he's funding geoengineering of the skies to change the atmosphere. Some of this came to light with the Gates-backed plan to release tonnes of chalk into the atmosphere to 'deflect the Sun and cool the planet'. Funny how they do this while the heating effect of the Sun is not factored into climate projections focussed on carbon dioxide. The reason is that they want to reduce carbon dioxide (so don't mention the Sun), but at the same time they do want to reduce the impact of the Sun which is so essential to human life and health. I have mentioned the sun-cholesterol-vitamin D connection as they demonise the Sun with warnings about skin cancer (caused by the chemicals in sun cream they tell you to splash on). They come from the other end of the process with statin drugs to reduce cholesterol that turns sunlight into vitamin D. A lack of vitamin D leads to a long list of health effects and how vitamin D levels must have fallen with people confined to their homes over 'Covid'. Gates is funding other forms of geoengineering and most importantly chemtrails which are dropping heavy metals, aluminium and self-replicating nanotechnology onto the Earth which is killing the natural world. See *Everything You Need To Know, But Have Never Been Told* for the detailed background to this.

Every human system is being targeted for deletion by a force that's not human. The Wetiko Cult has embarked on the process of transforming the human body from biological to synthetic biological as I have explained. Biological is being replaced by the artificial and synthetic – Archontic 'countermimicry' – right across human society. The plan eventually is to dispense with the human body altogether

and absorb human consciousness – which it wouldn't really be by then – into cyberspace (the simulation which is Wetiko/Yaldabaoth). Preparations for that are already happening if people would care to look. The alternative media rightly warns about globalism and 'the globalists', but this is far bigger than that and represents the end of the human race as we know it. The 'bad copy' of prime reality that Gnostics describe was a bad copy of harmony, wonder and beauty to start with before Wetiko/Yaldabaoth set out to change the simulated 'copy' into something very different. The process was slow to start with. Entrapped humans in the simulation timeline were not technologically aware and they had to be brought up to intellectual speed while being suppressed spiritually to the point where they could build their own prison while having no idea they were doing so. We have now reached that stage where technological intellect has the potential to destroy us and that's why events are moving so fast. Central American shaman Don Juan Matus said:

> Think for a moment, and tell me how you would explain the contradictions between the intelligence of man the engineer and the stupidity of his systems of belief, or the stupidity of his contradictory behaviour. Sorcerers believe that the predators have given us our systems of beliefs, our ideas of good and evil; our social mores. They are the ones who set up our dreams of success or failure. They have given us covetousness, greed, and cowardice. It is the predator who makes us complacent, routinary, and egomaniacal.
>
> In order to keep us obedient and meek and weak, the predators engaged themselves in a stupendous manoeuvre – stupendous, of course, from the point of view of a fighting strategist; a horrendous manoeuvre from the point of those who suffer it. They gave us their mind. The predators' mind is baroque, contradictory, morose, filled with the fear of being discovered any minute now.

For 'predators' see Wetiko, Archons, Yaldabaoth, Jinn, and all the other versions of the same phenomenon in cultures and religions all over the world. The theme is always the same because it's true and it's real. We have reached the point where we have to deal with it. The question is – how?

Don't fight – walk away

I thought I'd use a controversial subheading to get things moving in terms of our response to global fascism. What do you mean 'don't fight'? What do you mean 'walk away'? We've got to fight. We can't walk away. Well, it depends what we mean by fight and walk away. If fighting means physical combat we are playing Wetiko's game and falling for its trap. It wants us to get angry, aggressive, and direct hate and hostility at the enemy we think we must fight. Every war, every battle, every conflict, has been fought with Wetiko leading both sides. It's what it does. Wetiko wants a fight, anywhere, any place. Just hit me, son, so I can hit you back. Wetiko hits Wetiko and Wetiko hits Wetiko in return. I am very forthright as you can see in exposing Wetikos of the Cult, but I don't hate them. I refuse to hate them. It's what they want. What you hate you become. What you *fight* you become. Wokers, 'anti-haters' and 'anti-fascists' prove this every time they reach for their keyboards or don their balaclavas. By walk away I mean to disengage from Wetiko which includes ceasing to cooperate with its tyranny. Paul Levy says of Wetiko:

> The way to 'defeat' evil is not to try to destroy it (for then, in playing evil's game, we have already lost), but rather, to find the invulnerable place within ourselves where evil is unable to vanquish us – this is to truly 'win' our battle with evil.

Wetiko is everywhere in human society and it's been on steroids since the 'Covid' hoax. Every shouting match over wearing masks has Wetiko wearing a mask and Wetiko not wearing one. It's an electrical circuit of push and resist, push and resist, with Wetiko pushing *and* resisting. Each polarity is Wetiko empowering itself. Dictionary definitions of 'resist' include 'opposing, refusing to accept or comply with' and the word to focus on is 'opposing'. What form does this take – setting police cars alight or 'refusing to accept or comply with'? The former is Wetiko opposing Wetiko while the other points the way forward. This is the difference between those aggressively demanding that government fascism must be obeyed who stand in stark contrast to the great majority of Pushbackers. We saw this clearly with a march by thousands of Pushbackers against lockdown in London followed days later by a Woker-hijacked

protest in Bristol in which police cars were set on fire. Masks were virtually absent in London and widespread in Bristol. Wetiko wants lockdown on every level of society and infuses its aggression to police it through its unknowing stooges. Lockdown protesters are the ones with the smiling faces and the hugs, The two blatantly obvious states of being – getting more obvious by the day – are the result of Wokers and their like becoming ever more influenced by the simulation Field of Wetiko and Pushbackers ever more influenced by The Field of a far higher vibration beyond the simulation. Wetiko can't invade the heart which is where most lockdown opponents are coming from. It's the heart that allows them to see through the lies to the truth in ways I will be highlighting.

Renegade Minds know that calmness is the place from which wisdom comes. You won't find wisdom in a hissing fit and wisdom is what we need in abundance right now. Calmness is not weakness – you don't have to scream at the top of your voice to be strong. Calmness is indeed a sign of strength. 'No' means I'm not doing it. *NOOOO!!!* doesn't mean you're not doing it even more. Volume does not advance 'No – I'm not doing it'. You are just not doing it. Wetiko possessed and influenced don't know how to deal with that. Wetiko wants a fight and we should not give it one. What it needs more than anything is our *cooperation* and we should not give that either. Mass rallies and marches are great in that they are a visual representation of feeling, but if it ends there they are irrelevant. You demand that Wetikos act differently? Well, they're not going to are they? They are Wetikos. We don't need to waste our time demanding that something doesn't happen when that will make no difference. We need to delete the means that *allows* it to happen. This, invariably, is our cooperation. You can demand a child stop firing a peashooter at the dog or you can refuse to buy the peashooter. If you provide the means you are cooperating with the dog being smacked on the nose with a pea. How can the authorities enforce mask-wearing if millions in a country refuse? What if the 74 million Pushbackers that voted for Trump in 2020 refused to wear masks, close their businesses or stay in their homes. It would be unenforceable. The

few control the many through the compliance of the many and that's always been the dynamic be it 'Covid' regulations or the Roman Empire. I know people can find it intimidating to say no to authority or stand out in a crowd for being the only one with a face on display; but it has to be done or it's over. I hope I've made clear in this book that where this is going will be far more intimidating than standing up now and saying 'No' – I will not cooperate with my own enslavement and that of my children. There might be consequences for some initially, although not so if enough do the same. The question that must be addressed is what is going to happen if we don't? It is time to be strong and unyieldingly so. No means no. Not here and there, but *everywhere* and *always*. I have refused to wear a mask and obey all the other nonsense. I will not comply with tyranny. I repeat: Fascism is not imposed by fascists – there are never enough of them. Fascism is imposed by the population acquiescing to fascism. *I will not do it*. I will die first, or my body will. Living meekly under fascism is a form of death anyway, the death of the spirit that Martin Luther King described.

Making things happen

We must not despair. This is not over till it's over and it's far from that. The 'fat lady' must refuse to sing. The longer the 'Covid' hoax has dragged on and impacted on more lives we have seen an awakening of phenomenal numbers of people worldwide to the realisation that what they have believed all their lives is not how the world really is. Research published by the system-serving University of Bristol and King's College London in February, 2021, concluded: 'One in every 11 people in Britain say they trust David Icke's take on the coronavirus pandemic.' It will be more by now and we have gathering numbers to build on. We must urgently progress from seeing the scam to ceasing to cooperate with it. Prominent German lawyer Reiner Fuellmich, also licenced to practice law in America, is doing a magnificent job taking the legal route to bring the psychopaths to justice through a second Nuremberg tribunal for crimes against humanity. Fuellmich has an impressive record of

beating the elite in court and he formed the German Corona Investigative Committee to pursue civil charges against the main perpetrators with a view to triggering criminal charges. Most importantly he has grasped the foundation of the hoax – the PCR test not testing for the 'virus' – and Christian Drosten is therefore on his charge sheet along with Gates frontman Tedros at the World Health Organization. Major players must be not be allowed to inflict their horrors on the human race without being brought to book. A life sentence must follow for Bill Gates and the rest of them. A group of researchers has also indicted the government of Norway for crimes against humanity with copies sent to the police and the International Criminal Court. The lawsuit cites participation in an internationally-planned false pandemic and violation of international law and human rights, the European Commission's definition of human rights by coercive rules, Nuremberg and Hague rules on fundamental human rights, and the Norwegian constitution. We must take the initiative from hereon and not just complain, protest and react.

There are practical ways to support vital mass non-cooperation. Organising in numbers is one. Lockdown marches in London in the spring in 2021 were mass non-cooperation that the authorities could not stop. There were too many people. Hundreds of thousands walked the London streets in the centre of the road for mile after mile while the Face-Nappies could only look on. They were determined, but calm, and just *did it* with no histrionics and lots of smiles. The police were impotent. Others are organising group shopping without masks for mutual support and imagine if that was happening all over. Policing it would be impossible. If the store refuses to serve people in these circumstances they would be faced with a long line of trolleys full of goods standing on their own and everything would have to be returned to the shelves. How would they cope with that if it kept happening? I am talking here about moving on from complaining to being pro-active; from watching things happen to making things happen. I include in this our relationship with the police. The behaviour of many Face-Nappies

has been disgraceful and anyone who thinks they would never find concentration camp guards in the 'enlightened' modern era have had that myth busted big-time. The period and setting may change – Wetikos never do. I watched film footage from a London march in which a police thug viciously kicked a protestor on the floor who had done nothing. His fellow Face-Nappies stood in a ring protecting him. What he did was a criminal assault and with a crowd far outnumbering the police this can no longer be allowed to happen unchallenged. I get it when people chant 'shame on you' in these circumstances, but that is no longer enough. They *have* no shame those who do this. Crowds needs to start making a citizen's arrest of the police who commit criminal offences and brutally attack innocent people and defenceless women. A citizen's arrest can be made under section 24A of the UK Police and Criminal Evidence (PACE) Act of 1984 and you will find something similar in other countries. I prefer to call it a Common Law arrest rather than citizen's for reasons I will come to shortly. Anyone can arrest a person committing an indictable offence or if they have reasonable grounds to suspect they are committing an indictable offence. On both counts the attack by the police thug would have fallen into this category. A citizen's arrest can be made to stop someone:

- Causing physical injury to himself or any other person
- Suffering physical injury
- Causing loss of or damage to property
- Making off before a constable can assume responsibility for him

A citizen's arrest may also be made to prevent a breach of the peace under Common Law and if they believe a breach of the peace will happen or anything related to harm likely to be done or already done in their presence. This is the way to go I think – the Common Law version. If police know that the crowd and members of the public will no longer be standing and watching while they commit

their thuggery and crimes they will think twice about acting like Brownshirts and Blackshirts.

Common Law – common sense

Mention of Common Law is very important. Most people think the law is the law as in one law. This is not the case. There are two bodies of law, Common Law and Statute Law, and they are not the same. Common Law is founded on the simple premise of do no harm. It does not recognise victimless crimes in which no harm is done while Statute Law does. There is a Statute Law against almost everything. So what is Statute Law? Amazingly it's the law of the *sea* that was brought ashore by the Cult to override the law of the land which is Common Law. They had no right to do this and as always they did it anyway. They had to. They could not impose their will on the people through Common Law which only applies to do no harm. How could you stitch up the fine detail of people's lives with that? Instead they took the law of the sea, or Admiralty Law, and applied it to the population. Statute Law refers to all the laws spewing out of governments and their agencies including all the fascist laws and regulations relating to 'Covid'. The key point to make is that Statute Law is *contract law*. It only applies between *contracting* corporations. Most police officers don't even know this. They have to be kept in the dark, too. Long ago when merchants and their sailing ships began to trade with different countries a contractual law was developed called Admiralty Law and other names. Again it only applied to *contracts* agreed between *corporate* entities. If there is no agreed contract the law of the sea had no jurisdiction *and that still applies to its new alias of Statute Law*. The problem for the Cult when the law of the sea was brought ashore was an obvious one. People were not corporations and neither were government entities. To overcome the latter they made governments and all associated organisations corporations. All the institutions are *private corporations* and I mean governments and their agencies, local councils, police, courts, military, US states, the whole lot. Go to the

Dun and Bradstreet corporate listings website for confirmation that they are all corporations. You are arrested by a private corporation called the police by someone who is really a private security guard and they take you to court which is another private corporation. Neither have jurisdiction over you unless you consent and *contract* with them. This is why you hear the mantra about law enforcement policing by *consent* of the people. In truth the people 'consent' only in theory through monumental trickery.

Okay, the Cult overcame the corporate law problem by making governments and institutions corporate entities; but what about people? They are not corporations are they? Ah ... well in a sense, and *only* a sense, they are. Not people exactly – the illusion of people. The Cult creates a corporation in the name of everyone at the time that their birth certificate is issued. Note birth/ *berth* certificate and when you go to court under the law of the sea on land you stand in a *dock*. These are throwbacks to the origin. My Common Law name is David Vaughan Icke. The name of the corporation created by the government when I was born is called Mr David Vaughan Icke usually written in capitals as MR DAVID VAUGHAN ICKE. That is not me, the living, breathing man. It is a fictitious corporate entity. The trick is to make you think that David Vaughan Icke and MR DAVID VAUGHAN ICKE are the same thing. *They are not*. When police charge you and take you to court they are prosecuting the corporate entity and not the living, breathing, man or woman. They have to trick you into identifying as the corporate entity and contracting with them. Otherwise they have no jurisdiction. They do this through a language known as legalese. Lawful and legal are not the same either. Lawful relates to Common Law and legal relates to Statute Law. Legalese is the language of Statue Law which uses terms that mean one thing to the public and another in legalese. Notice that when a police officer tells someone why they are being charged he or she will say at the end: 'Do you understand?' To the public that means 'Do you comprehend?' In legalese it means 'Do you stand under me?' Do you stand under my authority? If you say

yes to the question you are unknowingly agreeing to give them jurisdiction over you in a contract between two corporate entities.

This is a confidence trick in every way. Contracts have to be agreed between informed parties and if you don't know that David Vaughan Icke is agreeing to be the corporation MR DAVID VAUGHAN ICKE you cannot knowingly agree to contract. They are deceiving you and another way they do this is to ask for proof of identity. You usually show them a driving licence or other document on which your corporate name is written. In doing so you are accepting that you are that corporate entity when you are not. Referring to yourself as a 'person' or 'citizen' is also identifying with your corporate fiction which is why I made the Common Law point about the citizen's arrest. If you are approached by a police officer you identify yourself immediately as a living, breathing, man or woman and say 'I do not consent, I do not contract with you and I do not understand' or stand under their authority. I have a Common Law birth certificate as a living man and these are available at no charge from commonlawcourt.com. Businesses registered under the Statute Law system means that its laws apply. There are, however, ways to run a business under Common Law. Remember all 'Covid' laws and regulations are Statute Law – the law of *contracts* and you do not have to contract. This doesn't mean that you can kill someone and get away with it. Common Law says do no harm and that applies to physical harm, financial harm etc. Police are employees of private corporations and there needs to be a new system of non-corporate Common Law constables operating outside the Statute Law system. If you go to davidicke.com and put Common Law into the search engine you will find videos that explain Common Law in much greater detail. It is definitely a road we should walk.

With all my heart

I have heard people say that we are in a spiritual war. I don't like the term 'war' with its Wetiko dynamic, but I know what they mean. Sweep aside all the bodily forms and we are in a situation in which two states of consciousness are seeking very different realities.

Wetiko wants upheaval, chaos, fear, suffering, conflict and control. The other wants love, peace, harmony, fairness and freedom. That's where we are. We should not fall for the idea that Wetiko is all-powerful and there's nothing we can do. Wetiko is not all-powerful. It's a joke, pathetic. It doesn't have to be, but it has made that choice for now. A handful of times over the years when I have felt the presence of its frequency I have allowed it to attach briefly so I could consciously observe its nature. The experience is not pleasant, the energy is heavy and dark, but the ease with which you can kick it back out the door shows that its real power is in persuading us that it has power. It's all a con. Wetiko is a con. It's a trickster and not a power that can control us if we unleash our own. The con is founded on manipulating humanity to give its power to Wetiko which recycles it back to present the illusion that it has power when its power is *ours* that we gave away. This happens on an energetic level and plays out in the world of the seen as humanity giving its power to Wetiko authority which uses that power to control the population when the power is only the power the population has handed over. How could it be any other way for billions to be controlled by a relative few? I have had experiences with people possessed by Wetiko and again you can kick its arse if you do it with an open heart. Oh yes – the *heart* which can transform the world of perceived 'matter'.

We are receiver-transmitters and processors of information, but what information and where from? Information is processed into perception in three main areas – the brain, the heart and the belly. These relate to thinking, knowing, and emotion. Wetiko wants us to be head and belly people which means we think within the confines of the Matrix simulation and low-vibrational emotional reaction scrambles balance and perception. A few minutes on social media and you see how emotion is the dominant force. Woke is all emotion and is therefore thought-free and fact-free. Our heart is something different. It *knows* while the head *thinks* and has to try to work it out because it doesn't know. The human energy field has seven prime vortexes which connect us with wider reality (Fig 23). Chakra means

'wheels of light' in the Sanskrit language of ancient India. The main ones are: The crown chakra on top of the head; brow (or 'third eye') chakra in the centre of the forehead; throat chakra; heart chakra in the centre of the chest; solar plexus chakra below the sternum; sacral chakra beneath the navel; and base chakra at the bottom of the spine. Each one has a particular function or functions. We feel anxiety and nervousness in the belly where the sacral chakra is located and this processes emotion that can affect the colon to give people 'the shits' or make them 'shit scared' when they are nervous. Chakras all play an important role, but the Mr and Mrs Big is the heart chakra which sits at the centre of the seven, above the chakras that connect us to the 'physical' and below those that connect with higher realms (or at least should). Here in the heart chakra we feel love, empathy and compassion – 'My heart goes out to you'. Those with closed hearts become literally 'heart-less' in their attitudes and behaviour (see Bill Gates). Native Americans portrayed Wetiko with what Paul Levy calls a 'frigid, icy heart, devoid of mercy' (see Bill Gates).

Figure 23: The chakra system which interpenetrates the human energy field. The heart chakra is the governor – or should be.

Wetiko trembles at the thought of heart energy which it cannot infiltrate. The frequency is too high. What it seeks to do instead is close the heart chakra vortex to block its perceptual and energetic influence. Psychopaths have 'hearts of stone' and emotionally-damaged people have 'heartache' and 'broken hearts'. The astonishing amount of heart disease is related to heart chakra

disruption with its fundamental connection to the 'physical' heart. Dr Tom Cowan has written an outstanding book challenging the belief that the heart is a pump and making the connection between the 'physical' and spiritual heart. Rudolph Steiner who was way ahead of his time said the same about the fallacy that the heart is a pump. *What*? The heart is not a pump? That's crazy, right? Everybody knows that. Read Cowan's *Human Heart, Cosmic Heart* and you will realise that the very idea of the heart as a pump is ridiculous when you see the evidence. How does blood in the feet so far from the heart get pumped horizontally up the body by the heart?? Cowan explains in the book the real reason why blood moves as it does. Our 'physical' heart is used to symbolise love when the source is really the heart vortex or spiritual heart which is our most powerful energetic connection to 'out there' expanded consciousness. That's why we feel *knowing* – intuitive knowing – in the centre of the chest. Knowing doesn't come from a process of thoughts leading to a conclusion. It is there in an instant all in one go. Our heart knows because of its connection to levels of awareness that *do* know. This is the meaning and source of intuition – intuitive *knowing*.

For the last more than 30 years of uncovering the global game and the nature of reality my heart has been my constant antenna for truth and accuracy. An American intelligence insider once said that I had quoted a disinformer in one of my books and yet I had only quoted the part that was true. He asked: 'How do you do that?' By using my heart antenna was the answer and anyone can do it. Heart-centred is how we are meant to be. With a closed heart chakra we withdraw into a closed mind and the bubble of five-sense reality. If you take a moment to focus your attention on the centre of your chest, picture a spinning wheel of light and see it opening and expanding. You will feel it happening, too, and perceptions of the heart like joy and love as the heart impacts on the mind as they interact. The more the chakra opens the more you will feel expressions of heart consciousness and as the process continues, and becomes part of you, insights and knowings will follow. An open

heart is connected to that level of awareness that knows all is *One*. You will see from its perspective that the fault-lines that divide us are only illusions to control us. An open heart does not process the illusions of race, creed and sexuality except as brief experiences for a consciousness that is all. Our heart does not see division, only unity (Figs 24 and 25). There's something else, too. Our hearts love to laugh. Mark Twain's quote that says 'The human race has one really effective weapon, and that is laughter' is really a reference to the heart which loves to laugh with the joy of knowing the true nature of infinite reality and that all the madness of human society is an illusion of the mind. Twain also said: 'Against the assault of laughter nothing can stand.' This is so true of Wetiko and the Cult. Their insecurity demands that they be taken seriously and their power and authority acknowledged and feared. We should do nothing of the sort. We should not get aggressive or fearful which their insecurity so desires. We should laugh in their face. Even in their no-face as police come over in their face-nappies and expect to be taken seriously. They don't take themselves seriously looking like that so why should we? Laugh in the face of intimidation. Laugh in the face of tyranny. You will see by its reaction that you have pressed all of its buttons. Wetiko does not know what to do in the face of laughter or when its targets refuse to concede their joy to fear. We have seen many examples during the 'Covid' hoax when people have expressed their energetic power and the string puppets of Wetiko retreat with their tail limp between their knees. Laugh – the world is bloody mad after all and if it's a choice between laughter and tears I know which way I'm going.

Figure 24: Head consciousness without the heart sees division and everything apart from everything else.

Figure 25: Heart consciousness sees everything as One.

'Vaccines' and the soul

The foundation of Wetiko/Archon control of humans is the separation of incarnate five-sense mind from the infinite 'I' and closing the heart chakra where the True 'I' lives during a human life. The goal has been to achieve complete separation in both cases. I was interested therefore to read an account by a French energetic healer of what she said she experienced with a patient who had been given the 'Covid' vaccine. Genuine energy healers can sense information and consciousness fields at different levels of being which are referred to as 'subtle bodies'. She described treating the patient who later returned after having, without the healer's knowledge, two doses of the 'Covid vaccine'. The healer said:

I noticed immediately the change, very heavy energy emanating from [the] subtle bodies. The scariest thing was when I was working on the heart chakra, I connected with her soul: it was detached from the physical body, it had no contact and it was, as if it was floating in a state of total confusion: a damage to the consciousness that loses contact with the physical body, i.e. with our biological machine, there is no longer any communication between them.

I continued the treatment by sending light to the heart chakra, the soul of the person, but it seemed that the soul could no longer receive any light, frequency or energy. It was a very powerful experience for me. Then I understood that this substance is indeed used to detach consciousness so that this consciousness can no longer interact through this body that it possesses in life, where there is no longer any contact, no frequency, no light, no more energetic balance or mind.

This would create a human that is rudderless and at the extreme almost zombie-like operating with a fractional state of consciousness at the mercy of Wetiko. I was especially intrigued by what the healer said in the light of the prediction by the highly-informed Rudolf Steiner more than a hundred years ago. He said:

> In the future, we will eliminate the soul with medicine. Under the pretext of a 'healthy point of view', there will be a vaccine by which the human body will be treated as soon as possible directly at birth, so that the human being cannot develop the thought of the existence of soul and Spirit. To materialistic doctors will be entrusted the task of removing the soul of humanity.
>
> As today, people are vaccinated against this disease or that disease, so in the future, children will be vaccinated with a substance that can be produced precisely in such a way that people, thanks to this vaccination, will be immune to being subjected to the 'madness' of spiritual life. He would be extremely smart, but he would not develop a conscience, and that is the true goal of some materialistic circles.

Steiner said the vaccine would detach the physical body from the etheric body (subtle bodies) and 'once the etheric body is detached the relationship between the universe and the etheric body would become extremely unstable, and man would become an automaton'. He said 'the physical body of man must be polished on this Earth by spiritual will – so the vaccine becomes a kind of arymanique (Wetiko) force' and 'man can no longer get rid of a given materialistic feeling'. Humans would then, he said, become 'materialistic of constitution and can no longer rise to the spiritual'. I have been writing for years about DNA being a receiver-transmitter of information that connects us to other levels of reality and these 'vaccines' changing DNA can be likened to changing an antenna and what it can transmit and receive. Such a disconnection would clearly lead to changes in personality and perception. Steiner further predicted the arrival of AI. Big Pharma 'Covid vaccine' makers, expressions of Wetiko, are testing their DNA-manipulating evil on children as I write with a view to giving the 'vaccine' to babies. If it's a soul-body disconnector – and I say that it is or can be – every child would be disconnected from 'soul' at birth and the 'vaccine' would create a closed system in which spiritual guidance from the greater self would play no part. This has been the ambition of Wetiko all

along. A Pentagon video from 2005 was leaked of a presentation explaining the development of vaccines to change behaviour by their effect on the brain. Those that believe this is not happening with the 'Covid' genetically-modifying procedure masquerading as a 'vaccine' should make an urgent appointment with Naivety Anonymous. Klaus Schwab wrote in 2018:

> Neurotechnologies enable us to better influence consciousness and thought and to understand many activities of the brain. They include decoding what we are thinking in fine levels of detail through new chemicals and interventions that can influence our brains to correct for errors or enhance functionality.

The plan is clear and only the heart can stop it. With every heart that opens, every mind that awakens, Wetiko is weakened. Heart and love are far more powerful than head and hate and so nothing like a majority is needed to turn this around.

Beyond the Phantom

Our heart is the prime target of Wetiko and so it must be the answer to Wetiko. We *are* our heart which is part of one heart, the infinite heart. Our heart is where the true self lives in a human life behind firewalls of five-sense illusion when an imposter takes its place – *Phantom Self*; but our heart waits patiently to be set free any time we choose to see beyond the Phantom, beyond Wetiko. A Wetikoed Phantom Self can wreak mass death and destruction while the love of forever is locked away in its heart. The time is here to unleash its power and let it sweep away the fear and despair that is Wetiko. Heart consciousness does not seek manipulated, censored, advantage for its belief or religion, its activism and desires. As an expression of the One it treats all as One with the same rights to freedom and opinion. Our heart demands fairness for itself no more than for others. From this unity of heart we can come together in mutual support and transform this Wetikoed world into what reality is meant to be – a place of love, joy, happiness, fairness, justice and freedom. Wetiko has another agenda and that's why the world is as

it is, but enough of this nonsense. Wetiko can't stay where hearts are open and it works so hard to keep them closed. Fear is its currency and its food source and love in its true sense has no fear. Why would love have fear when it knows it is *All That Is, Has Been, And Ever Can Be* on an eternal exploration of all possibility? Love in this true sense is not the physical attraction that passes for love. This can be an expression of it, yes, but Infinite Love, a love without condition, goes far deeper to the core of all being. It *is* the core of all being. Infinite realty was born from love beyond the illusions of the simulation. Love infinitely expressed is the knowing that all is One and the swiftly-passing experience of separation is a temporary hallucination. You cannot disconnect from Oneness; you can only *perceive* that you have and withdraw from its influence. This is the most important of all perception trickery by the mind parasite that is Wetiko and the foundation of all its potential for manipulation.

If we open our hearts, open the sluice gates of the mind, and redefine self-identity amazing things start to happen. Consciousness expands or contracts in accordance with self-identity. When true self is recognised as infinite awareness and label self – Phantom Self – is seen as only a series of brief experiences life is transformed. Consciousness expands to the extent that self-identity expands and everything changes. You see unity, not division, the picture, not the pixels. From this we can play the long game. No more is an experience something in and of itself, but a fleeting moment in the eternity of forever. Suddenly people in uniform and dark suits are no longer intimidating. Doing what your heart knows to be right is no longer intimidating and consequences for those actions take on the same nature of a brief experience that passes in the blink of an infinite eye. Intimidation is all in the mind. Beyond the mind there is no intimidation.

An open heart does not consider consequences for what it knows to be right. To do so would be to consider not doing what it knows to be right and for a heart in its power that is never an option. The Renegade Mind is really the Renegade Heart. Consideration of consequences will always provide a getaway car for the mind and

the heart doesn't want one. What is right in the light of what we face today is to stop cooperating with Wetiko in all its forms and to do it without fear or compromise. You cannot compromise with tyranny when tyranny always demands more until it has everything. Life is your perception and you are your destiny. Change your perception and you change your life. Change collective perception and we change the world.

Come on people … One human family, One heart, One goal … *FREEEEEEDOM*!

We must settle for nothing less.

Postscript

The big scare story as the book goes to press is the 'Indian' variant and the world is being deluged with propaganda about the 'Covid catastrophe' in India which mirrors in its lies and misrepresentations what happened in Italy before the first lockdown in 2020.

The *New York Post* published a picture of someone who had 'collapsed in the street from Covid' in India in April, 2021, which was actually taken during a gas leak in May, 2020. Same old, same old. Media articles in mid-February were asking why India had been so untouched by 'Covid' and then as their vaccine rollout gathered pace the alleged 'cases' began to rapidly increase. Indian 'Covid vaccine' maker Bharat Biotech was funded into existence by the Bill and Melinda Gates Foundation (the pair announced their divorce in May, 2021, which is a pity because they so deserve each other). The Indian 'Covid crisis' was ramped up by the media to terrify the world and prepare people for submission to still more restrictions. The scam that worked the first time was being repeated only with far more people seeing through the deceit. Davidicke.com and Ickonic.com have sought to tell the true story of what is happening by talking to people living through the Indian nightmare which has nothing to do with 'Covid'. We posted a letter from 'Alisha' in Pune who told a very different story to government and media mendacity. She said scenes of dying people and overwhelmed hospitals were designed to hide what was really happening – genocide and starvation. Alisha said that millions had already died of starvation during the ongoing lockdowns while government and media were lying and making it look like the 'virus':

Restaurants, shops, gyms, theatres, basically everything is shut. The cities are ghost towns. Even so-called 'essential' businesses are only open till 11am in the morning. You basically have just an hour to buy food and then your time is up.

Inter-state travel and even inter-district travel is banned. The cops wait at all major crossroads to question why you are traveling outdoors or to fine you if you are not wearing a mask.

The medical community here is also complicit in genocide, lying about hospitals being full and turning away people with genuine illnesses, who need immediate care. They have even created a shortage of oxygen cylinders.

This is the classic Cult modus operandi played out in every country. Alisha said that people who would not have a PCR test not testing for the 'virus' were being denied hospital treatment. She said the people hit hardest were migrant workers and those in rural areas. Most businesses employed migrant workers and with everything closed there were no jobs, no income and no food. As a result millions were dying of starvation or malnutrition. All this was happening under Prime Minister Narendra Modi, a 100-percent asset of the Cult, and it emphasises yet again the scale of pure anti-human evil we are dealing with. Australia banned its people from returning home from India with penalties for trying to do so of up to five years in jail and a fine of £37,000. The manufactured 'Covid' crisis in India was being prepared to justify further fascism in the West. Obvious connections could be seen between the Indian 'vaccine' programme and increased 'cases' and this became a common theme. The Seychelles, the most per capita 'Covid vaccinated' population in the world, went back into lockdown after a 'surge of cases'.

Long ago the truly evil Monsanto agricultural biotechnology corporation with its big connections to Bill Gates devastated Indian farming with genetically-modified crops. Human rights activist Gurcharan Singh highlighted the efforts by the Indian government to complete the job by destroying the food supply to hundreds of millions with 'Covid' lockdowns. He said that 415 million people at the bottom of the disgusting caste system (still going whatever they say) were below the poverty line and struggled to feed themselves every year. Now the government was imposing lockdown at just the

time to destroy the harvest. This deliberate policy was leading to mass starvation. People may reel back at the suggestion that a government would do that, but Wetiko-controlled 'leaders' are capable of any level of evil. In fact what is described in India is in the process of being instigated worldwide. The food chain and food supply are being targeted at every level to cause world hunger and thus control. Bill Gates is not the biggest owner of farmland in America for no reason and destroying access to food aids both the depopulation agenda and the plan for synthetic 'food' already being funded into existence by Gates. Add to this the coming hyper-inflation from the suicidal creation of fake 'money' in response to 'Covid' and the breakdown of container shipping systems and you have a cocktail that can only lead one way and is meant to. The Cult plan is to crash the entire system to 'build back better' with the Great Reset.

'Vaccine' transmission

Reports from all over the world continue to emerge of women suffering menstrual and fertility problems after having the fake 'vaccine' and of the non-'vaccinated' having similar problems when interacting with the 'vaccinated'. There are far too many for 'coincidence' to be credible. We've had menopausal women getting periods, others having periods stop or not stopping for weeks, passing clots, sometimes the lining of the uterus, breast irregularities, and miscarriages (which increased by 400 percent in parts of the United States). Non-'vaccinated' men and children have suffered blood clots and nose bleeding after interaction with the 'vaccinated'. Babies have died from the effects of breast milk from a 'vaccinated' mother. Awake doctors – the small minority – speculated on the cause of non-'vaccinated' suffering the same effects as the 'vaccinated'. Was it nanotechnology in the synthetic substance transmitting frequencies or was it a straight chemical bioweapon that was being transmitted between people? I am not saying that some kind of chemical transmission is not one possible answer, but the foundation of all that the Cult does is frequency and

this is fertile ground for understanding how transmission can happen. American doctor Carrie Madej, an internal medicine physician and osteopath, has been practicing for the last 20 years, teaching medical students, and she says attending different meetings where the agenda for humanity was discussed. Madej, who operates out of Georgia, did not dismiss other possible forms of transmission, but she focused on frequency in search of an explanation for transmission. She said the Moderna and Pfizer 'vaccines' contained nano-lipid particles as a key component. This was a brand new technology never before used on humanity. 'They're using a nanotechnology which is pretty much little tiny computer bits … nanobots or hydrogel.' Inside the 'vaccines' was 'this sci-fi kind of substance' which suppressed immune checkpoints to get into the cell. I referred to this earlier as the 'Trojan horse' technique that tricks the cell into opening a gateway for the self-replicating synthetic material and while the immune system is artificially suppressed the body has no defences. Madej said the substance served many purposes including an on-demand ability to 'deliver the payload' and using the nano 'computer bits' as biosensors in the body. 'It actually has the ability to accumulate data from your body, like your breathing, your respiration, thoughts, emotions, all kinds of things.'

She said the technology obviously has the ability to operate through Wi-Fi and transmit and receive energy, messages, frequencies or impulses. 'Just imagine you're getting this new substance in you and it can react to things all around you, the 5G, your smart device, your phones.' We had something completely foreign in the human body that had never been launched large scale at a time when we were seeing 5G going into schools and hospitals (plus the Musk satellites) and she believed the 'vaccine' transmission had something to do with this: '… if these people have this inside of them … it can act like an antenna and actually transmit it outwardly as well.' The synthetic substance produced its own voltage and so it could have that kind of effect. This fits with my own contention that the nano receiver-transmitters are designed to connect people to the

Smart Grid and break the receiver-transmitter connection to expanded consciousness. That would explain the French energy healer's experience of the disconnection of body from 'soul' with those who have had the 'vaccine'. The nanobots, self-replicating inside the body, would also transmit the synthetic frequency which could be picked up through close interaction by those who have not been 'vaccinated'. Madej speculated that perhaps it was 5G and increased levels of other radiation that was causing the symptoms directly although interestingly she said that non-'vaccinated' patients had shown improvement when they were away from the 'vaccinated' person they had interacted with. It must be remembered that you can control frequency and energy with your mind and you can consciously create energetic barriers or bubbles with the mind to stop damaging frequencies from penetrating your field. American paediatrician Dr Larry Palevsky said the 'vaccine' was not a 'vaccine' and was never designed to protect from a 'viral' infection. He called it 'a massive, brilliant propaganda of genocide' because they didn't have to inject everyone to get the result they wanted. He said the content of the jabs was able to infuse any material into the brain, heart, lungs, kidneys, liver, sperm and female productive system. 'This is genocide; this is a weapon of mass destruction.' At the same time American colleges were banning students from attending if they didn't have this life-changing and potentially life-ending 'vaccine'. Class action lawsuits must follow when the consequences of this college fascism come to light. As the book was going to press came reports about fertility effects on sperm in 'vaccinated' men which would absolutely fit with what I have been saying and hospitals continued to fill with 'vaccine' reactions. Another question is what about transmission via blood transfusions? The NHS has extended blood donation restrictions from seven days after a 'Covid vaccination' to 28 days after even a sore arm reaction.

I said in the spring of 2020 that the then touted 'Covid vaccine' would be ongoing each year like the flu jab. A year later Pfizer CEO, the appalling Albert Bourla, said people would 'likely' need a 'booster dose' of the 'vaccine' within 12 months of getting 'fully

vaccinated' and then a yearly shot. 'Variants will play a key role', he said confirming the point. Johnson & Johnson CEO Alex Gorsky also took time out from his 'vaccine' disaster to say that people may need to be vaccinated against 'Covid-19' each year. UK Health Secretary, the psychopath Matt Hancock, said additional 'boosters' would be available in the autumn of 2021. This is the trap of the 'vaccine passport'. The public will have to accept every last 'vaccine' they introduce, including for the fake 'variants', or it would cease to be valid. The only other way in some cases would be continuous testing with a test not testing for the 'virus' and what is on the swabs constantly pushed up your noise towards the brain every time?

'Vaccines' changing behaviour

I mentioned in the body of the book how I believed we would see gathering behaviour changes in the 'vaccinated' and I am already hearing such comments from the non-'vaccinated' describing behaviour changes in friends, loved ones and work colleagues. This will only increase as the self-replicating synthetic material and nanoparticles expand in body and brain. An article in the *Guardian* in 2016 detailed research at the University of Virginia in Charlottesville which developed a new method for controlling brain circuits associated with complex animal behaviour. The method, dubbed 'magnetogenetics', involves genetically-engineering a protein called ferritin, which stores and releases iron, to create a magnetised substance – 'Magneto' – that can activate specific groups of nerve cells from a distance. This is claimed to be an advance on other methods of brain activity manipulation known as optogenetics and chemogenetics (the Cult has been developing methods of brain control for a long time). The ferritin technique is said to be non-invasive and able to activate neurons 'rapidly and reversibly'. In other words, human thought and perception. The article said that earlier studies revealed how nerve cell proteins 'activated by heat and mechanical pressure can be genetically engineered so that they become sensitive to radio waves and magnetic fields, by attaching them to an iron-storing protein called ferritin, or to inorganic

paramagnetic particles'. Sensitive to radio waves and magnetic fields? You mean like 5G, 6G and 7G? This is the human-AI Smart Grid hive mind we are talking about. The *Guardian* article said:

> … the researchers injected Magneto into the striatum of freely behaving mice, a deep brain structure containing dopamine-producing neurons that are involved in reward and motivation, and then placed the animals into an apparatus split into magnetised and non-magnetised sections.
>
> Mice expressing Magneto spent far more time in the magnetised areas than mice that did not, because activation of the protein caused the striatal neurons expressing it to release dopamine, so that the mice found being in those areas rewarding. This shows that Magneto can remotely control the firing of neurons deep within the brain, and also control complex behaviours.

Make no mistake this basic methodology will be part of the 'Covid vaccine' cocktail and using magnetics to change brain function through electromagnetic field frequency activation. The Pentagon is developing a 'Covid vaccine' using ferritin. Magnetics would explain changes in behaviour and why videos are appearing across the Internet as I write showing how magnets stick to the skin at the point of the 'vaccine' shot. Once people take these 'vaccines' anything becomes possible in terms of brain function and illness which will be blamed on 'Covid-19' and 'variants'. Magnetic field manipulation would further explain why the non-'vaccinated' are reporting the same symptoms as the 'vaccinated' they interact with and why those symptoms are reported to decrease when not in their company. Interestingly 'Magneto', a 'mutant', is a character in the Marvel Comic *X-Men* stories with the ability to manipulate magnetic fields and he believes that mutants should fight back against their human oppressors by any means necessary. The character was born Erik Lehnsherr to a Jewish family in Germany.

Cult-controlled courts

The European Court of Human Rights opened the door for mandatory 'Covid-19 vaccines' across the continent when it ruled in a Czech Republic dispute over childhood immunisation that legally

enforced vaccination could be 'necessary in a democratic society'. The 17 judges decided that compulsory vaccinations did not breach human rights law. On the face of it the judgement was so inverted you gasp for air. If not having a vaccine infused into your body is not a human right then what is? Ah, but they said human rights law which has been specifically written to delete all human rights at the behest of the state (the Cult). Article 8 of the European Convention on Human Rights relates to the right to a private life. The crucial word here is *'except'*:

> There shall be no interference by a public authority with the exercise of this right EXCEPT such as is in accordance with the law and is necessary in a democratic society in the interests of national security, public safety or the economic wellbeing of the country, for the prevention of disorder or crime, for the protection of health or morals, or for the protection of the rights and freedoms of others [My emphasis].

No interference *except* in accordance with the law means there *are* no 'human rights' *except* what EU governments decide you can have at their behest. 'As is necessary in a democratic society' explains that reference in the judgement and 'in the interests of national security, public safety or the economic well-being of the country, for the prevention of disorder or crime, for the protection of health or morals, or for the protection of the rights and freedoms of others' gives the EU a coach and horses to ride through 'human rights' and scatter them in all directions. The judiciary is not a check and balance on government extremism; it is a vehicle to enforce it. This judgement was almost laughably predictable when the last thing the Cult wanted was a decision that went against mandatory vaccination. Judges rule over and over again to benefit the system of which they are a part. Vaccination disputes that come before them are invariably delivered in favour of doctors and authorities representing the view of the state which owns the judiciary. Oh, yes, and we have even had calls to stop putting 'Covid-19' on death certificates within 28 days of a 'positive test' because it is claimed the practice makes the 'vaccine' appear not to work. They are laughing at you.

The scale of madness, inhumanity and things to come was highlighted when those not 'vaccinated' for 'Covid' were refused evacuation from the Caribbean island of St Vincent during massive volcanic eruptions. Cruise ships taking residents to the safety of another island allowed only the 'vaccinated' to board and the rest were left to their fate. Even in life and death situations like this we see 'Covid' stripping people of their most basic human instincts and the insanity is even more extreme when you think that fake 'vaccine'-makers are not even claiming their body-manipulating concoctions stop 'infection' and 'transmission' of a 'virus' that doesn't exist. St Vincent Prime Minister Ralph Gonsalves said: 'The chief medical officer will be identifying the persons already vaccinated so that we can get them on the ship.' Note again the power of the chief medical officer who, like Whitty in the UK, will be answering to the World Health Organization. This is the Cult network structure that has overridden politicians who 'follow the science' which means doing what WHO-controlled 'medical officers' and 'science advisers' tell them. Gonsalves even said that residents who were 'vaccinated' after the order so they could board the ships would still be refused entry due to possible side effects such as 'wooziness in the head'. The good news is that if they were woozy enough in the head they could qualify to be prime minister of St Vincent.

Microchipping freedom

The European judgement will be used at some point to justify moves to enforce the 'Covid' DNA-manipulating procedure. Sandra Ro, CEO of the Global Blockchain Business Council, told a World Economic Forum event that she hoped 'vaccine passports' would help to 'drive forced consent and standardisation' of global digital identity schemes: 'I'm hoping with the desire and global demand for some sort of vaccine passport – so that people can get travelling and working again – [it] will drive forced consent, standardisation, and frankly, cooperation across the world.' The lady is either not very bright, or thoroughly mendacious, to use the term 'forced consent'.

You do not 'consent' if you are forced – you *submit*. She was describing what the plan has been all along and that's to enforce a digital identity on every human without which they could not function. 'Vaccine passports' are opening the door and are far from the end goal. A digital identity would allow you to be tracked in everything you do in cyberspace and this is the same technique used by Cult-owned China to enforce its social credit system of total control. The ultimate 'passport' is planned to be a microchip as my books have warned for nearly 30 years. Those nice people at the Pentagon working for the Cult-controlled Defense Advanced Research Projects Agency (DARPA) claimed in April, 2021, they have developed a microchip inserted under the skin to detect 'asymptomatic Covid-19 infection' before it becomes an outbreak and a 'revolutionary filter' that can remove the 'virus' from the blood when attached to a dialysis machine. The only problems with this are that the 'virus' does not exist and people transmitting the 'virus' with no symptoms is brain-numbing bullshit. This is, of course, not a ruse to get people to be microchipped for very different reasons. DARPA also said it was producing a one-stop 'vaccine' for the 'virus' and all 'variants'. One of the most sinister organisations on Planet Earth is doing this? Better have it then. These people are insane because Wetiko that possesses them is insane.

Researchers from the Salk Institute in California announced they have created an embryo that is part human and part monkey. My books going back to the 1990s have exposed experiments in top secret underground facilities in the United States where humans are being crossed with animal and non-human 'extraterrestrial' species. They are now easing that long-developed capability into the public arena and there is much more to come given we are dealing with psychiatric basket cases. Talking of which – Elon Musk's scientists at Neuralink trained a monkey to play Pong and other puzzles on a computer screen using a joystick and when the monkey made the correct move a metal tube squirted banana smoothie into his mouth which is the basic technique for training humans into unquestioning compliance. Two Neuralink chips were in the monkey's skull and

more than 2,000 wires 'fanned out' into its brain. Eventually the monkey played a video game purely with its brain waves. Psychopathic narcissist Musk said the 'breakthrough' was a step towards putting Neuralink chips into human skulls and merging minds with artificial intelligence. *Exactly*. This man is so dark and Cult to his DNA.

World Economic Fascism (WEF)

The World Economic Forum is telling you the plan by the statements made at its many and various events. Cult-owned fascist YouTube CEO Susan Wojcicki spoke at the 2021 WEF Global Technology Governance Summit (see the name) in which 40 governments and 150 companies met to ensure 'the responsible design and deployment of emerging technologies'. Orwellian translation: 'Ensuring the design and deployment of long-planned technologies will advance the Cult agenda for control and censorship.' Freedom-destroyer and Nuremberg-bound Wojcicki expressed support for tech platforms like hers to censor content that is 'technically legal but could be harmful'. Who decides what is 'harmful'? She does and they do. 'Harmful' will be whatever the Cult doesn't want people to see and we have legislation proposed by the UK government that would censor content on the basis of 'harm' no matter if the information is fair, legal and provably true. Make that *especially* if it is fair, legal and provably true. Wojcicki called for a global coalition to be formed to enforce content moderation standards through automated censorship. This is a woman and mega-censor so self-deluded that she shamelessly accepted a 'free expression' award – *Wojcicki* – in an event sponsored by her own *YouTube*. They have no shame and no self-awareness.

You know that 'Covid' is a scam and Wojcicki a Cult operative when YouTube is censoring medical and scientific opinion purely on the grounds of whether it supports or opposes the Cult 'Covid' narrative. Florida governor Ron DeSantis compiled an expert panel with four professors of medicine from Harvard, Oxford, and Stanford Universities who spoke against forcing children and

vaccinated people to wear masks. They also said there was no proof that lockdowns reduced spread or death rates of 'Covid-19'. Cult-gofer Wojcicki and her YouTube deleted the panel video 'because it included content that contradicts the consensus of local and global health authorities regarding the efficacy of masks to prevent the spread of Covid-19'. This 'consensus' refers to what the Cult tells the World Health Organization to say and the WHO tells 'local health authorities' to do. Wojcicki knows this, of course. The panellists pointed out that censorship of scientific debate was responsible for deaths from many causes, but Wojcicki couldn't care less. She would not dare go against what she is told and as a disgrace to humanity she wouldn't want to anyway. The UK government is seeking to pass a fascist 'Online Safety Bill' to specifically target with massive fines and other means non-censored video and social media platforms to make them censor 'lawful but harmful' content like the Cult-owned Facebook, Twitter, Google and YouTube. What is 'lawful but harmful' would be decided by the fascist Blair-created Ofcom.

Another WEF obsession is a cyber-attack on the financial system and this is clearly what the Cult has planned to take down the bank accounts of everyone – except theirs. Those that think they have enough money for the Cult agenda not to matter to them have got a big lesson coming if they continue to ignore what is staring them in the face. The World Economic Forum, funded by Gates and fronted by Klaus Schwab, announced it would be running a 'simulation' with the Russian government and global banks of just such an attack called Cyber Polygon 2021. What they simulate – as with the 'Covid' Event 201 – they plan to instigate. The WEF is involved in a project with the Cult-owned Carnegie Endowment for International Peace called the WEF-Carnegie Cyber Policy Initiative which seeks to merge Wall Street banks, 'regulators' (I love it) and intelligence agencies to 'prevent' (arrange and allow) a cyber-attack that would bring down the global financial system as long planned by those that control the WEF and the Carnegie operation. The Carnegie Endowment for International Peace sent an instruction to First World

War US President Woodrow Wilson not to let the war end before society had been irreversibly transformed.

The Wuhan lab diversion

As I close, the Cult-controlled authorities and lapdog media are systematically pushing 'the virus was released from the Wuhan lab' narrative. There are two versions – it happened by accident and it happened on purpose. Both are nonsense. The perceived existence of the never-shown-to-exist 'virus' is vital to sell the impression that there is actually an infective agent to deal with and to allow the endless potential for terrifying the population with 'variants' of a 'virus' that does not exist. The authorities at the time of writing are going with the 'by accident' while the alternative media is promoting the 'on purpose'. Cable news host Tucker Carlson who has questioned aspects of lockdown and 'vaccine' compulsion has bought the Wuhan lab story. 'Everyone now agrees' he said. Well, I don't and many others don't and the question is *why* does the system and its media suddenly 'agree'? When the media moves as one unit with a narrative it is always a lie – witness the hour by hour mendacity of the 'Covid' era. Why would this Cult-owned combination which has unleashed lies like machine gun fire suddenly 'agree' to tell the truth??

Much of the alternative media is buying the lie because it fits the conspiracy narrative, but it's the *wrong* conspiracy. The real conspiracy is that *there is no virus* and that is what the Cult is desperate to hide. The idea that the 'virus' was released by accident is ludicrous when the whole 'Covid' hoax was clearly long-planned and waiting to be played out as it was so fast in accordance with the Rockefeller document and Event 201. So they prepared everything in detail over decades and then sat around strumming their fingers waiting for an 'accidental' release from a bio-lab? *What??* It's crazy. Then there's the 'on purpose' claim. You want to circulate a 'deadly virus' and hide the fact that you've done so and you release it down the street from the highest-level bio-lab in China? I repeat – *What??*

You would release it far from that lab to stop any association being made. But, no, we'll do it in a place where the connection was certain to be made. Why would you need to scam 'cases' and 'deaths' and pay hospitals to diagnose 'Covid-19' if you had a real 'virus'? What are sections of the alternative media doing believing this crap? Where were all the mass deaths in Wuhan from a 'deadly pathogen' when the recovery to normal life after the initial propaganda was dramatic in speed? Why isn't the 'deadly pathogen' now circulating all over China with bodies in the street? Once again we have the technique of tell them what they want to hear and they will likely believe it. The alternative media has its 'conspiracy' and with Carlson it fits with his 'China is the danger' narrative over years. China *is* a danger as a global Cult operations centre, but not for this reason. The Wuhan lab story also has the potential to instigate conflict with China when at some stage the plan is to trigger a Problem-Reaction-Solution confrontation with the West. Question everything – *everything* – and especially when the media agrees on a common party line.

Third wave ... fourth wave ... fifth wave ...

As the book went into production the world was being set up for more lockdowns and a 'third wave' supported by invented 'variants' that were increasing all the time and will continue to do so in public statements and computer programs, but not in reality. India became the new Italy in the 'Covid' propaganda campaign and we were told to be frightened of the new 'Indian strain'. Somehow I couldn't find it within myself to do so. A document produced for the UK government entitled 'Summary of further modelling of easing of restrictions – Roadmap Step 2' declared that a third wave was inevitable (of course when it's in the script) and it would be the fault of children and those who refuse the health-destroying fake 'Covid vaccine'. One of the computer models involved came from the Cult-owned *Imperial College* and the other from Warwick University which I wouldn't trust to tell me the date in a calendar factory. The document states that both models presumed extremely high uptake

of the 'Covid vaccines' and didn't allow for 'variants'. The document states: 'The resurgence is a result of some people (mostly children) being ineligible for vaccination; others choosing not to receive the vaccine; and others being vaccinated but not perfectly protected.' The mendacity takes the breath away. Okay, blame those with a brain who won't take the DNA-modifying shots and put more pressure on children to have it as 'trials' were underway involving children as young as six months with parents who give insanity a bad name. Massive pressure is being put on the young to have the fake 'vaccine' and child age consent limits have been systematically lowered around the world to stop parents intervening. Most extraordinary about the document was its claim that the 'third wave' would be driven by 'the resurgence in both hospitalisations and deaths … dominated by *those that have received two doses of the vaccine*, comprising around 60-70% of the wave respectively'. The predicted peak of the 'third wave' suggested 300 deaths per day with 250 of them *fully 'vaccinated' people*. How many more lies do acquiescers need to be told before they see the obvious? Those who took the jab to 'protect themselves' are projected to be those who mostly get sick and die? So what's in the 'vaccine'? The document went on:

> It is possible that a summer of low prevalence could be followed by substantial increases in incidence over the following autumn and winter. Low prevalence in late summer should not be taken as an indication that SARS-CoV-2 has retreated or that the population has high enough levels of immunity to prevent another wave.

They are telling you the script and while many British people believed 'Covid' restrictions would end in the summer of 2021 the government was preparing for them to be ongoing. Authorities were awarding contracts for 'Covid marshals' to police the restrictions with contracts starting in July, 2021, and going through to January 31st, 2022, and the government was advertising for 'Media Buying Services' to secure media propaganda slots worth a potential £320 million for 'Covid-19 campaigns' with a contract not ending until March, 2022. The recipient – via a list of other front companies – was reported to be American media marketing giant Omnicom Group

Inc. While money is no object for 'Covid' the UK waiting list for all other treatment – including life-threatening conditions – passed 4.5 million. Meantime the Cult is seeking to control all official 'inquiries' to block revelations about what has really been happening and why. It must not be allowed to – we need Nuremberg jury trials in every country. The cover-up doesn't get more obvious than appointing ultra-Zionist professor Philip Zelikow to oversee two dozen US virologists, public health officials, clinicians, former government officials and four American 'charitable foundations' to 'learn the lessons' of the 'Covid' debacle. The personnel will be those that created and perpetuated the 'Covid' lies while Zelikow is the former executive director of the 9/11 Commission who ensured that the truth about those attacks never came out and produced a report that must be among the most mendacious and manipulative documents ever written – see *The Trigger* for the detailed exposure of the almost unimaginable 9/11 story in which Sabbatians can be found at every level.

Passive no more

People are increasingly challenging the authorities with amazing numbers of people taking to the streets in London well beyond the ability of the Face-Nappies to stop them. Instead the Nappies choose situations away from the mass crowds to target, intimidate, and seek to promote the impression of 'violent protestors'. One such incident happened in London's Hyde Park. Hundreds of thousands walking through the streets in protest against 'Covid' fascism were ignored by the Cult-owned BBC and most of the rest of the mainstream media, but they delighted in reporting how police were injured in 'clashes with protestors'. The truth was that a group of people gathered in Hyde Park at the end of one march when most had gone home and they were peacefully having a good time with music and chat. Face-Nappies who couldn't deal with the full-march crowd then waded in with their batons and got more than they bargained for. Instead of just standing for this criminal brutality the crowd used their numerical superiority to push the Face-Nappies out of the

park. Eventually the Nappies turned and ran. Unfortunately two or three idiots in the crowd threw drink cans striking two officers which gave the media and the government the image they wanted to discredit the 99.9999 percent who were peaceful. The idiots walked straight into the trap and we must always be aware of potential agent provocateurs used by the authorities to discredit their targets.

This response from the crowd – the can people apart – must be a turning point when the public no longer stand by while the innocent are arrested and brutally attacked by the Face-Nappies. That doesn't mean to be violent, that's the last thing we need. We'll leave the violence to the Face-Nappies and government. But it does mean that when the Face-Nappies use violence against peaceful people the numerical superiority is employed to stop them and make citizen's arrests or Common Law arrests for a breach of the peace. The time for being passive in the face of fascism is over.

We are the many, they are the few, and we need to make that count before there is no freedom left and our children and grandchildren face an ongoing fascist nightmare.

COME ON PEOPLE – IT'S TIME.

One final thought …

The power of love
A force from above
Cleaning my soul
Flame on burn desire
Love with tongues of fire
Purge the soul
Make love your goal

I'll protect you from the hooded claw
Keep the vampires from your door
When the chips are down I'll be around
With my undying, death-defying
Love for you

Envy will hurt itself
Let yourself be beautiful
Sparkling love, flowers
And pearls and pretty girls
Love is like an energy
Rushin' rushin' inside of me

This time we go sublime
Lovers entwine, divine, divine,
Love is danger, love is pleasure
Love is pure – the only treasure

I'm so in love with you
Purge the soul
Make love your goal

The power of love
A force from above
Cleaning my soul
The power of love
A force from above
A sky-scraping dove

Flame on burn desire
Love with tongues of fire
Purge the soul
Make love your goal

Frankie Goes To Hollywood

APPENDIX

Cowan-Kaufman-Morell Statement on Virus Isolation (SOVI)

Isolation: The action of isolating; the fact or condition of being isolated or standing alone; separation from other things or persons; solitariness
Oxford English Dictionary

The controversy over whether the SARS-CoV-2 virus has ever been isolated or purified continues. However, using the above definition, common sense, the laws of logic and the dictates of science, any unbiased person must come to the conclusion that the SARS-CoV-2 virus has never been isolated or purified. As a result, no confirmation of the virus' existence can be found. The logical, common sense, and scientific consequences of this fact are:

- the structure and composition of something not shown to exist can't be known, including the presence, structure, and function of any hypothetical spike or other proteins;
- the genetic sequence of something that has never been found can't be known;
- "variants" of something that hasn't been shown to exist can't be known;
- it's impossible to demonstrate that SARS-CoV-2 causes a disease called Covid-19.

In as concise terms as possible, here's the proper way to isolate, characterize and demonstrate a new virus. First, one takes samples (blood, sputum, secretions) from many people (e.g. 500) with symptoms which are unique and specific enough to characterize an illness. Without mixing these samples with ANY tissue or products that also contain genetic material, the virologist macerates, filters and ultracentrifuges i.e. *purifies* the specimen. This common virology technique, done for decades to isolate bacteriophages[1] and so-called giant viruses in every virology lab, then allows the virologist to demonstrate with electron microscopy thousands of identically sized and shaped particles. These particles are the isolated and purified virus.

These identical particles are then checked for uniformity by physical and/or microscopic techniques. Once the purity is determined, the particles may be further characterized. This would include examining the structure, morphology, and chemical composition of the particles. Next, their genetic makeup is characterized by extracting the genetic material directly from the purified particles and using genetic-sequencing techniques, such as Sanger sequencing, that have also been around for decades. Then one does an analysis to confirm that these uniform particles are exogenous (outside) in origin as a virus is conceptualized to be, and not the normal breakdown products of dead and dying tissues.[2] (As of May 2020, we know that virologists have no way to determine whether the particles they're seeing are viruses or just normal breakdown products of dead and dying tissues.)[3]

---

1 Isolation, characterization and analysis of bacteriophages from the haloalkaline lake Elmenteita, KenyaJuliah Khayeli Akhwale et al, PLOS One, Published: April 25, 2019. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0215734 – accessed 2/15/21

2 "Extracellular Vesicles Derived From Apoptotic Cells: An Essential Link Between Death and Regeneration," Maojiao Li1 et al, Frontiers in Cell and Developmental Biology, 2020 October 2. https://www.frontiersin.org/articles/10.3389/fcell.2020.573511/full – accessed 2/15/21

3 "The Role of Extraellular Vesicles as Allies of HIV, HCV and SARS Viruses," Flavia Giannessi, et al, Viruses, 2020 May

If we have come this far then we have fully isolated, characterized, and genetically sequenced an exogenous virus particle. However, we still have to show it is causally related to a disease. This is carried out by exposing a group of healthy subjects (animals are usually used) to this isolated, purified virus in the manner in which the disease is thought to be transmitted. If the animals get sick with the same disease, as confirmed by clinical and autopsy findings, one has now shown that the virus actually causes a disease. This demonstrates infectivity and transmission of an infectious agent.

None of these steps has even been attempted with the SARS-CoV-2 virus, nor have all these steps been successfully performed for any so-called pathogenic virus. Our research indicates that a single study showing these steps does not exist in the medical literature.

Instead, since 1954, virologists have taken unpurified samples from a relatively few people, often less than ten, with a similar disease. They then minimally process this sample and inoculate this unpurified sample onto tissue culture containing usually four to six other types of material – all of which contain identical genetic material as to what is called a "virus." The tissue culture is starved and poisoned and naturally disintegrates into many types of particles, some of which contain genetic material. Against all common sense, logic, use of the English language and scientific integrity, this process is called "virus isolation." This brew containing fragments of genetic material from many sources is then subjected to genetic analysis, which then creates in a computer-simulation process the alleged sequence of the alleged virus, a so called in silico genome. At no time is an actual virus confirmed by electron microscopy. At no time is a genome extracted and sequenced from an actual virus. This is scientific fraud.

The observation that the unpurified specimen — inoculated onto tissue culture along with toxic antibiotics, bovine fetal tissue, amniotic fluid and other tissues — destroys the kidney tissue onto which it is inoculated is given as evidence of the virus' existence and pathogenicity. This is scientific fraud.

From now on, when anyone gives you a paper that suggests the SARS-CoV-2 virus has been isolated, please check the methods sections. If the researchers used Vero cells or any other culture method, you know that their process was not isolation. You will hear the following excuses for why actual isolation isn't done:

1. There were not enough virus particles found in samples from patients to analyze.

2. Viruses are intracellular parasites; they can't be found outside the cell in this manner.

If No. 1 is correct, and we can't find the virus in the sputum of sick people, then on what evidence do we think the virus is dangerous or even lethal? If No. 2 is correct, then how is the virus spread from person to person? We are told it emerges from the cell to infect others. Then why isn't it possible to find it?

Finally, questioning these virology techniques and conclusions is not some distraction or divisive issue. Shining the light on this truth is essential to stop this terrible fraud that humanity is confronting. For, as we now know, if the virus has never been isolated, sequenced or shown to cause illness, if the virus is imaginary, then why are we wearing masks, social distancing and putting the whole world into prison?

Finally, if pathogenic viruses don't exist, then what is going into those injectable devices erroneously called "vaccines," and what is their purpose? This scientific question is the most urgent and relevant one of our time.

We are correct. The SARS-CoV2 virus does not exist.

Sally Fallon Morell, MA
Dr. Thomas Cowan, MD
Dr. Andrew Kaufman, MD

Bibliography

Alinsky, Saul: *Rules for Radicals* (Vintage, 1989)

Antelman, Rabbi Marvin: *To Eliminate the Opiate* (Zahavia, 1974)

Bastardi, Joe: *The Climate Chronicles* (Relentless Thunder Press, 2018)

Cowan, Tom: *Human Heart, Cosmic Heart* (Chelsea Green Publishing, 2016)

Cowan, Tom, and Fallon Morell, Sally: *The Contagion Myth* (Skyhorse Publishing, 2020)

Forbes, Jack D: *Columbus And Other Cannibals – The Wetiko Disease of Exploitation, Imperialism, and Terrorism* (Seven Stories Press, 2008 – originally published in 1979)

Gates, Bill: *How to Avoid a Climate Disaster: The Solutions We Have and the Breakthroughs We Need* (Allen Lane, 2021)

Huxley, Aldous: *Brave New World* (Chatto & Windus, 1932)

Köhnlein, Dr Claus, and Engelbrecht, Torsten: *Virus Mania* (emu-Vertag, Lahnstein, 2020)

Lanza, Robert, and Berman, Bob: *Biocentrism* (BenBella Books, 2010)

Lash, John Lamb: *Not In His Image* (Chelsea Green Publishing, 2006)

Lester, Dawn, and Parker, David: *What Really Makes You Ill – Why everything you thought you knew about disease is wrong* (Independently Published, 2019)

Levy, Paul: *Dispelling Wetiko, Breaking the Spell of Evil* (North Atlantic Books, 2013)

Marx, Karl: *A World Without Jews* (Philosophical Library, first edition, 1959)

Mullis, Kary: *Dancing Naked in the Mine Field* (Bloomsbury, 1999)

O'Brien, Cathy: *Trance-Formation of America* (Reality Marketing, 1995)

Scholem, Gershon: *The Messianic Idea in Judaism* (Schocken Books, 1994)

Schwab, Klaus, and Davis, Nicholas: *Shaping the Future of the Fourth Industrial Revolution: A guide to building a better world* (Penguin Books, 2018)

Schwab, Klaus: *The Great Reset* (Agentur Schweiz, 2020)

Sunstein, Cass and Thaler, Richard: *Nudge: Improving Decisions About Health, Wealth, and Happiness* (Penguin, 2009)

Swan, Shanna: *Count Down: How Our Modern World Is Threatening Sperm Counts, Altering Male and Female Reproductive Development and Imperiling the Future of the Human Race* (Scribner, 2021)

Tegmark, Max: *Our Mathematical Universe: My Quest for the Ultimate Nature of Reality* (Penguin, 2015)

Velikovsky, Immanuel: *Worlds in Collision* (Paradigma, 2009)

Wilton, Robert: *The Last Days of the Romanovs* (Blurb, 2018, first published 1920)

Index

A

abusive relationships
blaming themselves, abused as ref1
children ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
conspiracy theories ref1
domestic abuse ref1, ref2
economic abuse and dependency ref1
isolation ref1
physical abuse ref1
psychological abuse ref1
signs of abuse ref1
addiction
alcoholism ref1
frequencies ref1
substance abuse ref1, ref2
technology ref1, ref2, ref3
Adelson, Sheldon ref1, ref2, ref3
Agenda 21/Agenda 2030 (UN) ref1, ref2, ref3, ref4
AIDs/HIV ref1
causal link between HIV and AIDs ref1, ref2
retroviruses ref1
testing ref1, ref2
trial-run for Covid-19, as ref1, ref2
aliens/extraterrestrials ref1, ref2
aluminium ref1
Amazon ref1, ref2, ref3

amplification cycles ref1, ref2
anaphylactic shock ref1, ref2, ref3, ref4
animals ref1, ref2, ref3
antibodies ref1, ref2, ref3, ref4, ref5
Antifa ref1, ref2, ref3, ref4
antigens ref1, ref2
anti-Semitism ref1, ref2, ref3
Archons ref1, ref2
  consciousness ref1, ref2, ref3
  energy ref1, ref2, ref3
  ennoia ref1
  genetic manipulation ref1, ref2
  inversion ref1, ref2, ref3
  lockdowns ref1
  money ref1
  radiation ref1
  religion ref1, ref2
  technology ref1, ref2, ref3
  Wetiko factor ref1, ref2, ref3, ref4
artificial intelligence (AI) ref1
army made up of robots ref1, ref2
  Human 2.0 ref1, ref2
  Internet ref1
  MHRA ref1
  Morgellons fibres ref1, ref2
  Smart Grid ref1
  Wetiko factor ref1
asymptomatic, Covid-19 as ref1, ref2, ref3
aviation industry ref1

B

banking, finance and money ref1, ref2, ref3
2008 crisis ref1, ref2
boom and bust ref1
cashless digital money systems ref1
central banks ref1
credit ref1
digital currency ref1
fractional reserve lending ref1
Great Reset ref1
guaranteed income ref1, ref2, ref3
Human 2.0 ref1
incomes, destruction of ref1, ref2
interest ref1
one per cent ref1, ref2
scams ref1
BBC ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
Becker-Phelps, Leslie ref1
Behavioural Insights Team (BIT) (Nudge Unit) ref1, ref2, ref3
behavioural scientists *and* **psychologists, advice from** ref1, ref2
Bezos, Jeff ref1, ref2, ref3, ref4
Biden, Hunter ref1
Biden, Joe ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11, ref12, ref13, ref14, ref15, ref16, ref17
Big Pharma
cholesterol ref1
health professionals ref1, ref2
immunity from prosecution in US ref1
vaccines ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
Wetiko factor ref1, ref2
WHO ref1, ref2, ref3
Bill and Melinda Gates Foundation ref1, ref2, ref3, ref4, ref5, ref6, ref7

billionaires ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9 ref10, ref11
bird flu (H5N1) ref1
Black Lives Matter (BLM) ref1, ref2, ref3, ref4, ref5
Blair, Tony ref1, ref2, ref3, ref4, ref5, ref6, ref7
Brin, Sergei ref1, ref2, ref3, ref4, ref5, ref6, ref7
British Empire ref1
Bush, George HW ref1, ref2
Bush, George W ref1, ref2, ref3, ref4
Byrd, Robert ref1

C

Canada
  Global Cult ref1
  hate speech ref1
  internment ref1
  masks ref1
  old people ref1
  SARS-COV-2 ref1
  satellites ref1
  vaccines ref1
  wearable technology ref1
Capitol Hill riot ref1, ref2
  agents provocateur ref1
  Antifa ref1
  Black Lives Matter (BLM) ref1, ref2
  QAnon ref1
  security precautions, lack of ref1, ref2, ref3
carbon dioxide ref1, ref2
care homes, deaths in ref1, ref2
cashless digital money systems ref1
censorship ref1, ref2, ref3, ref4, ref5

fact-checkers ref1
masks ref1
media ref1, ref2
private messages ref1
social media ref1, ref2, ref3, ref4, ref5, ref6
transgender persons ref1
vaccines ref1, ref2, ref3
Wokeness ref1
Centers for Disease Control (CDC) (United States) ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11, ref12, ref13
centralisation ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
chakras ref1
change agents ref1, ref2, ref3
chemtrails ref1, ref2, ref3
chief medical officers and scientific advisers ref1, ref2, ref3, ref4, ref5, ref6
children *see also* **young people**
abuse ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
care, taken into ref1, ref2, ref3
education ref1, ref2, ref3, ref4
energy ref1
family courts ref1
hand sanitisers ref1
human sacrifice ref1
lockdowns ref1, ref2, ref3
masks ref1, ref2, ref3, ref4, ref5
mental health ref1
old people ref1
parents, replacement of ref1, ref2
Psyop (psychological operation), Covid as a ref1, ref2
reframing ref1
smartphone addiction ref1

social distancing and isolation ref1
social media ref1
transgender persons ref1, ref2
United States ref1
vaccines ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
Wetiko factor ref1
China ref1, ref2, ref3, ref4
anal swab tests ref1
Chinese Revolution ref1, ref2, ref3
digital currency ref1
Global Cult ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
guaranteed income ref1
Imperial College ref1
Israel ref1
lockdown ref1, ref2
masculinity crisis ref1
masks ref1
media ref1
origins of virus in China ref1, ref2, ref3, ref4, ref5
pollution causing respiratory diseases ref1
Sabbatians ref1, ref2
Smart Grid ref1, ref2
social credit system ref1
testing ref1, ref2
United States ref1, ref2
vaccines ref1, ref2
Wetiko factor ref1
wet market conspiracy ref1
Wuhan ref1, ref2, ref3, ref4, ref5, ref6, ref7
cholesterol ref1, ref2
Christianity ref1, ref2, ref3, ref4, ref5
criticism ref1
cross, inversion of the ref1

Nag Hammadi texts ref1, ref2, ref3
Roman Catholic Church ref1, ref2
Sabbatians ref1, ref2
Satan ref1, ref2, ref3, ref4
Wokeness ref1
class ref1, ref2
climate change hoax ref1, ref2, ref3, ref4, ref5
Agenda 21/Agenda 2030 ref1, ref2, ref3
carbon dioxide ref1, ref2
Club of Rome ref1, ref2, ref3, ref4, ref5
fear ref1
funding ref1
Global Cult ref1
green new deals ref1
green parties ref1
inversion ref1
perception, control of ref1
PICC ref1
reframing ref1
temperature, increases in ref1
United Nations ref1, ref2
Wikipedia ref1
Wokeness ref1, ref2
Clinton, Bill ref1, ref2, ref3, ref4, ref5, ref6
Clinton, Hillary ref1, ref2, ref3
the cloud ref1, ref2, ref3, ref4, ref5, ref6, ref7
Club of Rome and climate change hoax ref1, ref2, ref3, ref4, ref5
cognitive therapy ref1
Cohn, Roy ref1
Common Law ref1
Admiralty Law ref1
arrests ref1, ref2

contractual law, Statute Law as ref1
corporate entities, people as ref1
legalese ref1
sea, law of the ref1
Statute Law ref1
Common Purpose leadership programme ref1, ref2
communism ref1, ref2
co-morbidities ref1
computer-generated virus,
Covid-19 as ref1, ref2, ref3
computer models ref1, ref2, ref3, ref4, ref5
connections ref1, ref2, ref3, ref4
consciousness ref1, ref2, ref3, ref4
Archons ref1, ref2, ref3
expanded ref1, ref2, ref3, ref4, ref5, ref6, ref7
experience ref1
heart ref1
infinity ref1, ref2
religion ref1, ref2
self-identity ref1
simulation thesis ref1
vaccines ref1
Wetiko factor ref1, ref2
conspiracy theorists ref1, ref2, ref3, ref4, ref5
contradictory rules ref1
contrails ref1
Corman-Drosten test ref1, ref2, ref3, ref4
countermimicry ref1, ref2, ref3
Covid-19 vaccines *see* vaccines
Covidiots ref1, ref2
Cowan, Tom ref1, ref2, ref3, ref4
crimes against humanity ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8

cyber-operations ref1
cyberwarfare ref1

D

DARPA (Defense Advanced Research Projects Agency) ref1
deaths
  care homes ref1
  certificates ref1, ref2, ref3, ref4
  mortality rate ref1
  post-mortems/autopsies ref1
  recording ref1, ref2, ref3, ref4, ref5, ref6, ref7
  vaccines ref1, ref2, ref3, ref4, ref5
deceit
  pyramid of deceit ref1, ref2
  sequence of deceit ref1
decoding ref1, ref2, ref3
dehumanisation ref1, ref2, ref3
Delphi technique ref1
democracy ref1
dependency ref1, ref2, ref3, ref4, ref5
Descartes, René ref1
DNA
  numbers ref1
  vaccines ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
DNR (do not resuscitate)
  orders ref1
domestic abuse ref1, ref2
downgrading of Covid-19 ref1
Drosten, Christian ref1, ref2, ref3, ref4, ref5, ref6, ref7
Duesberg, Peter ref1, ref2

E

economic abuse ref1
Edmunds, John ref1, ref2
education ref1, ref2, ref3, ref4
electromagnetic spectrum ref1, ref2
Enders, John ref1
energy
  Archons ref1, ref2, ref3
  children and young people ref1
  consciousness ref1
  decoding ref1
  frequencies ref1, ref2, ref3, ref4
  heart ref1
  human energy field ref1
  source, humans as an energy ref1, ref2
  vaccines ref1
  viruses ref1
ennoia ref1
Epstein, Jeffrey ref1, ref2
eternal 'I' ref1, ref2
ethylene oxide ref1
European Union ref1, ref2, ref3, ref4
Event ref1 *and* **Bill Gates** ref2
exosomes, Covid-19 as natural defence mechanism called ref1
experience ref1, ref2
Extinction Rebellion ref1, ref2

F

Facebook
  addiction ref1, 448–50
  Facebook

Archons ref1
censorship ref1, ref2, ref3
hate speech ref1
monopoly, as ref1
private messages, censorship of ref1
Sabbatians ref1
United States election fraud ref1
vaccines ref1
Wetiko factor ref1
fact-checkers ref1
Fauci, Anthony ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11, ref12
fear ref1, ref2, ref3, ref4
climate change ref1
computer models ref1
conspiracy theories ref1
empty hospitals ref1
Italy ref1, ref2, ref3
lockdowns ref1, ref2, ref3, ref4
masks ref1, ref2
media ref1, ref2
medical staff ref1
Psyop (psychological operation), Covid as a ref1
Wetiko factor ref1, ref2
female infertility ref1
Fermi Paradox ref1
Ferguson, Neil ref1, ref2, ref3, ref4, ref5, ref6, ref7
fertility, decline in ref1
The Field ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
finance *see* **banking, finance and money**
five-senses ref1, ref2
Archons ref1, ref2, ref3

censorship ref1
consciousness, expansion of ref1, ref2, ref3, ref4, ref5, ref6
decoding ref1
education ref1, ref2
the Field ref1, ref2
God, personification of ref1
infinity ref1, ref2
media ref1
paranormal ref1
perceptual programming ref1, ref2
Phantom Self ref1
pneuma not nous, using ref1
reincarnation ref1
self-identity ref1
Wetiko factor ref1, ref2, ref3, ref4, ref5, ref6
5G ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
Floyd, George and protests, killing of ref1
flu, re-labelling of ref1, ref2, ref3
food and water, control of ref1, ref2
Freemasons ref1, ref2, ref3, ref4, ref5, ref6
Frei, Rosemary ref1
frequencies
addictions ref1
Archons ref1, ref2, ref3
awareness ref1
chanting and mantras ref1
consciousness ref1
decoding ref1, ref2
education ref1
electromagnetic (EMF) frequencies ref1
energy ref1, ref2, ref3, ref4
fear ref1

the Field ref1, ref2 5G ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
five-senses ref1, ref2
ghosts ref1
Gnostics ref1
hive-minds ref1
human, meaning of ref1
light ref1, ref2
love ref1, ref2
magnetism ref1
perception ref1
reality ref1, ref2, ref3
simulation ref1
terror ref1
vaccines ref1
Wetiko ref1, ref2, ref3
Fuellmich, Reiner ref1, ref2, ref3
furlough/rescue payments ref1

G

Gallo, Robert ref1, ref2, ref3
Gates, Bill
Archons ref1, ref2, ref3
climate change ref1, ref2, ref3, ref4
Daily Pass tracking system ref1
Epstein ref1
fascism ref1
five senses ref1
GAVI ref1
Great Reset ref1
GSK ref1
Imperial College ref1, ref2
Johns Hopkins University ref1, ref2, ref3

lockdowns ref1, ref2
masks ref1
Nuremberg trial, proposal for ref1, ref2
Rockefellers ref1, ref2
social distancing and isolation ref1
Sun, dimming the ref1
synthetic meat ref1, ref2
vaccines ref1, ref2, ref3, ref4, ref5, ref6, ref7
Wellcome Trust ref1
Wetiko factor ref1, ref2, ref3
WHO ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
Wokeness ref1
World Economic Forum ref1, ref2, ref3, ref4
Gates, Melinda ref1, ref2, ref3
GAVI vaccine alliance ref1
genetics, manipulation of ref1, ref2, ref3
Germany ref1, ref2, ref3, ref4, ref5, ref6 *see also* **Nazi Germany**
Global Cult ref1, ref2, ref3, ref4, ref5
anti-human, why Global Cult is ref1
Black Lives Matter (BLM) ref1, ref2, ref3, ref4
China ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
climate change hoax ref1
contradictory rules ref1
Covid-19 ref1, ref2, ref3
fascism ref1
geographical origins ref1
immigration ref1
Internet ref1
mainstream media ref1, ref2
masks ref1, ref2
monarchy ref1
non-human dimension ref1

perception ref1
political parties ref1, ref2
pyramidal hierarchy ref1, ref2, ref3
reframing ref1
Sabbantian-Frankism ref1, ref2
science, manipulation of ref1
spider and the web ref1
transgender persons ref1
vaccines ref1
who controls the Cult ref1
Wokeness ref1, ref2, ref3, ref4
globalisation ref1, ref2
Gnostics ref1, ref2, ref3, ref4, ref5
Google ref1, ref2, ref3, ref4
government
behavioural scientists and psychologists, advice from ref1, ref2
definition ref1
Joint Biosecurity Centre (JBC) ref1
people, abusive relationship with ref1
Great Reset ref1, ref2, ref3, ref4, ref5, ref6
fascism ref1, ref2, ref3
financial system ref1
Human 2.0 ref1
water and food, control of ref1
green parties ref1
Griesz-Brisson, Margarite ref1
guaranteed income ref1, ref2, ref3

H

Hancock, Matt ref1, ref2, ref3, ref4, ref5
hand sanitisers ref1
heart ref1, ref2

hive-minds/groupthink ref1, ref2, ref3
holographs ref1, ref2, ref3, ref4
hospitals, empty ref1
human, meaning of ref1
Human 2.0 ref1
addiction to technology ref1
artificial intelligence (AI) ref1, ref2
elimination of Human 1.0 ref1
fertility, decline in ref1
Great Reset ref1
implantables ref1
money ref1
mRNA ref1
nanotechnology ref1
parents, replacement of ref1, ref2
Smart Grid, connection to ref1, ref2
synthetic biology ref1, ref2, ref3, ref4
testosterone levels, decrease in ref1
transgender = transhumanism ref1, ref2, ref3
vaccines ref1, ref2, ref3, ref4
human sacrifice ref1, ref2, ref3
Hunger Games Society ref1, ref2, ref3, ref4, ref5, ref6, ref7
Huxley, Aldous ref1, ref2, ref3

I

identity politics ref1, ref2, ref3
Illuminati ref1, ref2
illusory physical reality ref1
immigration ref1, ref2, ref3, ref4
Imperial College ref1, ref2, ref3, ref4, ref5, ref6
implantables ref1, ref2

incomes, destruction of ref1, ref2
Infinite Awareness ref1, ref2, ref3, ref4
Internet ref1, ref2 *see also* social media
artificial intelligence (AI) ref1
independent journalism, lack of ref1
Internet of Bodies (IoB) ref1
Internet of Everything (IoE) ref1, ref2
Internet of Things (IoT) ref1, ref2
lockdowns ref1
Psyop (psychological operation), Covid as a ref1
trolls ref1
intersectionality ref1
inversion
Archons ref1, ref2, ref3
climate change hoax ref1
energy ref1
Judaism ref1, ref2, ref3
symbolism ref1
Wetiko factor ref1
Wokeness ref1, ref2, ref3
Islam
Archons ref1
crypto-Jews ref1
Islamic State ref1, ref2
Jinn and Djinn ref1, ref2, ref3
Ottoman Empire ref1
Wahhabism ref1
isolation *see* **social distancing** *and* **isolation**
Israel
China ref1
Cyber Intelligence Unit Beersheba complex ref1
expansion of illegal settlements ref1

formation ref1
Global Cult ref1
Judaism ref1, ref2, ref3, ref4, ref5
medical experiments, consent for ref1
Mossad ref1, ref2, ref3, ref4
Palestine-Israel conflict ref1, ref2, ref3
parents, replacement of ref1
Sabbatians ref1, ref2, ref3, ref4, ref5
September 11, 2001, terrorist attacks on United States ref1
Silicon Valley ref1
Smart Grid ref1, ref2
United States ref1, ref2
vaccines ref1
Wetiko factor ref1

Italy
fear ref1, ref2, ref3
Lombardy ref1, ref2, ref3
vaccines ref1

J

Johns Hopkins University ref1, ref2, ref3, ref4, ref5, ref6, ref7
Johnson, Boris ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
Joint Biosecurity Centre (JBC) ref1
Judaism
anti-Semitism ref1, ref2, ref3
Archons ref1, ref2
crypto-Jews ref1
inversion ref1, ref2, ref3
Israel ref1, ref2, ref3, ref4, ref5
Labour Party ref1
Nazi Germany ref1, ref2, ref3, ref4
Sabbatians ref1, ref2, ref3, ref4, ref5

Silicon Valley ref1
Torah ref1
United States ref1, ref2
Zionists ref1, ref2, ref3

K

Kaufman, Andrew ref1, ref2, ref3, ref4
knowledge ref1, ref2, ref3, ref4, ref5, ref6
Koch's postulates ref1
Kurzweil, Ray ref1, ref2, ref3, ref4, ref5, ref6, ref7
Kushner, Jared **ref1**, **ref2**

L

Labour Party ref1, ref2
Lanka, Stefan ref1, ref2
Lateral Flow Device (LFD) ref1
Levy, Paul ref1, ref2, ref3
Life Program ref1
lockdowns ref1, ref2, ref3
amplification tampering ref1
Archons ref1
Behavioural Insights Team ref1
Black Lives Matter (BLM) ref1
care homes, deaths in ref1
children
abuse ref1, ref2
mental health ref1
China ref1, ref2
computer models ref1
consequences ref1, ref2
dependency ref1, ref2, ref3

domestic abuse ref1
fall in cases ref1
fear ref1, ref2, ref3, ref4
guaranteed income ref1
Hunger Games Society ref1, ref2, ref3
interaction, destroying ref1
Internet ref1, ref2
overdoses ref1
perception ref1
police-military state ref1, ref2
protests ref1, ref2, ref3, ref4, ref5
psychopathic personality ref1, ref2, ref3
reporting/snitching, encouragement of ref1, ref2
testing ref1
vaccines ref1
Wetiko factor ref1
WHO ref1
love ref1, ref2, ref3
Lucifer ref1, ref2, ref3

M

Madej, Carrie ref1, ref2
Magufuli, John ref1, ref2
mainstream media ref1
BBC ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
censorship ref1, ref2
China ref1
climate change hoax ref1
fear ref1, ref2
Global Cult ref1, ref2
independent journalism, lack of ref1
Ofcom ref1, ref2, ref3

perception ref1, ref2
Psyop (psychological operation), Covid as a ref1
Sabbatians ref1, ref2
social disapproval ref1
social distancing and isolation ref1
United States ref1, ref2
vaccines ref1, ref2, ref3, ref4, ref5
Mao Zedong ref1, ref2, ref3
Marx and Marxism ref1, ref2, ref3, ref4, ref5, ref6
masculinity ref1
masks/face coverings ref1, ref2, ref3
censorship ref1
children ref1, ref2, ref3, ref4, ref5
China, made in ref1
dehumanisation ref1, ref2, ref3
fear ref1, ref2
flu ref1
health professionals ref1, ref2, ref3, ref4
isolation ref1
laughter ref1
mass non-cooperation ref1
microplastics, risk of ref1
mind control ref1
multiple masks ref1
oxygen deficiency ref1, ref2, ref3
police ref1, ref2, ref3, ref4, ref5
pollution, as cause of plastic ref1
Psyop (psychological operation), Covid as a ref1
reframing ref1, ref2
risk assessments, lack of ref1, ref2
self-respect ref1
surgeons ref1

United States ref1
vaccines ref1, ref2, ref3, ref4, ref5
Wetiko factor ref1
'worms' ref1
The Matrix movies ref1, ref2, ref3
measles ref1, ref2
media see mainstream media
Medicines and Healthcare products Regulatory Agency (MHRA) ref1, ref2, ref3, ref4
Mesopotamia ref1
messaging ref1
military-police state ref1, ref2, ref3
mind control ref1, ref2, ref3, ref4, ref5, ref6 *see also* **MKUltra**
MKUltra ref1, ref2, ref3
monarchy ref1
money *see* **banking, finance and money**
Montagnier, Luc ref1, ref2, ref3
Mooney, Bel ref1
Morgellons disease ref1, ref2
mortality rate ref1
Mullis, Kary ref1, ref2, ref3
Musk, Elon ref1

N

Nag Hammadi texts ref1, ref2, ref3
nanotechnology ref1, ref2, ref3
narcissism ref1
Nazi Germany ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
near-death experiences ref1, ref2
Neocons ref1, ref2, ref3

Neuro-Linguistic Programming (NLP) and the Delphi technique ref1
NHS (National Health Service)
amplification cycles ref1
Common Purpose ref1, ref2
mind control ref1
NHS England ref1
saving the NHS ref1, ref2
vaccines ref1, ref2, ref3, ref4, ref5
whistle-blowers ref1, ref2, ref3
No-Problem-Reaction-Solution ref1, ref2, ref3, ref4
non-human dimension of Global Cult ref1
nous ref1
numbers, reality as ref1
Nuremberg Codes ref1, ref2, ref3
Nuremberg-like tribunal, proposal for ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11, ref12

O

Obama, Barack ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
O'Brien, Cathy ref1, ref2, ref3, ref4
Ochel, Evita ref1
Ofcom ref1, ref2, ref3
old people ref1, ref2, ref3, ref4, ref5
Oneness ref1, ref2, ref3
Open Society Foundations (Soros) ref1, ref2, ref3
oxygen 406, 528–34

P

paedophilia ref1, ref2
Page, Larry ref1, ref2, ref3, ref4, ref5, ref6, ref7

Palestine-Israel conflict ref1, ref2, ref3
pandemic, definition of ref1
pandemic and health crisis scenarios/simulations ref1, ref2, ref3, ref4
paranormal ref1
PCR tests ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
Pearl Harbor attacks, prior knowledge of ref1
Pelosi, Nancy ref1, ref2, ref3
perception ref1, ref2, ref3, ref4
  climate change hoax ref1
  control ref1, ref2, ref3
  decoding ref1, ref2
  enslavement ref1
  externally-delivered perceptions ref1
  five senses ref1
  human labels ref1
  media ref1, ref2
  political parties ref1, ref2
  Psyop (psychological operation), Covid as a ref1
  sale of perception ref1
  self-identity ref1, ref2
  Wokeness ref1
Phantom Self ref1, ref2, ref3
pharmaceutical industry *see* **Big Pharma**
phthalates ref1
Plato's Allegory of the Cave ref1, ref2
pneuma ref1
police
  Black Lives Matter (BLM) ref1
  brutality ref1
  citizen's arrests ref1, ref2
  common law arrests ref1, ref2

Common Purpose ref1
defunding ref1
lockdowns ref1, ref2
masks ref1, ref2, ref3, ref4
police-military state ref1, ref2, ref3
psychopathic personality ref1, ref2, ref3, ref4
reframing ref1
United States ref1, ref2, ref3, ref4
Wokeness ref1

polio ref1

political correctness ref1, ref2, ref3, ref4

political parties ref1, ref2, ref3, ref4

political puppets ref1

pollution ref1, ref2, ref3

post-mortems/autopsies ref1

Postage Stamp Consensus ref1, ref2

pre-emptive programming ref1

Problem-Reaction-Solution ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8

Project for the New American Century ref1, ref2, ref3, ref4

psychopathic personality ref1
Archons ref1
heart energy ref1
lockdowns ref1, ref2, ref3
police ref1, ref2, ref3, ref4
recruitment ref1, ref2
vaccines ref1
wealth ref1
Wetiko ref1, ref2

Psyop (psychological operation), Covid as a ref1, ref2, ref3, ref4, ref5

Pushbackers ref1, ref2, ref3, ref4

pyramid structure ref1, ref2, ref3, ref4

Q

QAnon Psyop ref1, ref2, ref3

R

racism *see also* **Black Lives**
Matter (BLM)
anti-racism industry ref1
class ref1
critical race theory ref1
culture ref1
intersectionality ref1
reverse racism ref1
white privilege ref1, ref2
white supremacy ref1, ref2, ref3, ref4, ref5
Wokeness ref1, ref2, ref3
radiation ref1, ref2
randomness, illusion of ref1, ref2, ref3
reality ref1, ref2, ref3
reframing ref1, ref2
change agents ref1, ref2
children ref1
climate change ref1
Common Purpose leadership programme ref1, ref2
contradictory rules ref1
enforcers ref1
masks ref1, ref2
NLP and the Delphi technique ref1
police ref1
Wetiko factor ref1
Wokeness ref1, ref2
religion *see also* particular religions
alien invasions ref1

Archons ref1, ref2
consciousness ref1, ref2
control, system of ref1, ref2, ref3
criticism, prohibition on ref1
five senses ref1
good and evil, war between ref1
hidden non-human forces ref1, ref2
Sabbatians ref1
save me syndrome ref1
Wetiko ref1
Wokeness ref1
repetition and mind control ref1, ref2, ref3
reporting/snitching, encouragement of ref1, ref2
Reptilians/Grey entities ref1
rewiring the mind ref1
Rivers, Thomas Milton ref1, ref2
Rockefeller family ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
Rockefeller Foundation documents ref1, ref2, ref3, ref4
Roman Empire ref1
Rothschild family ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
RT-PCR tests ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
Russia
collusion inquiry in US ref1
Russian Revolution ref1, ref2
Sabbatians ref1

S

Sabbantian-Frankism ref1, ref2
anti-Semitism ref1, ref2
banking and finance ref1, ref2, ref3
China ref1, ref2
Israel ref1, ref2, ref3, ref4, ref5

Judaism ref1, ref2, ref3, ref4, ref5
Lucifer ref1
media ref1, ref2
Nazis ref1, ref2
QAnon ref1
Rothschilds ref1, ref2, ref3, ref4, ref5, ref6
Russia ref1
Saudi Arabia ref1
Silicon Valley ref1
Sumer ref1
United States ref1, ref2, ref3
Wetiko factor ref1
Wokeness ref1, ref2, ref3

SAGE (Scientific Advisory Group for Emergencies) ref1, ref2, ref3, ref4

SARS-1 ref1

SARs-CoV-2 ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8

Satan/Satanism ref1, ref2, ref3, ref4, ref5, ref6, ref7

satellites in low-orbit ref1

Saudi Arabia ref1

Save Me Syndrome ref1

scapegoating ref1

Schwab, Klaus ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11, ref12

science, manipulation of ref1

self-identity ref1, ref2, ref3, ref4

self-respect, attacks on ref1

September 11, 2001, terrorist attacks on United States ref1, ref2, ref3, ref4

77th Brigade of UK military ref1, ref2, ref3

Silicon Valley/tech giants ref1, ref2, ref3, ref4, ref5, ref6 *see also* **Facebook**

Israel ref1
Sabbatians ref1
technocracy ref1
Wetiko factor ref1
Wokeness ref1
simulation hypothesis ref1, ref2, ref3, ref4, ref5
Smart Grid ref1, ref2, ref3
artificial intelligence (AI) ref1
China ref1, ref2
control centres ref1
the Field ref1
Great Reset ref1
Human 2.0 ref1, ref2
Israel ref1, ref2
vaccines ref1
Wetiko factor ref1
social disapproval ref1
social distancing *and* **isolation** ref1, ref2, ref3
abusive relationships ref1, ref2
children ref1
flats and apartments ref1
heart issues ref1
hugs ref1
Internet ref1
masks ref1
media ref1
older people ref1, ref2
one-metre (three feet) rule ref1
rewiring the mind ref1
simulation, universe as a ref1
SPI-B ref1
substance abuse ref1

suicide and self-harm ref1, ref2, ref3, ref4, ref5
technology ref1
torture, as ref1, ref2
two-metre (six feet) rule ref1
women ref1
social justice ref1, ref2, ref3, ref4
social media *see also* **Facebook bans on alternative views** ref1
censorship ref1, ref2, ref3, ref4, ref5, ref6
children ref1
emotion ref1
perception ref1
private messages ref1
Twitter ref1, ref2, ref3, ref4, ref5, ref6, ref7
Wetiko factor ref1
YouTube ref1, ref2, ref3, ref4, ref5
Soros, George ref1, ref2, ref3, ref4, ref5, ref6
Spain ref1
SPI-B (Scientific Pandemic Insights Group on Behaviours) ref1, ref2, ref3, ref4
spider and the web ref1, ref2, ref3, ref4
Starmer, Keir ref1
Statute Law ref1
Steiner, Rudolf ref1, ref2, ref3
Stockholm syndrome ref1
streptomycin ref1
suicide and self-harm ref1, ref2, ref3, ref4, ref5
Sumer ref1, ref2
Sunstein, Cass ref1, ref2, ref3
swine flu (H1N1) ref1, ref2, ref3
synchronicity ref1
synthetic biology ref1, ref2, ref3, ref4
synthetic meat ref1, ref2

T

technology *see also* **artificial intelligence (AI); Internet;**
social media addiction ref1, ref2, ref3, ref4
Archons ref1, ref2
the cloud ref1, ref2, ref3, ref4, ref5, ref6, ref7
cyber-operations ref1
cyberwarfare ref1
radiation ref1, ref2
social distancing and isolation ref1
technocracy ref1
Tedros Adhanom Ghebreyesus ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11, ref12, ref13
telepathy ref1
Tenpenny, Sherri ref1
Tesla, Nikola ref1
testosterone levels, decrease in ref1
testing for Covid-19 ref1, ref2
anal swab tests ref1
cancer ref1
China ref1, ref2, ref3
Corman-Drosten test ref1, ref2, ref3, ref4
death certificates ref1, ref2
fraudulent testing ref1
genetic material, amplification of ref1
Lateral Flow Device (LFD) ref1
PCR tests ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8
vaccines ref1, ref2, ref3
Thunberg, Greta ref1, ref2, ref3
Totalitarian Tiptoe ref1, ref2, ref3, ref4
transgender persons
activism ref1
artificial wombs ref1

censorship ref1
child abuse ref1, ref2
Human 2.0 ref1, ref2, ref3
Wokeness ref1, ref2, ref3, ref4, ref5
women, deletion of rights and status of ref1, ref2
young persons ref1
travel restrictions ref1
Trudeau, Justin ref1, ref2, ref3
Trump, Donald ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11
Twitter **ref1, ref2, ref3, ref4, ref5, ref6, ref7**

U

UKColumn ref1, ref2
United Nations (UN) ref1, ref2, ref3, ref4, ref5 *see also* **Agenda 21/Agenda 2030 (UN)**
United States ref1, ref2
American Revolution ref1
borders ref1, ref2
Capitol Hill riot ref1, ref2
children ref1
China ref1, ref2
CIA ref1, ref2
Daily Pass tracking system ref1
demographics by immigration, changes in ref1
Democrats ref1, ref2, ref3, ref4, ref5, ref6, ref7
election fraud ref1
far-right domestic terrorists, pushbackers as ref1
Federal Reserve ref1
flu/respiratory diseases statistics ref1
Global Cult ref1, ref2
hand sanitisers, FDA warnings on ref1

immigration, effects of illegal ref1
impeachment ref1
Israel ref1, ref2
Judaism ref1, ref2, ref3
lockdown ref1
masks ref1
mass media ref1, ref2
nursing homes ref1
Pentagon ref1, ref2, ref3, ref4
police ref1, ref2, ref3, ref4
pushbackers ref1
Republicans ref1, ref2
borders ref1, ref2
Democrats ref1, ref2, ref3, ref4, ref5
Russia, inquiry into collusion with ref1
Sabbatians ref1, ref2, ref3
September 11, 2001, terrorist attacks ref1, ref2, ref3, ref4
UFO sightings, release of information on ref1
vaccines ref1
white supremacy ref1, ref2, ref3, ref4
Woke Democrats ref1, ref2

V

vaccines ref1, ref2, ref3
adverse reactions ref1, ref2, ref3, ref4, ref5
Africa ref1
anaphylactic shock ref1, ref2, ref3, ref4
animals ref1, ref2
anti-vax movement ref1, ref2, ref3, ref4, ref5
AstraZeneca/Oxford ref1, ref2, ref3, ref4
autoimmune diseases, rise in ref1, ref2
Big Pharma ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8

bioweapon, as real ref1, ref2
black and ethnic minority communities ref1
blood clots ref1, ref2
Brain Computer Interface (BCI) ref1
care homes, deaths in ref1
censorship ref1, ref2, ref3
chief medical officers and scientific advisers, financial interests of ref1, ref2
children ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
China ref1, ref2
clinical trials ref1, ref2, ref3, ref4, ref5, ref6
compensation ref1
compulsory vaccinations ref1, ref2, ref3
computer programs ref1
consciousness ref1
cover-ups ref1
creation before Covid ref1
cytokine storm ref1
deaths and illnesses caused by vaccines ref1, ref2, ref3, ref4, ref5
definition ref1
developing countries ref1
digital tattoos ref1
DNA-manipulation ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10
emergency approval ref1, ref2, ref3, ref4, ref5
female infertility ref1
funding ref1
genetic suicide ref1
Global Cult ref1
heart chakras ref1
hesitancy ref1
Human 2.0 ref1, ref2, ref3, ref4
immunity from prosecution ref1, ref2, ref3

implantable technology ref1
Israel ref1
Johnson & Johnson ref1, ref2, ref3, ref4
lockdowns ref1
long-term effects ref1
mainstream media ref1, ref2, ref3, ref4, ref5
masks ref1, ref2, ref3, ref4, ref5
Medicines and Healthcare products Regulatory Agency (MHRA) ref1, ref2
messaging ref1
Moderna ref1, ref2, ref3, ref4, ref5, ref6
mRNA vaccines ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
nanotechnology ref1, ref2
NHS ref1, ref2, ref3, ref4, ref5
older people ref1, ref2
operating system ref1
passports ref1, ref2, ref3, ref4
Pfizer/BioNTech ref1, ref2, ref3, ref4, ref5, ref6, ref7
polyethylene glycol ref1
pregnant women ref1
psychopathic personality ref1
races, targeting different ref1
reverse transcription ref1
Smart Grid ref1
social distancing ref1
social media ref1
sterility ref1
synthetic material, introduction of ref1
tests ref1, ref2, ref3
travel restrictions ref1
variants ref1, ref2
viruses, existence of ref1
whistle-blowing ref1

WHO ref1, ref2, ref3, ref4
Wokeness ref1
working, vaccine as ref1
young people ref1
Vallance, Patrick ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
variants ref1, ref2, ref3
vegans ref1
ventilators ref1, ref2
virology ref1, ref2
virtual reality ref1, ref2, ref3
viruses, existence of ref1
visual reality ref1, ref2
vitamin D ref1, ref2
von Braun, Wernher ref1, ref2

W

war-zone hospital myths ref1
waveforms ref1, ref2
wealth ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9 ref10, ref11
wet market conspiracy ref1
Wetiko factor ref1
alcoholism and drug addiction ref1
anti-human, why Global Cult is ref1
Archons ref1, ref2, ref3, ref4
artificial intelligence (AI) ref1
Big Pharma ref1, ref2
children ref1
China ref1
consciousness ref1, ref2
education ref1
Facebook ref1

fear ref1, ref2
frequency ref1, ref2
Gates ref1, ref2
Global Cult ref1, ref2
heart ref1, ref2
lockdowns ref1
masks ref1
Native American concept ref1
psychopathic personality ref1, ref2
reframing/retraining programmes ref1
religion ref1
Silicon Valley ref1
Smart Grid ref1
smartphone addiction ref1, ref2
social media ref1
war ref1, ref2
WHO ref1
Wokeness ref1, ref2, ref3
Yaldabaoth ref1, ref2, ref3, ref4

whistle-blowing ref1, ref2, ref3, ref4, ref5, ref6, ref7

white privilege ref1, ref2

white supremacy ref1, ref2, ref3, ref4, ref5

Whitty, Christopher ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10

'who benefits' ref1

Wi-Fi ref1, ref2, ref3, ref4

Wikipedia ref1, ref2

Wojcicki, Susan ref1, ref2, ref3, ref4, ref5, ref6, ref7

Wokeness
Antifa ref1, ref2, ref3, ref4
anti-Semitism ref1
billionaire social justice warriors ref1, ref2, ref3

Capitol Hill riot ref1, ref2
censorship ref1
Christianity ref1
climate change hoax ref1, ref2
culture ref1
education, control of ref1
emotion ref1
facts ref1
fascism ref1, ref2, ref3
Global Cult ref1, ref2, ref3, ref4
group-think ref1
immigration ref1
indigenous people, solidarity with ref1
inversion ref1, ref2, ref3
left, hijacking the ref1, ref2
Marxism ref1, ref2, ref3
mind control ref1
New Woke ref1
Old Woke ref1
Oneness ref1
perceptual programming ref1
Phantom Self ref1
police ref1
defunding the ref1
reframing ref1
public institutions ref1
Pushbackers ref1, ref2, ref3
racism ref1, ref2, ref3
reframing ref1, ref2
religion, as ref1
Sabbatians ref1, ref2, ref3
Silicon Valley ref1
social justice ref1, ref2, ref3, ref4

transgender ref1, ref2, ref3, ref4, ref5
United States ref1, ref2
vaccines ref1
Wetiko factor ref1, ref2, ref3
young people ref1, ref2, ref3
women, deletion of rights and status of ref1, ref2
World Economic Forum (WEF) ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
World Health Organization (WHO) ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9
AIDs/HIV ref1
amplification cycles ref1
Big Pharma ref1, ref2, ref3
cooperation in health emergencies ref1
creation ref1, ref2
fatality rate ref1
funding ref1, ref2, ref3
Gates ref1
Internet ref1
lockdown ref1
vaccines ref1, ref2, ref3, ref4
Wetiko factor ref1
world number 1 (masses) ref1, ref2
world number 2 ref1
Wuhan ref1, ref2, ref3, ref4, ref5, ref6, ref7 ref8

Y

Yaldabaoth ref1, ref2, ref3, ref4, ref5, ref6
Yeadon, Michael ref1, ref2, ref3, ref4
young people *see also* **children** addiction to technology ref1
Human 2.0 ref1
vaccines ref1, ref2

Wokeness ref1, ref2, ref3
YouTube ref1, ref2, ref3, ref4, ref5
WHO 548

Z

Zaks, Tal ref1
Zionism ref1, ref2, ref3
Zuckerberg, Mark ref1, ref2, ref3, ref4, ref5, ref6, ref7, ref8, ref9, ref10, ref11, ref12
Zulus ref1

ICKONIC
THE ALTERNATIVE

Ickonic is something that has been a dream of mine for the last 5 years, growing up around alternative information I have always had a natural interest in what is going on in the World and what could I do to make it better.
Across the range of subjects and positions of influence occupied mainly by people who don't strive to make things better it's the Media that I have always found the most frustrating and fascinating. Mainly because if the Media did their Jobs properly then so much of the negative things happening in the World simply would not be able to happen, because they would be exposed within a heartbeat.
Free Press and the Opportunities that the internet could have given would mean that the Media are able to expose things like never before and hold people to account for their actions. As we all know there are 'Untouchables' that walk among us, people the Media simply won't touch, expose or investigate and that leads to the dark underworlds that infest the establishment the World over.
Well I say enough, it's time for something different, a different kind of Media, where no one is off limits from exposing and investigating. All we're interested in at Ickonic is the truth of what is really going on in the World on whichever subject we're covering.
We hope you enjoy what we have created and take something away from the platform, we aim to deliver information that's informative and most importantly self-empowering, you're not a little person, you're part of something much bigger than that and its time we as a collective race began to understand that and look to the future as ours to take.

It's time...

Jaymie Icke - Founder Ickonic Alternative Media.

SIGN UP NOW AT ICKONIC.COM

DAVID ICKE

THE ANSWER

We live in extraordinary times with billions bewildered and seeking answers for what is happening. David Icke, the man who has been proved right again and again, has spent 30 years uncovering the truth behind world affairs and in a stream of previous books he predicted current events.

The Answer will change your every perception of life and the world and set you free of the illusions that control human society. There is nothing more vital for our collective freedom than humanity becoming aware of what is in this book.

Available now at davidicke.com.

THE TRIGGER

THE LIE THAT CHANGED THE WORLD – WHO REALLY DID IT AND WHY

DAVID ICKE

EVERYTHING YOU NEED TO KNOW

BUT HAVE NEVER BEEN TOLD

DAVID ICKE

DAVIDICKE.COM

DAVID ICKE STORE

LATEST NEWS ARTICLES

DAVID ICKE VIDEOS

WEEKLY DOT-CONNECTOR PODCASTS

LIVE EVENTS

WWW.DAVIDICKE.COM

THE LIFE STORY OF DAVID ICKE

RENEGADE

THE FEATURE LENGTH FILM

/ˈren·ɪˌgeɪd/

noun

A person who behaves in a rebelliously unconventional manner.

AVAILABLE NOW AT DAVIDICKE.COM

2 NEW BOOKS
BY NEIL HAGUE
ORION'S DOOR
SYMBOLS OF CONSCIOUSNESS & BLUEPRINTS OF CONTROL
- THE STORY OF ORION'S INFLUENCE OVER HUMANITY
CUTTING EDGE VISIONARY ART
& UNIQUE ILLUSTRATED BOOKS
THE DR. COVID UNIVERSE
ADVENTURES IN CLOWNLAND
NEIL HAGUE
FOR
BOOKS, PRINTS & T-SHIRTS
VISIT:
NEILHAGUEBOOKS.COM
OR NEILHAGUE.COM

Before you go ...

For more detail, background and evidence about the subjects in *Perceptions of a Renegade Mind* – and so much more – see my others books including *And The Truth Shall Set You Free; The Biggest Secret; Children of the Matrix; The David Icke Guide to the Global Conspiracy; Tales from the Time Loop; The Perception Deception; Remember Who You Are; Human Race Get Off Your Knees; Phantom Self; Everything You Need To Know But Have Never Been Told, The Trigger and The Answer*.

You can subscribe to the fantastic new Ickonic media platform where there are many hundreds of hours of cutting-edge information in videos, documentaries and series across a whole range of subjects which are added to every week. This includes my 90 minute breakdown of the week's news every Friday to explain *why* events are happening and to what end.