bcompiler Documentation

Release 1.3.15

Matthew Lemon

Feb 15, 2018

Contents:

A tool for managing DfT BICC data.

CHAPTER 1

Introduction

`bcompiler` is a tool to manage data involved in the BICC reporting process at the UK Department for Transport.

It is developed and maintained by Matthew Lemon and licensed under MIT. Source code is available at Bitbucket.

`bcompiler` processes data held in Excel files, either compiling similar data from many Excel files into a single master spreadsheet, or populating many Excel files using the data from a master spreadsheet.

"Auxiliary" files (see *Auxiliary files*) are required to map data in each direction, and to templates. These files are contained in a DfT reporitory on GitHub. `bcompiler` can be used to obtain/update these files.

1.1 Quick Start

- Ensure Python 3.6.2 or later is installed on your system.
- Ensure git is installed on your system.
- `pip install bcompiler`
- `bcompiler-init`
- Refer to *Check integrity of populated template files*.

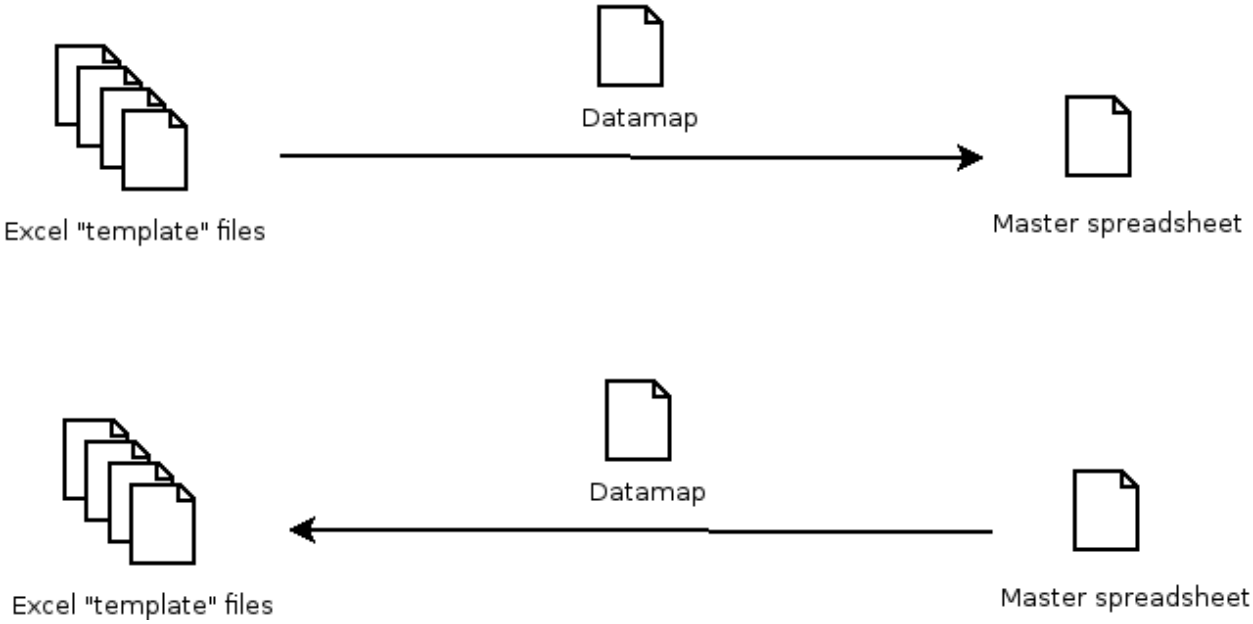
1.2 Concept

1.2.1 Template and master

The primary function of `bcompiler` is very simple: it extracts data from one Excel spreadsheet and puts it into another spreadsheet. More specifically, it extracts data from a spreadsheet which has an ad hoc layout, and multiple sheets (which we call a *template*), and puts it into a simple database-like table on a single sheet (which we call a *master*). The template is a controlled document which is intended to be completed as a form by some stakeholder and the master is a document store which holds data from multiple templates. It could therefore be said `bcompiler` is a

collection tool that gathers data from a controlled, Excel-based, user interface and "compiles" it into a central point, allowing for storage or further interrogation by other tools, such as Excel or even `bcompiler` itself, as we shall see.

This process can also be operated in reverse, i.e. data can be transferred from a master to a set of templates.



1.2.2 Datamap

A template is intended to be used as a form to collect data. It's design is free-form and should facilitate data-entry in a user-friendly way, therefore it is likely to contain:

- empty cells, for the user to complete
- locked cells, containing formulae
- locked cells, for spacing or other aesthetic purposes
- cells controlled by data verification, such as drop-down lists
- styled cells and various formatting
- any other plausible design element which facilitates successful data-entry

When seeking to extract data from a template which has been populated by a user, the task is therefore to know which cells in the template contain the data entered and which can be ignored as cells used for aesthetics, user information, spacing, design, etc. This is achieved in `bcompiler` using a *datamap*.

The datamap is a simple CSV file which maps *keys* to *values*. The key is the arbitary name, or descriptor of some piece of data you want to capture, and the value is the data contained in the cell which represents that piece of data. The job of the datamap is to tell `bcompiler` which cell in the template contains that piece of data you want to capture.

An extract from a datamap:

```
First Name,Summary,F10,
Last Name,Summary,F11
Date of Birth,Summary,G10,
Nickname,Summary,G11
```

Here, "First Name" is a *key*, whose *value* can be found in cell *F10* of the *Summary* sheet in the target template. Likewise for "Last Name", "Date of Birth", etc.

Note: The datamap (called `datamap.csv`) is an *auxiliary file* (see *Auxiliary files*), created by `bcompiler` in a special location inside the the `Documents` folder of your computer. An auxiliary file is simply a file whose contents help `bcompiler` do its job and can be amended by any user of the program.

Warning: Without a correctly populated datamap, `bcompiler` has no way of finding or placing data, so it is an essential component of the process and can be the source errors and unexpected values.

1.2.3 Designing or amending a template

The process of designing a new template (or amending and existing one) is therefore very straightforward.

The template is laid-out according to whatever design/principles are suitable. Cell-locking and other security measures are inacted within the Excel file to control where data can be entered by the user and to protect formulas, adding or deleted rows/columns, etc.

A `datamap.csv` file is then created (or amended if changing an existing template), using Excel or Notepad or any other text editing application, and each cell in the template intended to be populated by the user and/or captured by `bcompiler` is listed on a single line, in CSV (comma-seperated) format:

```
First Name,Summary,F10,
Last Name,Summary,F11
Date of Birth,Summary,G10,
Nickname,Summary,G11
Data Field 1,Finance,A3
Data Field 2,Finance,A4
...
```

The `datamap.csv` file is saved and placed in the `bcompiler/source/` directory in the computer's `Documents` directory (the name of which differs depending on whether using Windows, Mac OS X or Linux).

CHAPTER 2

Installation

Note: This guide refers specifically to installing on a Windows system as that is anticipated to be the primary operating system for typical `bcompiler` users. However, `bcompiler` is installable on Linux and Mac using the same `pip` commands. The only difference is how Python and `git` are installed on those systems. Please refer to python.org and git-scm.com.

2.1 Install Python

1. To install Python, download installer file from http://www.python.org/ftp/python/3.6.2/python-3.6.3.exe. Choose to save it to a location on your harddrive, such as your Desktop or Downloads folder.
2. Run the installer. On the Install Python Setup screen, ensure "Add Python 3.6 to PATH" and "Install launcher for all users (recommended)" is checked. Click "Install Now".
3. Open a new command window (Start -> type "cmd" in Search box and hit enter).

2.2 Update pip (if required)

- In command window, type `python -m pip install -U pip`.

2.3 Install git

1. Go to https://git-scm.com/download/win. The download will begin automatically. Save it to a location on your hardrive, such as your Desktop or Downloads folder.
2. Run the installer, accepting all default options. If you get a message saying that you cannot run the 64-bit installer, choose the 32-bit installer from the above page.

2.4 Install bcompiler

- If you do not already have `bcompiler` installed, in the command window, type `pip install bcompiler`.
- If you have `bcompiler` installed, it is a good idea to update to the latest version. In the command window, type `pip install -U bcompiler`.

Note: Use the latest version of bcompiler. You can find out what the latest version of bcompiler is by doing `pip search bcompiler`. If you can see that there is a later version, but `pip install -U bcompiler` does not install the latest version for some reason, try uninstalling bcompiler `pip uninstall bcompiler` first, then installing with `pip install bcompiler`. You can also specify which version of `bcompiler` you want to download with `pip install bcompiler==1.1.0a1` - make sure that version is listed as the latest doing `pip search bcompiler`.

CHAPTER 3

Initialise bcompiler

bcompiler needs auxiliary files to run, including a datamap.csv and config.ini files. These files are stored in a directory called bcompiler in your Documents directory. Before running bcompiler, this directory structure needs to be set up. The auxiliary files also need to be downloaded from a git repository on Github. bcompiler can do the necessary work to set this up.

- In the command window, type bcompiler-init.

Changing settings for various things in bcompiler is done using a *config.ini* file.

3.1 Auxiliary files

bcompiler requires three files to be present in the auxiliary directory, created during bcompiler-init:

- config.ini
- datamap.csv
- bicc_template.xlsm

3.1.1 config.ini

This is a text file in Documents/bcompiler/source that allows allows the user to set basic configuration options.

INI files are an informal standard for configuration files. The basic element contained in an INI file is the *key* or *property*. Every key has a *name* and *value*, delimted by an equals sign (=). The name appears to the left of the equals sign.

Keys may be grouped into sections (this is the case for bcompiler). The section name appears on a line by itself in square brackets ([and]). All keys declared after the section declaration are associated with that section.

Example:

```
[QuarterData]
CurrentQuarter = Q2 Jul - Oct 2017
```

The options available to set for `bcompiler` are:

Purpose	Description
QuarterData	In `Q2 Jul - Oct 2017`. Appears in appropriate field in template.
TemplateSheets	The names of each relevant sheet in the template must be set here
BlankTemplate	Set the name of the template kept in the *Documents/bcompiler/source directory*
Datamap	Set the name of the datamap kept in the *Documents/bcompiler/source directory*
Master	Set the name of the master file kept in the *Documents/bcompiler/source directory*

Note that sensible values are set by default. The option you will most likely need to change is `Master` as this is most often renamed by the user ourside of `bcompiler` use.

3.1.2 datamap.csv

In order for `bcompiler` to retrieve data from cells in an Excel spreadsheet, it requires a mapping between the master to the template. This is achieved in a CSV file with the following headers:

- **cell_key**: The name of the value as it appears in Column A of the master
- **template_sheet**: The name of the sheet in the template
- **cell_reference**: The cell reference of the cell where data lives in the template
- **verification_list**: **LEGACY** Not currently implemented

3.1.3 bicc_template.xlsm

The Excel file that is populated by `bcompiler` and sent to project teams and subsquently queried by `bcompiler` when populating the master spreadsheet. Contains macros to handle cell verification so must be saved in `.xlsm` format.

3.2 Other options

- In a command window, run `bcompiler --help` to see other options. **Please note**: some of these are legacy options and will be changed or removed in future versions of `bcompiler`.

CHAPTER 4

Populating templates based on a master spreadsheet

Attention: The macros explained in the *Handling RAG-colour and Data Validation macros* section below have been replaced with a single macro calledd *UniversalMacro* which will unlock all worksheets in the template, run both formatting macros and re-lock sheets Use this unless you need to debug a particular step, or you're a maschocist...

- Ensure the master spreadsheet is in the `Documents/bcompiler` directory.
- Ensure the filename of the master spreadsheet is included in the `[Master]` section in `config.ini`.
- In a command window, run `bcompiler -a`.
- The resulting files will be created in `Documents/bcompiler/output`.
- Carry out RAG-colour and Data Validation handling as *described*.
- Ensure each sheet and each workbook is protected using a password (either *View*, *Protect Sheet* and *View*, *Protect Workbook*, or by running the macro *Protect_All_Sheets*).
- Save the workbook

Warning: Make sure the password is retained by all admin users. You will not be able to amend the worksheet or workbook if the password is forgotten.

4.1 Handling RAG-colour and Data Validation macros

The BICC data collection process requires that 'blank' templates are sent to project teams using a number of data validation rules. For example, certain cells must only be populated by dates or by one a restricted list of options. This is handled by standard Excel data validation which is mostly set within the `bicc_template.xlsm` form.

However, currently the form contains **two** macros which must be run following a `bcompiler -a` operation to populate all templates from a master spreadsheet:

- *DataVerification*
- *RAG_Conditional*

which provide the template with dropdown choices on certain cells and conditional formatting on all cells whose value relates to a RAG rating. These macros are required due to limitations in creating data validation within `bcompiler` and its underlying libraries.

Unfortunately, the macros have to be run on each individual file.

To apply data validation and RAG conditional formatting, do the following:

1. Run `bcompiler -a`, as explained above.

Ensure no other Excel files are open on your machine to prevent additional macros being listed. Then, open each exported populated template in turn, and:

2. Unprotect each sheet (either *Review*, *Unprotect Sheet*, or run the *Unprotect_All_Sheets* macro)
3. Run the *DataVerification* macro (*View*, *Macros*, highlight *DataVerification*, click *Run*)
4. Run the *RAG_Conditional* macro (*View*, *Macros*, highlight *RAG_Conditional*, click *Run*)

Warning: You **must** unlock each worksheet before running the macros, otherwise you will encounter a `Run-time error '1004'` message in Excel.

CHAPTER 5

Creating a master spreadsheet from populated templates

- Ensure all populated returns are copied to the `Documents/bcompiler/source/returns` directory. Ensure no other files are present in this directory.
- In a command window, run `bcompiler` (no arguments are required).
- The resulting master file will be created in `Documents/bcompiler/output` directory.
- To compare values from a previous master, run `bcompiler --compare <PATH-TO-MASTER-TO-COMPARE>`

CHAPTER 6

Check integrity of populated template files

The template used to collect data should not be changed by the user; allowing the user to add rows or columns will cause a world of problems for `bcompiler`. To ensure the integrity of the template, sheets in `bicc_template.xlsm` are locked to prevent rows being added or deleted.

However, `bcompiler` is able to check the validity of all returned templates if required, by comparing the number of rows in each sheet with what it expects from `bicc_template.xlsm`.

- Ensure all populated returns are copied to `Documents/bcompiler/source/returns`.
- In a command window, run `bcompiler -r`

This will print the count of rows in each sheet in each template file. Any row count that differs from the equivalent sheet in `bicc_template.xlsm` will be marked with a *.

- To output this data to the `Documents/bcompiler/output` directory, run `bcompiler -r --csv`.
- To only show differences between the file and `bicc_template.xlsm`, run `bcompiler -r --quiet`.

CHAPTER 7

Extending bcompiler

The main functionality of `bcompiler` is obtained via the command line, e.g:

```
bcompiler -h
```

is used to obtain the basic help menu.

In addition, `bcompiler` allows anyone with a knowledge of basic Python to be able to interact with the program and to generate their own output. Some parts of `bcompiler` are 'exposed' to the user via an API (Application Programming Interface) which is designed to be easy to use and useful as a component to building new functionality.

In this version of `bcompiler` (1.3 series), the API is very limited, however it allows you to interrogate the data held in an master xlsx file and do things with the data. Essentially `bcompiler` does the hard work of pulling the data out of a master, formatting it in some way, and presenting it to you in a format for doing something else with, e.g. writing it to another file, such as an Excel or a Word document.

The key API objects documented here are:

- *Master*
- *Quarter*
- *FinancialYear*
- *Row*

7.1 Examples

7.1.1 Filtering project data

One of the most simple tasks might be to list the projects contained with a particular master file:

```
from bcompiler.api import project_data_from_master
m = project_data_from_master('/tmp/master_1_2017.xlsx', 1, 2017)
m.projects
```

output: `['Project Name 1', 'Project Name 2', ...]`

Say you wish to interrogate a master file and output all values from keys which contain the word "Total" in a project whose title is "Project Name 1". Here's what you could do:

```
from bcompiler.api import project_data_from_master
m = project_data_from_master('/tmp/master_1_2017.xlsx', 1, 2017)
p = m['Project Name 1']
totals = p.key_filter("Total")
```

output: `[('Import Total Budget/BL', 10), ('Another Total Budget/BL', 199.1),` ...

7.1.2 Checking for duplicate keys in a master

A master file containing duplicate keys will not function correctly. `bcompiler` will test for this during its normal operation, but if you wish to check a master file yourself, you can do this very easily once you have a `bcompiler.api.Master` object, obtained using the `project_data_from_master` function as demonstrated above, or by directly creating a `bcompiler.api.Master` object (see *Master*):

```
from bcompiler.api import project_data_from_master
m = project_data_from_master('/tmp/master_1_2017.xlsx', 1, 2017)
m.duplicate_keys()
```

output: `False` if there are no duplicate keys, `True` otherwise.

7.1.3 Computing financial quarter/year dates

You're writing a script that requires computation involved with financial years...:

```
from bcompiler.api import FinancialYear
fy = FinancialYear(2016)
fy.start_date
```

output: `datetime.date(2016, 4, 1)`:

```
fy.end_date
```

output: `datetime.date(2017, 3, 31)`:

```
quarter1_2016 = fy.q1
```

output: `Quarter(1, 2016)`:

```
quarter1_2016.fy
```

output: `2016`:

```
quarter1_2016.end_date
```

output: `datetime.date(2016, 6, 30)`

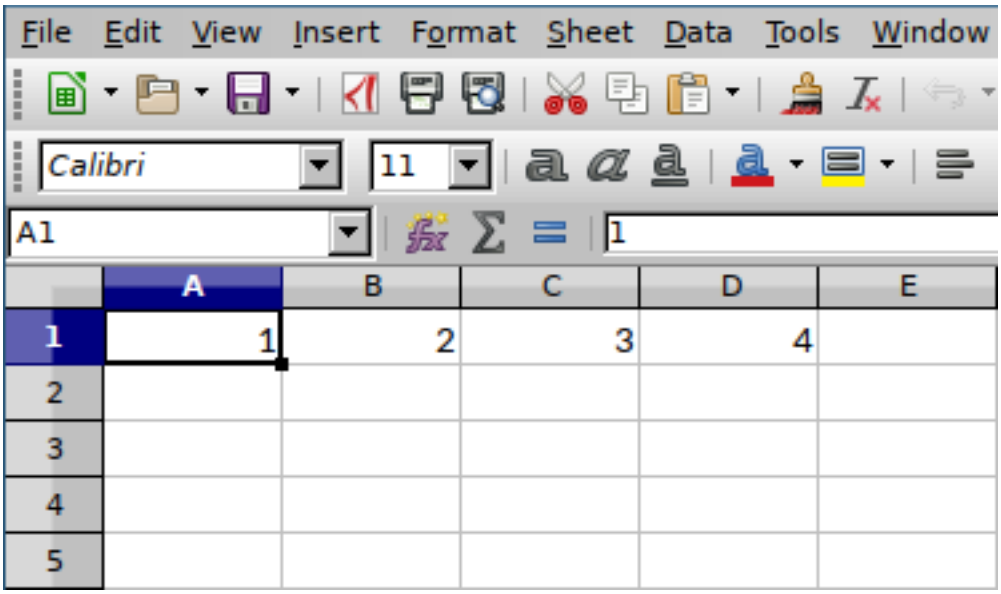
7.1.4 Writing data to a new Excel file

You are writing a program that exports data from a master file to another workbook, in the same way that bcompiler analysers work.

To write data into a row in your workbook, `bcompiler` will do the hard work for you - you don't have to write data into individual cells. Use the `bcompiler.api.Row` object:

```
from openpyxl import Workbook

from bcompiler.api import Row

data = [1, 2, 3, 4]

wb = Worbook()
ws = wb.active

r = Row(1, 1, data)
r.bind(ws)

wb.save('/tmp/test.xlsx')
```

output: an Excel file at `/tmp/test.xlsx` whose default sheet contains a row of values: `1 2 3 4`, starting at cell `A1` (or 1, 1).



This could be combined with other elements of the API, for example to write the list of project titles from a master file to a new Excel file:

```
from openpyxl import Workbook

from bcompiler.api import Row
from bcompiler.api import project_data_from_master

m = project_data_from_master('/tmp/master_1_2017.xlsx', 1, 2017)
projects = m.projects

wb = Worbook()
ws = wb.active

r = Row(1, 1, projects)
r.bind(ws)

wb.save('/tmp/test.xlsx')
```

You can also use the column letter as the first parameter in the `Row()` function:

```
r = Row('A', 1, projects)
```

and the effect will be the same.

7.2 API Reference

7.2.1 Master

project_data_from_master

As well as dealing with `Master` objects directly, the `bcompiler.api.project_data_from_master()` function does exactly the same job.

Note: This function is not the same as `bcompiler.utils.project_data_from_master()`. That function produces a complex data structure containing a dictionary of `collections.OrderedDict` objects, whilst this one returns a `bcompiler.api.Master` object, which is more user-friendly to work with.

7.2.2 Quarter

class bcompiler.api.**Quarter**(*quarter: int*, *year: int*)

A Quarter object enapsulates data about a financial quarter in `bcompiler`. Because it contains data about dates (start dates and end dates for a particular quarter, for instance), it can be used for calculating differences between dates and ordering objects which are associated with it. A good example is a `bcompiler.api.Master` object, which is a composition of an Excel file (providing the data) and a *`bcompiler.api.Quarter`* object (providing temporal data). This allows `bcompiler` and anyone using a `bcompiler.api.Master` object to order data by date.

To create a `Quarter` object is very easy:

```
from bcompiler.api import Quarter
q1 = Quarter(2, 2015)
```

The following attributes of the resulting object are available:

year

An integer representing the calendar year.

quarter

An integer representing the quarter (1, 2, 3 or 4)

start_date

A `datetime.date` object

end_date

A `datetime.date` object

fy

A `bcompiler.api.FinancialYear` object

7.2.3 FinancialYear

7.2.4 Row

See an example of `Row` in use: *Writing data to a new Excel file*

CHAPTER 8

Analysers

8.1 Introduction

`bcompiler` is able to conduct basic analysis on spreadsheets. An analyser will usually process some data in a master spreadsheet and produce another spreadsheet (CSV, Excel), an Excel chart, commandline output, or some other data type.

Built-in analysers can be used in **two** ways:

- from the command line
- importing into your own Python programs

Analysers available from the commandline use mostly default options and are relatively limited. More extensive configuration can be gained by writing your own scripts and importing bcompiler analyser code into your project to help you. See *Importing analyser code into your own projects* for more details.

8.1.1 Running from the commandline

Basic command

```
>> bcompiler --analyser ANALYSER OPTIONS
```

Available options

Available to all analysers

- `--master PATH_TO_DIRECTORY_CONTAINING_MASTER`

Available to swimlane_milestones analyser

The default is chart milestones within a range of 365 days from today. However, the following options are available to give greater control to this band:

- `--output PATH_TO_OUTPUT_DIRECTORY`
- `--start_date DATE (dd/mm/yyyy)`
- `--end_date DATE (dd/mm/yyyy)`

8.1.2 Importing analyser code into your own projects

Warning: This functionality is not yet implemented.

```
from bcompiler.analysers import Swimlane

s = SwimlaneMilestones()
s.output('/home/user/Desktop/swimlane_milestones.xlsx')
s.add_to_worksheet(worksheet)
workbook.save()
```

8.2 Built-in Analysers

8.2.1 rcf

Perform Reference Class Forecasting on selected master files. Target master files must be named according to this pattern: `*_N_YYYY.xlsx` where `N` represents a number between 1 and 4 and `YYY` represents a year. This file therefore represents the Nth quarter of year Financial Year YYYY.

Default >> bcompiler --analyser rcf

Default options require master files to be referenced in `Documents/bcompiler` directory A single workbook for each project is output to the `Documents/bcompiler` directory.

Output files to a different directory >> bcompiler --analyser rcf --output C:\Users\jim\Desktop

This options requires the master files to be present in the `Documents/bcompiler` directory. The data is output to the directory specified after the `--output` flag, in this case `C:\Users\jim\Desktop`.

Set target master directory manually

```
>> bcompiler --analyser rcf --master C:\Users\jim\Downloads
```

This options requires a master files to be present in the `C:\Users\jim\Downloads` directory, named `q1_master.xlsx`. The files are output to `Documents/bcompiler/output` directory.

8.2.2 financial analysis

Output a workbook containing a single worksheet which includes a graph mapping change in certain financial data from designated quarters. **Must** include **four** quarters, named correctly in `config.ini` file.

Default >> `bcompiler --analyser financial`

Default options require master files to be referenced in `config.ini` file and present in `Documents/bcompiler` directory. A single workbook for each project is output to the `Documents/bcompiler` directory.

Output files to a different directory >> `bcompiler --analyser financial --output C:\Users\jim\Desktop`

This options requires the master files to be present in the `Documents/bcompiler` directory, and referenced in the `config.ini` file. The data is output to the directory specified after the `--output` flag, in this case `C:\Users\jim\Desktop`.

8.2.3 keyword

Search for a keyword in the master key column (Column A) (e.g. RAG, or SRO). By default, outputs to terminal.

Default

```
>> bcompiler --analyser keyword "RAG"
```

Default options require a master file to be present in the `Documents/bcompiler` directory, named `target_master.xlsx` as per the `config.ini` file.

Output is sent to your terminal.

Warning: Terminal output will exceed 80 characters. If you are using Windows, you should go to Preferences in `cmd` application and increase the width of the terminal window to something like 150 characters.

Output to xlsx (Excel) file

```
>> bcompiler --analyser keyword "RAG" --xlsx C:\Users\jim\Desktop\rag.xlsx
```

This options requires a master file to be present in the `Documents/bcompiler` directory, named `target_master.xlsx` as per the `config.ini` file. The data is output to the file specified after the `--xlsx` flag, in this case `C:\Users\jim\Desktop\rag.xlsx`.

Output to xlsx (Excel) and get data from a specific master

```
>> bcompiler --analyser keyword "RAG" --xlsx C:\Users\jim\Desktop\rag.xlsx
--master C:\Users\jim\Downloads\q1_master.xlsx
```

This options requires a master file to be present in the `C:\Users\jim\Downloads` directory, named `q1_master.xlsx`. The data is output to the directory specified after the `--output` flag, in this case `C:\Users\jim\Desktop\rag.xlsx`.

8.2.4 annex

Creates individual project spreadsheets pulling out pertinent headline and textual data from a master. Intended to be used a Annex to BICC report. The analyser relies on two master files to be present: a master representing current data and one representing historical data. This is to allow for annex to report a "DCA Last Quarter" value.

Default

```
>> bcompiler --analyser annex
```

Default options require a master file to be present in the `Documents/bcompiler` directory, named `target_master.xlsx` as per the `config.ini` file, and a second master file, perhaps representing the previous quarter, named `compare_master.xlsx` in the same directory. You can use different filenames but this must be reflected in `[MasterForAnalysis]` and `[AnalyserAnnex]` in `config.ini`.

Set compare master manually (overriding value in `config.ini`)

```
>> bcompiler --analyser annex --compare C:\Users\jim\Desktop\q1_master.xlsx
```

Set output directory manually (overriding default of Documents/bcompiler/output

```
>> bcompiler --analyser annex --output C:\Users\jim\Desktop
```

This options requires a master file to be present in the `Documents/bcompiler` directory, named `target_master.xlsx` as per the `config.ini` file. The files are output to the directory specified after the `--output` flag, in this case `C:\Users\jim\Desktop`.

Set output directory manually (overriding default output directory of Documents/bcompiler/output and master set in `config.ini`

```
>> bcompiler --analyser annex --output C:\Users\jim\Desktop --master
C:\Users\jim\Downloads\q1_master.xlsx
```

This options requires a master file to be present in the `C:\Users\jim\Downloads` directory, named `q1_master.xlsx`. The files are output to the directory specified after the `--output` flag, in this case `C:\Users\jim\Desktop`.

Set target master manually (overriding default set in `config.ini`)

```
>> bcompiler --analyser annex --master C:\Users\jim\Downloads\q1_master.xlsx
```

This options requires a master file to be present in the `C:\Users\jim\Downloads` directory, named `q1_master.xlsx`. The files are output to `Documents/bcompiler/output` directory.

8.2.5 swimlane_milestones

Specific analyser uses project data from a master file and creates a new Excel scatter chart, showing a timeline of major **approval** milestones horizontally in swimlane fashion.

Note: By default, the swimlane chart will be produced with multi-coloured markers. If you wish all markers to be grey, ensure the following setting is present in `config.ini`:

```
[AnalyserSwimlane]
grey_markers = true
```

Note: Basic configuration for milestones analysers is done in `config.ini`. Documentation for these is contained in comments in the file.

Default options

```
>> bcompiler --analyser swimlane_milestones
```

Default options require a master file to be present in the `Documents/bcompiler` directory, named `target_master.xlsx` as per the `config.ini` file. The chart is output in a file called `swimlane_milestones.xlsx` in the `Documents/bcompiler/output` directory.

By default, the analyser will chart only those milestones that fall within 365 days of today. This can be changed in `config.ini` by changing the `range` value in the `['AnalyserSwimlane']` section.

Set output directory manually (overriding default of Documents/bcompiler/output

```
>> bcompiler --analyser swimlane_milestones --output C:\Users\jim\Desktop
```

This options requires a master file to be present in the `Documents/bcompiler` directory, named `target_master.xlsx` as per the `config.ini` file. The chart is output to the directory specified after the `--output` flag, in this case `C:\Users\jim\Desktop`.

Set output directory manually (overriding default output directory of Documents/bcompiler/output and master set in `config.ini`

```
>> bcompiler --analyser swimlane_milestones --output C:\Users\jim\Desktop
--master C:\Users\jim\Downloads\q1_master.xlsx
```

This options requires a master file to be present in the `C:\Users\jim\Downloads` directory, named `q1_master.xlsx`. The chart is output to the directory specified after the `--output` flag, in this case `C:\Users\jim\Desktop`.

Set target master manually (overriding default set in `config.ini`)

```
>> bcompiler --analyser swimlane_milestones --master C:\Users\jim\Downloads\q1_master.
xlsx
```

This options requires a master file to be present in the `C:\Users\jim\Downloads` directory, named `q1_master.xlsx`. The chart is output to `Documents/bcompiler/output` directory.

Set start and end date

```
>> bcompiler --analyser swimlane_milestones --start_date 20/1/2016
--end_date 20/1/2017
```

8.2.6 swimlane_assurance_milestones

As *swimlane_milestones* but showing **assurance** milestones.

CHAPTER 9

Roadmap

`bcompiler` makes use of semantic versioning and therefore follows the MAJOR.MINOR.PATH version pattern.

9.1 2.0 - Plugins

- Allow integration of own analysers written in Python
- Simple plugin management interface through commandline

9.2 1.0 - Analysers

- Commandline analysers for simple features
- API for analysers to be customised and used outside bcompiler
- `bcompiler-init` wrapper for auxiliary files repository so user doesn't have to push, pull and merge in git

9.2.1 Commandline analysers

Analyser	Product	Status
swimlane_milestones	Excel chart	Implemented
financial_analysis	Excel spreadsheet	
report_annex	Excel spreadsheet	
project_list	terminal output	
sro_list	terminal output	
rag_ratings	terminal output	

9.2.2 API

Analyser	Product	Status
swimlane_milestones	SwimlaneChart()	
others. . .		

9.3 0.0 - Stability

- Compile master from populated templates
- Populate templates from master
- Commandline interface
- Test suite
- Clean data in both directions
- Integrate with auxiliary files repository
- `bcompiler-init` to set up project
- Documentation

CHAPTER 10

Changes

10.1 v1.3.19

- annex analyser fixes

10.2 v1.3.18

- change to require openpyxl 2.4.9

10.3 v1.3.17

- date fix

10.4 v1.3.16

- introduced the bcompiler.api module
- updated docs to cover basic API and give examples of use

10.5 v1.3.15

- fixed encoding bug that was preventing running of –compare function

10.6 v1.3.14

- provisional fix Windows character encoding bug
- provisional fix Excel file corruption

10.7 v1.3.13

- small change to financial analyser to allow additional keys to be collected in certain circumstances
- improved date handling
- improved string cleaning of master keys
- various bug fixes

10.8 v1.3.12

- improvements to `bcompiler-init` bootstrapping functionality

10.9 v1.3.11

- significant speed optimisation when using `bcompiler -a` option
- when doing `bcompiler -a` will warn if master contains duplicate keys, which aren't allowed

10.10 v1.3.10

- Fixed bug where rcf analyser wouldn't run with no arguments
- Removed necessity to have to stipulate different keys for Q3 and Q4 in financial analysis
- Improved test speed; better test coverage
- Feneral improvements and rationalisations in template population code
- Bug fixes

10.11 v1.3.9

- ability to output only grey markers on the swimlane analyser charts

10.12 v1.3.8

- new Reference Class Forecasting analyser

10.13 v1.3.7

- new financial analysis analyser

10.14 v1.3.6

- Chart is based on start_date option when using swimlane analysers, rather than today's date.
- swimlane charts use 30 as main x axis unit rather than 50 to approximate months.

10.15 v1.3.5

- Bug fixes

10.16 v1.3.4

- Fixed bug whereby creating an annex from a master containing a project not in the compare master threw an error
- Fixes for annex analyser

10.17 v1.3.3

- new swimlane assurance milestones analyser
- annex analyser now does comparison with previous master document
- fix issues in annex analyser

10.18 v1.3.2

- Partial fix for final project milestone not ending up on swimlane chart.

10.19 v1.3.1

- Fixed bug which prevented setting the title of the output sheet from the keyword analyser with xlsx output option, to a disallowed character.

10.20 v1.3.0

- Added keyword analsyer. Search fields in a master file and return the values for each field, for each project in the terminal or optionally to an xlsx file.

10.21 v1.2.2

- Ability to set `--start_date` and `--end_date` parameters for `swimlane_milestones` analyser.
- Fix bug where date differences not being calculated correctly in `swimlane_milestones` analyser.
- Fix bug where wrong milestone type was being charted by `swimlane_milestones` analyser.
- Many more configurations available in `config.ini` file relating to `swimlane_analyser`.
- Better logging to `bcompiler.log` during `swimlane_milestones` analyser.
- Better handling of date objects.
- Various bug fixes

10.22 v1.2.1

- Added `annex` analyser, allowing for easy summarise by project from master.
- Added ASCII art to `bcompiler --help`!
- Various bug fixes

10.23 30 October 2017

- Fix bug where not all columns in master are being processed during swimlane analyser.

10.24 17 October 2017

- Changed ERROR log message to WARNING to accommodate dates mixed with free text.

10.25 16 October 2017

- Fix bug where cell value in string and datetime value would try to compare arithmetically.

10.26 11 October 2017

- Fix bug where *.xlsx* files not being picked up.
- Improved exception handling and bug fixes.

10.27 10 October 2017

- Handling cp1252 encoding coming through from Windows
- Added CHANGES.txt
- Minor bugfixes

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Index