The Linkage Coder program User Manual

Wouter Spekkink

October 16, 2017

Contents

Chapter 1

Introduction

1.1 The Linkage Coder program

The document you are currently reading is the user manual for the open source (GPL 3.0) software tool called ***Linkage Coder***, written by me (Wouter Spekkink) in C++ and Qt5 [1].

The program is intended to facilitate in the qualitative coding of event data. In this case, event data refers specifically to bracketed, chronologically ordered, qualitative data that captures activity that occurred in some type of process. My thinking about what event data is, and much of the terminology that I use, is heavily inspired by activities carried out as part of the Minnesota Innovation Research Program[2]. I have many other sources of inspiration that I will not mention in this manual. It should be easy enough to find my other sources of inspiration in the publications I have written (see my contact details in chapter 5).

Why is it useful to have this tool? Over the past years I have worked together with other people (especially Frank Boons) to develop and use various methods and tools for the study of social processes. In doing so, we have always worked with the principle that our fundamental units of analysis should be events, that is, things that "happened" or "came to pass", as opposed to, for example, variables. Based on ideas developed by the researchers from the Minnesota Innovation Research Program, we started collecting data in the form of *incidents*, which are qualitative, bracketed descriptions of activity, which consist (at least) of (1) an indication of the time at which the activity occurred, (2) a (brief) qualitative description of the activity, and (3) the source of data (e.g., a document, an interview, personal conversation)[3]. During the collection of

[1]The source code is available from my Github page (see chapter 5).

[2]see especially Poole, M. S., Van De Ven, A. H., Dooley, K., & Holmes, M. E. (2000). *Organizational Change and Innovation Processes: Theory and Methods for Research.* Oxford: Oxford University Press. I was not involved in this work in any way.

[3]The descriptions (point 2) are typically created by the researcher him/herself. When I

event data, we create numerous incidents, which we store in chronologically ordered event data sets.

Again, following ideas developed by researchers from the Minnesota Innovation Research Program, we have been using qualitative coding procedures as a step in the analysis of event data. In doing so, we treat our incidents as *empirical observations* that serve as indicators for *theoretical constructs* that cannot be observed directly. Examples of such theoretical constructs are (types of) *events* and *linkages between events*. We may, for example, treat incidents that describe a sequence of activities as indicators for the performance of a social practice[4] (an event). If we record multiple such performances, we might also be interested in knowing whether some of the performances were performed in response to others[5] (a type of linkage between the events). The tool discussed in this manual, the ***Linkage Coder*** tool, is dedicated to the coding of incident data to identify linkages between events. I also developed another tool, called the ***Event Coder*** program, the primary purpose of which is to facilitate the coding of incident data to identify and qualify events (also see chapter 4).

This manual gives an in-depth introduction to the program and its functionality. In the remainder of the introduction, I offer some additional background to the program, and I introduce some necessary disclaimers. Consider reading these disclaimers before you send me angry emails. In chapter 2 I go into some of the assumptions that the program builds on (e.g., about the data sets that you will code), and what preparations this implies (i.e., things you are assumed to have done before using the program). In chapter 3 I go through all the features of the program, including importing data, saving and loading files, coding data, and exporting the results. I conclude the main body of the manual with chapter 4, discussing some of the things that one could do with the data that the program exports. My contact details can be found in chapter 5.

1.2 Part of a bigger program

It is very important for the reader to realise that, even though I wrote the ***Linkage Coder*** as a standalone program, I wrote it primarily with the intention to later integrate it, as a module, in a larger program. The idea is that, in the larger program, you would be able to do everything from entering data into your data set, coding the data in various ways, doing basic forms of analysis, creating visualisations, and exporting data files that can be easily imported into other useful software. The current (standalone) version of the program

am using textual sources (e.g., documents, web pages, interview transcripts), I also like to add the "raw" text on which my incident description was based.

[4]If you are unfamiliar with the concept of social practice, see the following paper for a good introduction: Reckwitz, A. (2002). Toward a Theory of Social Practices: A Development in Culturalist Theorizing. *European Journal of Social Theory*, 5(2), 243âĂŞ263.

[5]See the concept of *chains of action* in Schatzki, T. R. (2016). Keeping Track of Large Phenomena. Geographische Zeitschrift, 104, 4âĂŞ24.

still requires you to (1) create your data set with external spreadsheet software (e.g., LibreOffice Calc or Excel), (2) use external software to further visualise and analyse the coded data.

And here is a very important thing to consider before you start using the current version of the program: I would advice you to at least skim through chapter 4 first. In this chapter, I offer some examples of how the data that the program exports can be imported into other software, and what possibilities this offers. Making use of this program probably only makes sense if these possibilities are actually interesting for your work.

Please, also see section 3.8 to get a good idea of what kind of data the program actually exports. The motivations for the choices that I made in this are perhaps not immediately obvious to everyone, and probably for some they will not become obvious until I actually get to integrating my work in the larger program that I have in mind. However, one thing I can say is that I have a strong preference for storing, visualising, and analysing event data (and related data) in some type of graph format. This can be achieved most efficiently if data are eventually stored either as **nodes**, **relationships between those nodes**, or **properties of nodes or relationships**. This is the reason why the data that the program exports take the form of nodes and edges, which allows you to import and visualise the data into various software tools for network visualisation [6].

1.3 Other disclaimers

Throughout the introduction, I already snuck in a few disclaimers, such as the fact that my work on this particular (standalone) version of the program will probably stop once I get to integrating it as a module into a larger program. There are several additional disclaimers I would like to make here.

It is very important that the user realises that I wrote this program, in the first place, for personal use, and that I only make it available for free in case other people may find it helpful in their work. However, you will use the program at your own risk. I do **not** take any responsibility for problems that you run into when using the program. That being said, I am always open to receiving comments, positive or negative, about problems that may occur, bugs you encounter, features you would like to be included, and so on. If you do indeed run into problems as a result of using this program, I am willing to help you think of a solution. You can find my contact details in chapter 5 of this manual.

The user should also realise that I am **not** a professional programmer. With some help from the Internet, I taught myself how to code to keep my mind occupied during some of the lonely evening hours I spent in a campus apartment somewhere in Shenyang. I never had any formal training in writing code, and

[6]see, for example, https://gephi.org, which is my favourite.

I probably make a lot of ugly mistakes, write inefficient code, and overlook simple solutions for the various problems I face while writing code. While I always try to improve my coding skills, this requires a lot of time, energy, and verbal abuse of my computers[7]. This means that I will not always have the time, energy and/or the skills required to fix certain bugs, add new features, and etcetera.

I also do **not** have a team of beta-testers to help me test the program. You are basically it. Congratulations! If I had T-shirts, I would ship you one to give you some recognition, but unfortunately I have no budget for that. The program is still fairly simple in structure, and I tried to get rid of most bugs by doing my own testing. However, the program is already complex enough for me to overlook things, so there is a good possibility to several annoying bugs remain. The only way to find these, sadly, is to encounter them while using the program. Fortunately for you, if there is one thing that I really hate, it is having bugs in my programs. If you do encounter a bug, please contact me (see chapter 5) ASAP, and I (or we) will try to figure out what is wrong, so that I can fix it.

[7]If you happen to be someone with more experience in writing code, feel free to go to my Github page (see chapter 5) to inspect the source code of this program, and if you have the time and patience, please offer me suggestions on how to improve my skills.

Chapter 2

Preparations

2.1 It is all about the data

As mentioned in section 1.2, the ***Linkage Coder*** program is designed to be eventually integrated into a larger program. In that larger program, data would be entered into, and managed by the program directly. For now, however, the ***Linkage Coder*** program needs to import all data from external sources, that is, files created by the user before using the program. The program can be understood to make some assumptions about the nature of these files, and if these assumptions are not met while reading files, the program may not work as it should.

Fortunately, creating files that can be read by the program is not difficult. Moreover, the ways these files should be created follow logically from the kind of research approach that this program is designed to fit into. In this chapter, I go into all things that the user should think of well before using this program, preferably in the stage before data collection. Most of these details have to do with the kind of data sets that the program "expects" you to work with.

2.2 Data sets

The program expects that you will import data from what I will call an *event data set*, which is nothing more then a table of chronologically ordered *incidents*. The idea of having such data sets, and the concept of incidents are both based on ideas developed during the Minnesota Innovation Research Program[1]. An incident is a bracketed, qualitative description of an observed activity, which includes, at least, the following information:

1. An indication of the time at which the incident occurred.

[1]see especially Poole, M. S., Van De Ven, A. H., Dooley, K., & Holmes, M. E. (2000). *Organizational Change and Innovation Processes: Theory and Methods for Research.* Oxford: Oxford University Press. I was not involved in this work in any way.

2. A brief (qualitative) description of the activity, including a description of the actors/entities responsible for the activity.

3. A reference to the source of data.

This is still a very open description of what an incident actually can contain. As a social scientist, I am typically mostly interested in *human activity*, that is, in activity performed by human beings. That, in itself, is still a very broad category, and it leaves open the question at what level of abstraction that activity can and/or should be described[2]. We may also understand the concept of activity in a much broader sense, such that we also include natural occurrences, such as volcanic eruptions, an apple that falls from a tree, or the wind that blows. In the end, what makes sense to count as activity will depend on your research questions, your theoretical orientation, and probably on who you count among your favourite philosophers.

Fortunately, the ***Linkage Coder*** program does not care about your definition of activity. However, whatever definition you decide to work with, the program does care about how exactly you store information on that activity as incidents in your data sets. In the type of event data sets that we work with here, the very first row of data is always the **header** of the file, which contains the names of the columns in the data file (see figure 2.1). All the remaining rows of the data set represent individual incidents. The three aspects of incidents that we mentioned above (timing, description and source) should be stored in separate columns. You may label and position these columns however you like, but we expect these columns to be there.

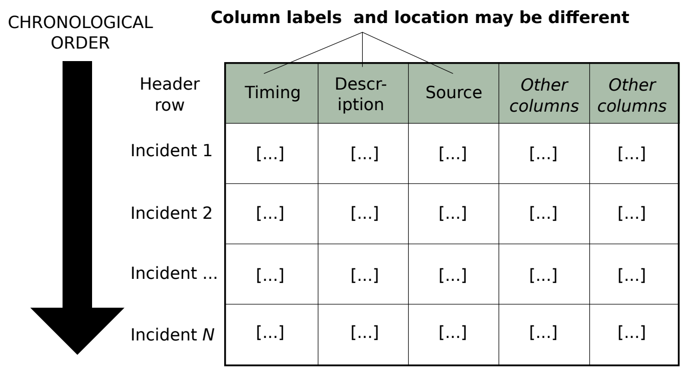
You may also add additional columns to your data set, to expand on the information about incidents that the data set records. For example, I made a habit of also including fragments of text from the original sources of data on which I based my incident descriptions. In my opinion, this makes the process of coding incidents both easier, and more transparent.

Another important assumption that the ***Linkage Coder*** program makes about your data set is that it is chronologically ordered. That means that the incident reported in a given row is assumed to have happened after the incident preceding it (of course, with the exception of the first incident in the data set). Even if two or more incidents actually happened simultaneously, based on their position in the data set, the program will always assume that there is an order in their occurrence. In practice, this should not be a serious problem.

These are basically all the assumptions that the program makes about the overall structure of your data set. However, there are some other practical considerations to be taken into account. The most important one is that the

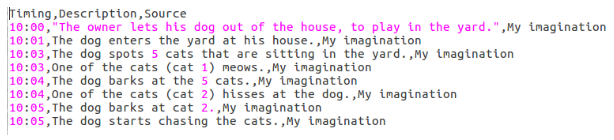
[2]One of my other sources of inspiration is the work of Peter Abell on Comparative Narratives. Abell's work engages more or less directly with questions of how to describe activity at different levels of abstraction. See especially: Abell, P. (1987). *The syntax of social life: the theory and method of comparative narratives.* Oxford, Angleterre: Clarendon.

Figure 2.1: The expected structure of event data sets.



Linkage Coder program can only import data from so-called *comma-delimited files* (I will refer to them as csv files), which can be recognised by their "*.csv" extension (e.g., "My_Dataset.csv"). These files are very much like plain text files ("*.txt"), but they use so-called *delimiters* (most often commas or semi-colons) to distinguish between different columns. You can open csv files in a basic text editor to see what this looks like (see figure 2.2).

Figure 2.2: What a csv file looks like in a basic text editor.



```
Timing,Description,Source
10:00,"The owner lets his dog out of the house, to play in the yard.",My imagination
10:01,The dog enters the yard at his house.,My imagination
10:03,The dog spots 5 cats that are sitting in the yard.,My imagination
10:03,One of the cats (cat 1) meows.,My imagination
10:04,The dog barks at the 5 cats.,My imagination
10:04,One of the cats (cat 2) hisses at the dog.,My imagination
10:05,The dog barks at cat 2.,My imagination
10:05,The dog starts chasing the cats.,My imagination
```

Spreadsheet software, like Excel or LibreOffice Calc, typically allows you to store tables as csv files, using the **"Save As"** option[3]. Some software, like Excel, will use a delimiter that is set "system-wide". If you are unsure what delimiter your system uses, just open your csv file with a text editor, and see what characters are used to separate columns. You will need to know what delimiter your files use to be able to import them properly. Other software, like LibreOffice Calc, allows you to choose what delimiter to use when you save the csv file.

So, you can create your data set using in any spreadsheet software that you

[3]I have encountered a few people who misunderstood how this works, and who simply tried to convert files by changing their extension. For example, they would rename a file called "My_Dataset.xls" to "My_Dataset.csv". This will not work, because it will not actually change the file itself. It is more likely that you will just render your file unreadable for most software, because the software will think it is reading a csv file, while it is really reading an xls file that is only disguised as a csv file.

prefer. You can also store your data set in any format that you prefer while building it. The important thing is that you should be able to store it as a simple csv file as soon as your are ready to import it into the ***Linkage Coder*** program. For example, while working on my data sets, I typically store my data in ods files or xls files, but as soon as I want to start coding my data, I will use the **"Save as"** feature of whatever program I am working with to store a csv file.

Other notes

There is a little bit more going on in csv files than what is shown in a regular text editor. What figure 2.2 does not show, for example, is that there are also hidden characters present in the file that tell the computer where each line of data ends (so-called newline symbols: \n). You usually do not have to worry about the presence of such characters, with some small exceptions.

The ***Linkage Coder*** program is not able to read csv files that have so-called newline symbols (\n) or carriage return symbols (\r) *within* their text cells. The reason for this is that the program uses a relatively simple csv file parser, which will think that a new line of data will start after encountering one of these symbols (each line of data in a csv file will end with a newline symbol by default). Fortunately, inserting such symbols into the text cells will only happen if you deliberately create them, or accidentally paste them into your file.

The program is typically able to recognise when an 'illegal' newline is encountered, and it will throw an error (see figure 3.2). Solving the error is left to the user. The problem can be solved by removing all newline symbols and carriage return symbols from the csv file. These symbols are not visible in programs like Excel or LibreOffice Calc, and in both programs you will need to use special search and replace options to get rid of the unwanted symbols. I advise you to Google for "Find and replace regular expressions with [your spreadsheet program]".

Chapter 3

Using the program: Basic features and coding

3.1 Loading a new dataset

Assuming that you have a data set ready, importing the data into the program works as follows. You first need to select the csv file containing your data. For this you will click the **Select File** button (see figure 3.1), which will open a file dialog that you can use to navigate to, and select the file.

Once a file has been selected, you will need to select the delimiter symbol that is used in the csv file to distinguish between different columns of the data table. For this, you can use the dropdown menu that reads **-Select delimiter-** by default. Four different symbols are allowed as delimiter, which are the comma (,), the semicolon (;), the colon (:), and the vertical bar (|). Make sure that the delimiter that you select matches the one used in the file.

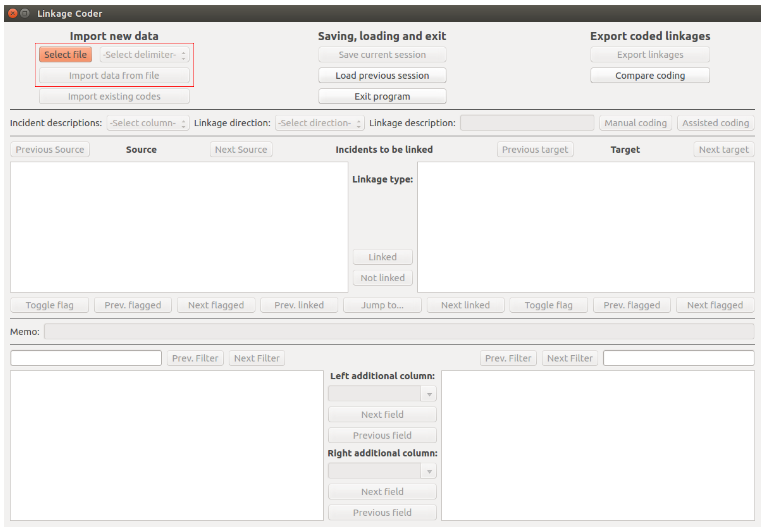
Once you have selected a delimiter, you can import the data, using the **Import data** button. Once you click this button, the program will attempt to read data from selected file and, if successful, enable all other options of the program, allowing you to start coding.

Problems when importing data

If you (1) selected a valid csv file, (2) selected the correct delimiter for this file, and (3) structured your data set using instructions offered in section 2.2, you should encounter no problems when importing the data. If something goes wrong when importing data, then the problem will usually lie with one of these three points.

One possibility is that you have not selected a valid csv file. I have encountered a few people that have tried to create csv files from (for example) xls-files by simply changing the file extension. Doing this will not actually create a valid csv file that can be read by the program. The correct way for creating csv files

Figure 3.1: Options to import data.

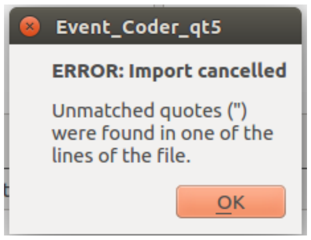


is to use the **Save as** option in your spreadsheet editor, and select to save the file with the ***.csv** extension.

If you selected the wrong delimiter, the program will usually import the data, but it will fail to distinguish between different columns of the data set, and possibly assume that the entire dataset only contains one column. This should be obvious from the texts displayed by the program. In this case, simply import the data again, using the correct delimiter.

If you see the error message displayed in figure 3.2, this means that some cells of your data set probably contain newline symbols and/or carriage return symbols that need to be removed before importing data (see section 2.2).

Figure 3.2: Data import error report.


Event_Coder_qt5

ERROR: Import cancelled

Unmatched quotes (") were found in one of the lines of the file.

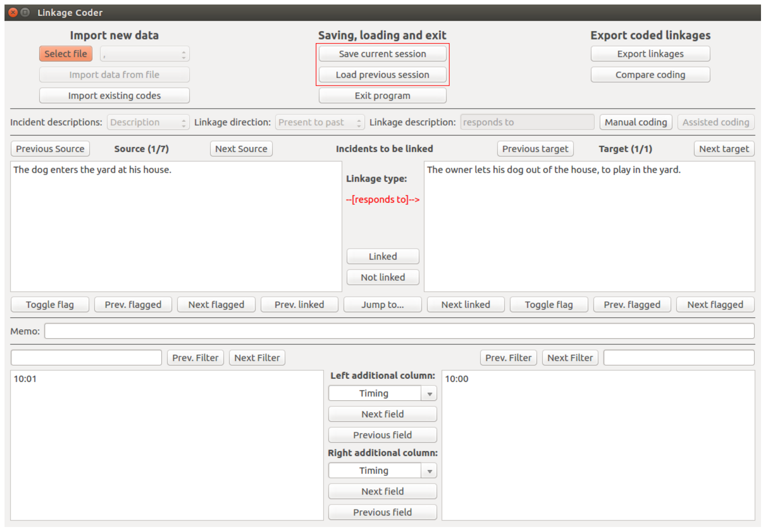
If you are certain that you made no mistakes in one of these points, and you still encounter problems when importing your data set, then you may have encountered a bug in the program that needs to be fixed. In that case, please get in touch with me (see chapter 5).

3.2 Saving and loading data

Coding a data set typically will take a long time, which is why the program allows you to save your progress, and to load the saved session at another moment. Saving data can be done by clicking the **Save current session** button (see figure 3.3). A file dialog will appear, asking you to select a location to store the file, as well as a name for the file. The files will always be saved with the ".sav" extension.

If you want to load a previously stored session, click the **Load previous session** button (see figure 3.3). A file dialog will appear, allowing you to navigate to, and select the file that you wish to load.

Figure 3.3: Saving and loading files.



Incident descriptions: Description
Linkage direction: Present to past
Linkage description: responds to

Source (1/7)
Target (1/1)

The dog enters the yard at his house.

Linkage type:
--[responds to]-->

The owner lets his dog out of the house, to play in the yard.

10:01

Left additional column: Timing
Right additional column: Timing

10:00

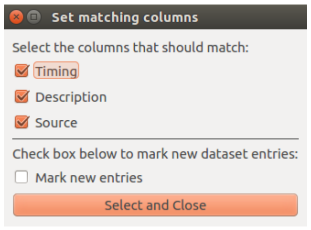
3.3 Importing existing codes

Coding data is typically an iterative process, and it is possible that, during the coding process, the user makes changes to the data set being coded, for example, by adding new rows of data, by adding new columns of data, or by changing the contents of data cells. The program therefore allows the user to import existing codes from an old version of a given data set into a new version of the same data set.

The procedure for importing codes involves the following steps:

1. The user should first make sure that the codes assigned to the **old version** of the data set are stored, by using the **Save session option** (see section 3.2). At a later step, we will import the codes from this save file. For this example, we refer to this file as **Saved_Codes.sav**.

Figure 3.4: Importing codes.



Select the columns that should match:
- [✓] Timing
- [✓] Description
- [✓] Source

Check box below to mark new dataset entries:
- [ ] Mark new entries

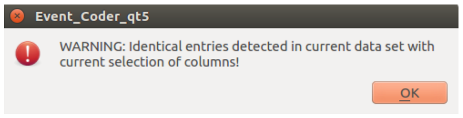
2. After saving the codes assigned to the **old version** of the data set, we can import the **new version** of the data set, using the procedure described in section 3.1. Thus, the steps taken here are the same as when you would start coding a completely new data set.

3. After the **new version** of the dataset has been loaded, you should click the **Import codes** button (see figure 3.4). You will first be shown a warning dialog, just to make sure that you can double check what you are doing. If you select **Ok** in the warning dialog, you will be shown a file dialog. Use this dialog to find the file with your saved codes (**Saved_Codes.sav** in this example). Select and open this file.

4. A new dialog will appear, the specific contents of which will depend on the contents of your data sets. An example is shown in figure 3.4, but yours will probably look different. The dialog will show the columns that the **old version** of your data set and the **new version** of your data set have in common. The program uses these columns to match entries in the **old version** of the data set with entries in the **new version** of the data set. It inspects the data in the selected columns of both files, and it records which rows in the **old version** of the data set are also present in the **new version** of the data set. By default, the program will inspect all columns, but you may want to deselect one or more columns if you decided to make small updates to the contents of these columns. However, for the codes to be imported correctly, the rows of data in both data sets need to be unique. For example, if you set the program to match the data sets using only one column, which has the same contents in multiple rows in one or both of the data sets, then the program will not be able to decide which codes are associated with which row of data. The program will report the error show in figure 3.5, and the import process will be cancelled. If you just added some incidents to your data set, removed incidents from your data set, and/or changed a few existing incidents, then it is usually safest to just have the program match the data sets using all columns. Otherwise, it should be enough to select the column that indicates the timing of the incident, and the column that describes what happened in that incident, because having two incidents with the exact same timing, and the exact same contents (description)

would probably never make sense, because they would describe the same activity, making one of them redundant. Once the program has matched all rows between the two versions of the data set, it will make sure that any linkages that were present in the **old version** of the data set, and that are still valid, will also be present in the **new version** of the data set. See figure 3.6 for a schematic overview that may help to understand the process.

Figure 3.5: Importing codes.



Event_Coder_qt5

WARNING: Identical entries detected in current data set with current selection of columns!

5. You also have the option to have the program mark any new entries in the **new version** of the data set. This means that, while checking the columns you have selected, the program will also remember any rows of data in the **new version** of the data set that it could not match to any rows of data in the **old version** of the data set. These entries will be marked (see 3.6), so that you can easily find them while coding the data.

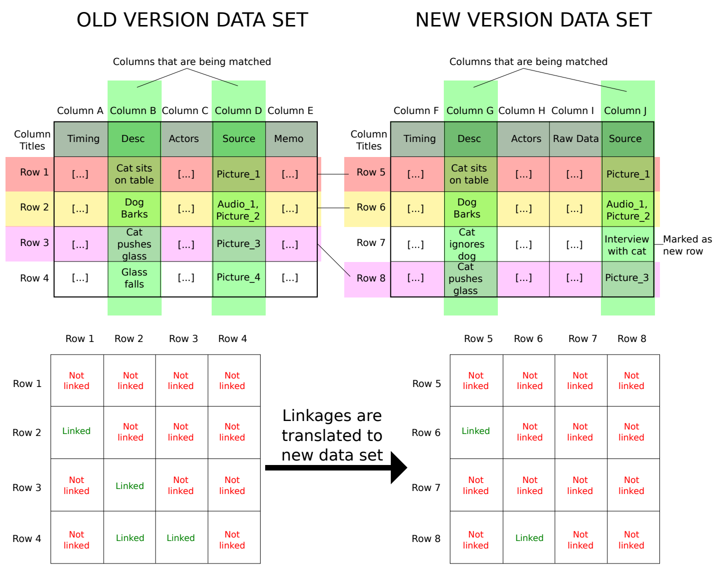
6. If you happened to have changed data in one of the rows of the data set, that is, the row was already present in the **old version** of the data set, but you changed some of the contents of that row of data in the **new version** of the data set (in one of the selected columns), that row of data will be treated as new entry, and any codes assigned to that entry will be lost.

7. The program should now return to the main screen, and any linkages that were present in the save file that you loaded should now also be present in the current session.

Problems when importing codes

There are several things that could go wrong when importing codes from an old data set into a new one. One problem would occur if you do not select any columns to be imported in the appropriate dialog. In this case the program will simply report that no columns were selected, and it will take no further action. Another similar problem would occur if you try to import codes from a data set that has no columns in common with the data set that is loaded into your current session. In this case the dialog where columns can be selected will be empty (except for the option to mark new entries). If you try to proceed, the program will behave as if no columns were selected (see above), report an error, and take no further action.

Figure 3.6: Importing codes.



Another problem may be that some of the codes assigned to entries in the **old version** of the data set do not get imported into the **new version** of the data set, even though these entries appear in both. This should only happen if something changed in the contents of these entries (and only in the contents of those columns that the program tries to match). Even if you made only small changes in the contents of the selected columns, the program will treat the corresponding entry as a new, uncoded entry.

3.4 Starting a new session

Once you have imported data, you are required to set a few important settings that will be used during the coding process. **These settings cannot be changed after the coding process has started, so you have to think carefully in advance what the settings should be**. What these settings should be will depend on your research questions, your theoretical perspective and/or your research design.

Figure 3.7 shows where the settings are located. The left most drop-down menu (**Column with incident descriptions**) requires you to select the column in your data set that contains the descriptions of your incidents, which are assumed to be the main source of information for identifying linkages between incidents (although additional information can be displayed - see section 3.5). In the second drop-down menu (**Linkage direction**) you should select which direction the linkage you wish to code for is pointing in. There are two options to choose from:

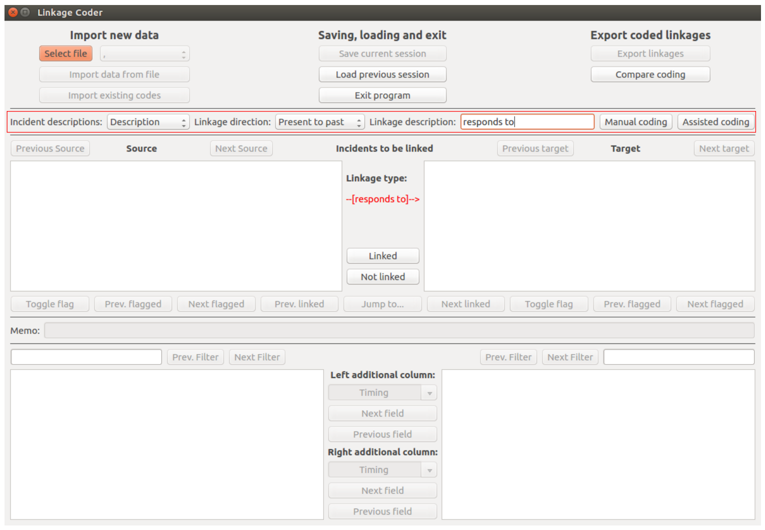
1. *Present to past*: Linkages point from incidents in the "present" to incidents that lie in their "past". For example, if we have two incidents (A and B), and incident B happened after incident A, then there can only be a linkage from incident B to incident A, and not the other way around ($A \leftarrow B$).

2. *Past to present*: Linkages point from incidents that happened in the "past" to incidents in the "present". For example, if we have two incidents (A and B), and incident B happened after incident A, then there can only be a linkage from incident A to incident B, and not the other way around ($A \rightarrow B$).

Finally, a brief description (or label) of the linkage should be given. This description will be shown throughout the coding process, to remind the coder what type of linkage is being considered.

If all these settings have been chosen, then the **Manual coding** and **Assisted coding** buttons will be enabled. Clicking one of these buttons will start the coding process in one of the two available coding modes (the differences between the two coding modes are described in section 3.7). **After this, if the user wants to change any settings with regard to relationship type and direction (s)he will have to restart the entire process.**

Figure 3.7: Settings for new sessions.



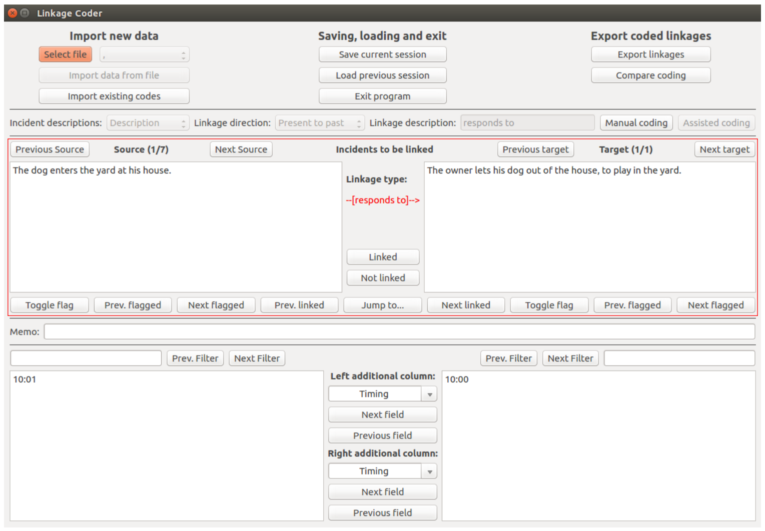
Incident descriptions: Description
Linkage direction: Present to past
Linkage description: responds to
Linkage type: --[responds to]-->

3.5 Navigating through the data

The program, to some extent, enforces a specific way to walk through your data set. As explained in section 2.2, the program assumes that your data are chronologically ordered, with the incidents that occurred earliest at the top, and the incidents that occurred latest at the bottom. When you start coding a new data set, the program will always assume that you wish to start the coding process with the earliest incidents. The program will always present to you the descriptions of two incidents (see figure 3.8).

In the left text field, the program will show the description of the incident that is the **source** of the potential linkage, and in the right text field the program will show the description of the incident that is the **target** of the potential linkage. In the example shown in figure 3.8 we have chosen for a linkage type that points from the present to the past. Thus, when we start coding, in the left field we see the incident that comes second in the chronological order of the data set that we loaded, and in the right field we see the incident that comes first in the chronological order of the data set.

Figure 3.8: Overview of incidents section.



Incident descriptions: Description
Linkage direction: Present to past
Linkage description: responds to

The dog enters the yard at his house.

Linkage type:
--[responds to]-->

The owner lets his dog out of the house, to play in the yard.

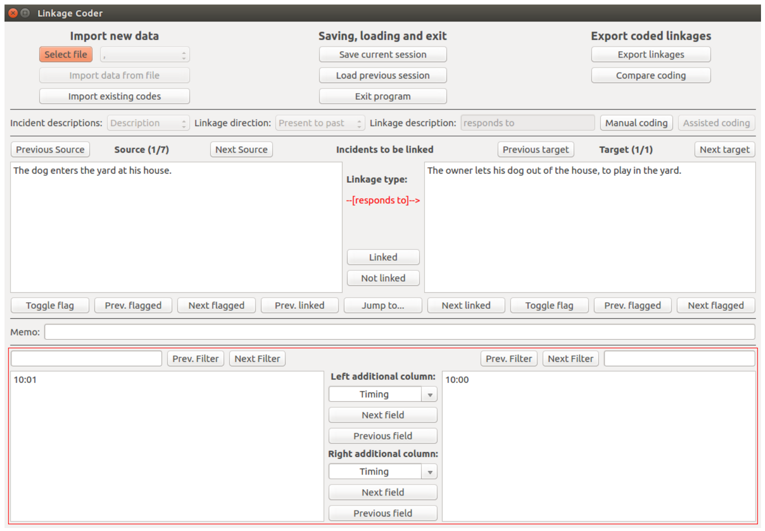
10:01

Left additional column: Timing
Right additional column: Timing

10:00

You will see an extra pair of fields in the bottom of the screen (see figure 3.9), which can be used to display additional information on the incidents that are currently being shown to the user. The left field shows information on the current **source** incident, and the right field shows information on the current **target** incident. The extra fields to be shown can be selected using the controls that can be found in between the two extra fields. The fields can either be selected with the **Next field** and **Previous field** buttons, or with the drop-down menus.

Figure 3.9: Extra fields.



The dog enters the yard at his house.

The owner lets his dog out of the house, to play in the yard.

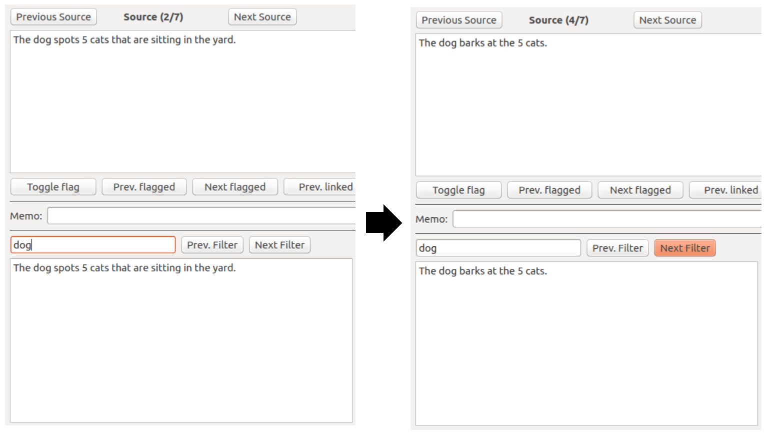
10:01

10:00

Above the extra fields, you will also find two smaller text fields, accompanied by two **Prev. Filter** and **Next Filter** buttons. These fields and buttons allow you to navigate to other **source** or **target** incidents based on the contents of their data columns. If you enter text in the left field, and then click the **Prev. Filter** or **Next Filter** button, then the program will navigate to the previous/next incident that includes that text in the currently selected column. For example, if we set the **left additional column** to the column with incident descriptions, and we type the word "dog" in the left field, then we can quickly navigate to other incidents that have this word in the same column (see figure 3.10). The same can be done with the **target** incidents by using the corresponding field and navigation buttons. These controls can be useful when coding larger data sets: Rather than exploring all pairs potential **source** and **target** incidents (which will increase exponentially!), it may be more practical to identify potentially linked pairs of incidents by searching for key words in their descriptions.

Typically, you will want to consider assigning a linkage between the current pair of incidents, and then move on to another pair. Indications of the incidents' place in the chronological order of the data set are shown with indexes located above the two main fields (see figure 3.11). The indexes are formatted as **([Current source(target)] / [Total possible sources(targets)])**. The total number of possible **source** incidents is always the total number of incidents in the data set minus one. The total number of possible **target** incidents indicated in the index depends on the currently selected index of the **source** incident. For example, in figure 3.11, we have set the linkage direction to "Present to past". The currently selected **source** incident comes second in

Figure 3.10: Jumping to the next incident with the word “dog” in the incident description.



The dog spots 5 cats that are sitting in the yard.

dog

The dog spots 5 cats that are sitting in the yard.

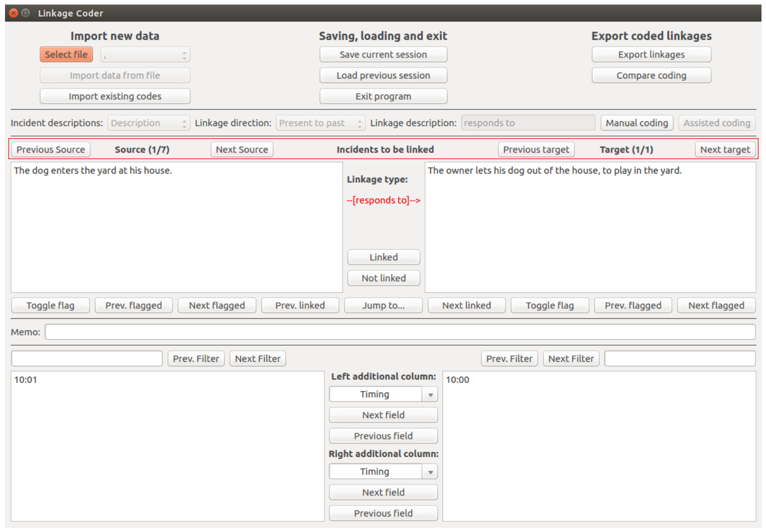
The dog barks at the 5 cats.

dog

The dog barks at the 5 cats.

the chronological order of the complete data set, but the index is shown as (1/7), because the first incident in the data set cannot have a linkage to any other incident in the data set, given the currently selected linkage direction. The currently selected **target** incident comes first in the chronological order of the complete data set. Its index is shown as (1/1), because this incident is the only incident in the data set that the currently selected **source** incident could have a linkage to.

Figure 3.11: The location of the incident indexes.



Incident descriptions: Description

Linkage direction: Present to past

Linkage description: responds to

The dog enters the yard at his house.

--[responds to]-->

The owner lets his dog out of the house, to play in the yard.

Left additional column: Timing

Right additional column: Timing

10:01

10:00

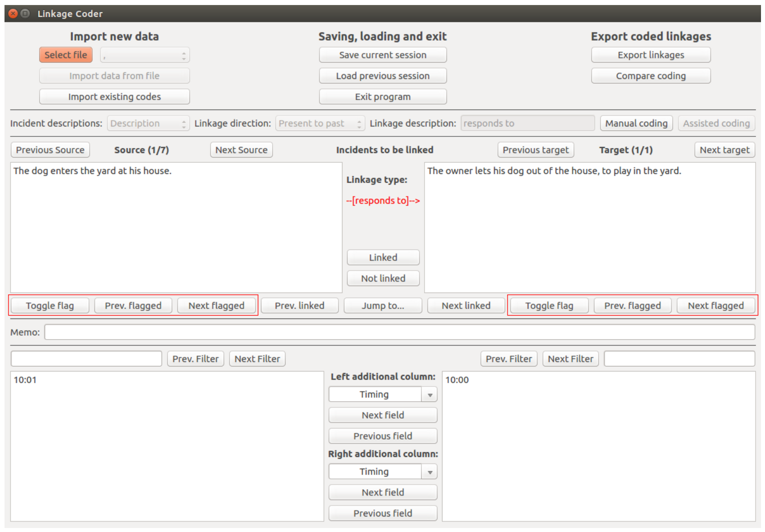
Although the **Assisted coding** mode will help the user navigate through the data set (see 3.7), it is always possible to move to other incidents manually, using one of the navigation buttons, which are located to the left and right of the incident indexes (see figure 3.11). The current **source** incident can be changed with the **Next source** and **Previous source** buttons, and the current **target** incident can be changed with the **Next target** and **Previous target** buttons.

Below the two main fields you will find a collection of additional navigation buttons. In the middle you will find a button named **Jump to**. Clicking this button will open a small dialog that allows you to type index numbers for a **source** incident and a **target** incident (both always have to be entered). If you then click the **Go** button, the program will jump to the incidents with the selected indexes. If you select an "illegal" index for the **source** incident, nothing will happen. If you select a "legal" index for the **source** incident, but an "illegal" index for the target incident, then the program will jump to the selected **source** incident, and to the first available **target** incident.

3.6 Marking incidents

In the area demarcated in figure 3.12, you will find several buttons that refer to flagged incidents (**Toggle flag**, **Prev. flagged**, **Next flagged**). Flags can be used to mark incidents that you would like to return to later. If you click the LEFT **Toggle flag** button, an exclamation mark will appear next to the index of the **source** incident to show that this incident is currently flagged (see figure 3.13). The same can be achieved for any **target** incident, using the **Toggle flag** button on the right.

Figure 3.12: Options related to flagging incidents.


Linkage Coder

Incident descriptions: Description
Linkage direction: Present to past
Linkage description: responds to

Source (1/7)
The dog enters the yard at his house.

Linkage type: --[responds to]-->

Target (1/1)
The owner lets his dog out of the house, to play in the yard.

Memo:

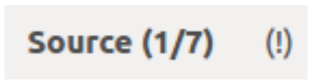
Left additional column: Timing
Right additional column: Timing

10:01

10:00

If you later want to return to a flagged **source** incident or a flagged **target** incident, you can use the corresponding **Prev. flagged** or the **Next flagged** buttons. These buttons work similar to the **Prev. source/target** and the **Next source/target** buttons, but they will skip all incidents that are not flagged. This should allow you to relatively easily find incidents that you flagged earlier.

Figure 3.13: Indication of flagged incident.



Automatically flagging new incidents when importing codes

As described in section 3.3, when you import codes from a previously stored session into a new version of a data set, you will also have the option to mark any new incidents. This means that the program will automatically flag all incidents that are in the **new version** of the data set that you are currently coding, but that were not in the **old version** of the data set from which you are importing the codes. This allows you to always easily find the incidents that have not been considered for coding before.

3.7 Coding linkages

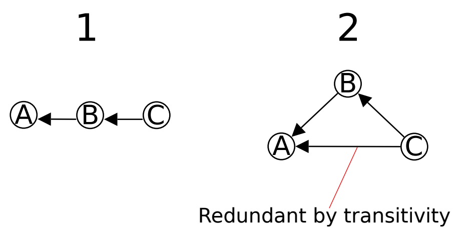
Indeed, the main purpose of the program is to code for linkages between the incident. There are two different modes that the user can use for this: **Manual coding** and **Assisted coding**. The differences between these two modes are most easily explained by describing how the **Assisted coding** mode actually assists the user.

The logic underlying the **Assisted coding** mode is based on ideas from David Heise [1]. Heise discusses a practical problem that occurs when trying to identify relationships between a larger number of incidents: The number of relationships to consider increases exponentially with an increase in the number of incidents. We can deal with this problem by assuming that the linkages between incidents are transitive, and that this makes some linkages redundant. Consider an example in which we have three incidents (A, B, and C), where the order of occurrence is the same as their alphabetical order, and where B occurred in response to A and C occurred in response to B (see figure 3.14.1).

If incident C also occurred in response to incident A the user might normally want to add a linkage between them (see figure 3.14.2). However, if we assume that the linkages between incidents are transitive, then this third linkage

[1]See especially Heise, D. R. (1989). Modeling event structures. *The Journal of Mathematical Sociology*, 14(2), 139âĂŞ169.

Figure 3.14: A simple chain of incidents.



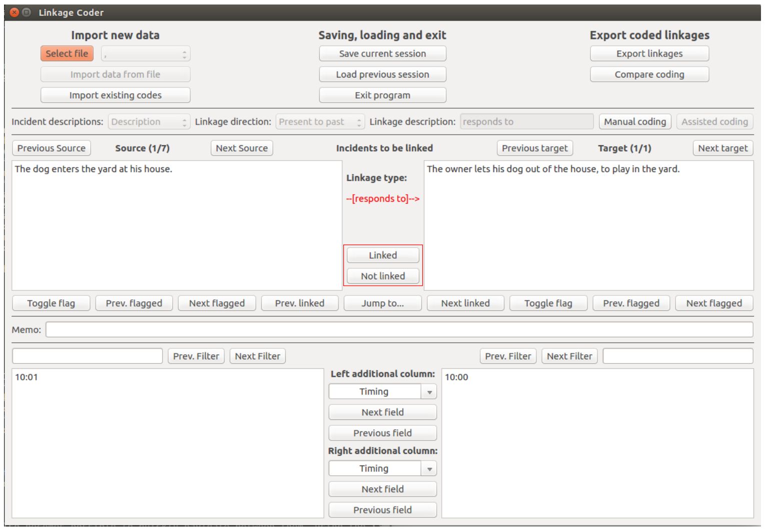
could be considered redundant, as the first situation already suggests that C responded to A indirectly, through B. In other words, in this case we say that if some incident (C) responds to some other incident (B) that is the last in one or more chains of incidents ($A \leftarrow B$), then we say that the incident (C) also responds indirectly to incidents that occurred earlier in those chains (A). Indeed, this is quite a strong assumption to make for some types of relationships between incidents (including the one considered in the example used here). However, if we work with this assumption, it can greatly reduce the number of linkages that we need to consider during the coding process. In the example above, if we already know that incident B occurred in response to incident A, and that incident C occurred in response to incident B, then there is no need for us to consider the linkage between C, since it is already present in its transitive form.

The **Assisted coding** mode helps the user by figuring out which linkages are redundant. When the user starts the coding process in this mode, the user is required to explicitly indicate, for each pair of incidents that the user is presented, whether it is linked or not linked, using the corresponding buttons (see figure 3.15).

The program always assumes that the user works her/his way through the data set from the first pair of incidents to the last. If the user has decided whether or not the initial pair of incidents is linked by clicking the corresponding button, then the program calculates which linkages are redundant according to the new situation, and then jump to the next non-redundant pair after a short pause. In other words, all pairs of incidents that are already linked transitively will be skipped. This means that, in the **Assisted coding** mode, the user can code the entire data set without ever having to use the navigation buttons (see 3.5). However, manual navigation is also possible in the **Assisted coding** mode, by using the appropriate navigation buttons[2]. Also, if you feel that you have made a mistake somewhere, you can jump between linked pairs of incidents, using the **Prev. linked** and **Next linked** buttons (see figure 3.16).

[2]It is very important to remember that when you decide to navigate to some pair of incidents manually, and then use **Linked** or **Not linked** buttons, the **Assisted mode** will simply pick up from there, and assume that any pairs of incidents that you have skipped manually are not linked.

Figure 3.15: Location of the linking buttons.



Incident descriptions: Description
Linkage direction: Present to past
Linkage description: responds to

The dog enters the yard at his house.

--[responds to]-->

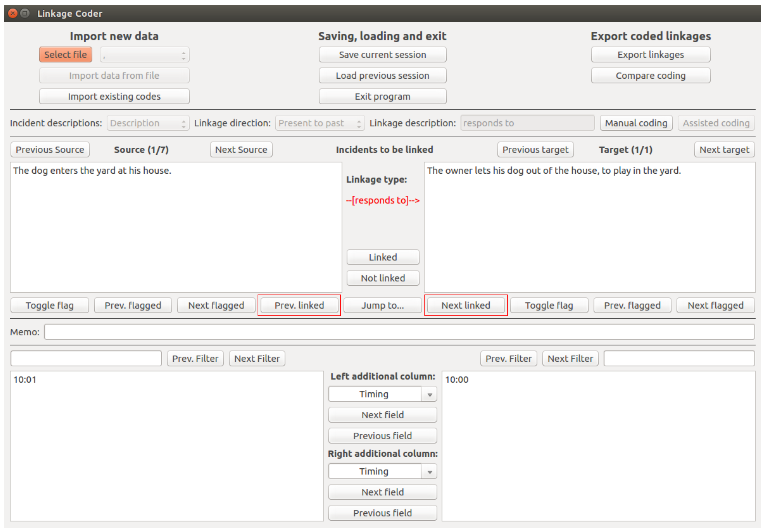
The owner lets his dog out of the house, to play in the yard.

10:01

Left additional column: Timing
Right additional column: Timing

10:00

Figure 3.16: Navigating based on linkages.



Incident descriptions: Description
Linkage direction: Present to past
Linkage description: responds to

The dog enters the yard at his house.

--[responds to]-->

The owner lets his dog out of the house, to play in the yard.

10:01

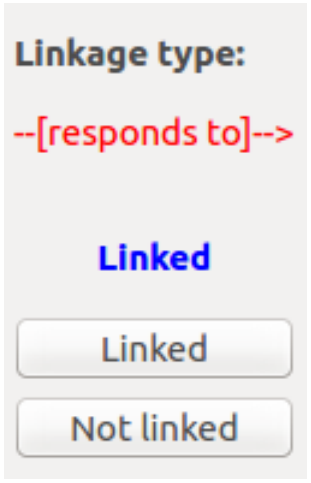
Left additional column: Timing
Right additional column: Timing

10:00

The **Manual coding** mode does not automatically jump between pairs of incidents, and thus requires the user to navigate the data her/himself. If the data set is small enough, and the user prefers to consider all the pairs of incidents, then this is the mode that should be used. It is always possible to switch between **Manual coding** and **Assisted coding** by clicking the corresponding button.

As discussed above the user can indicate where or not a pair of incident is linked with the **Linked** and **Not linked** buttons (see figure 3.15). If the user clicks the **Link** button, an indication of the link will appear on the screen (see figure 3.17). if the user clicks the **Not linked** button, the linkage between the current pair of incidents will be removed in case it is present.

Figure 3.17: Indication that incidents are linked.



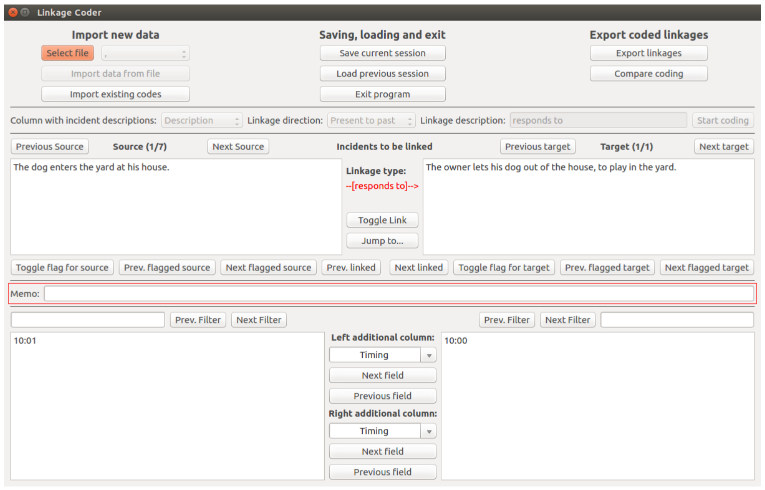
Below the two main text fields you will find another text field called **Memo** (see figure 3.18). This field can be used to record any memos that you would like to add to the current pair of incidents. Whatever you type here will be instantly saved. If you want to remove a memo, simply delete the text. When you export your coded data (see 3.8), the memos will also be included with all linkages that you have created (i.e., memos for pairs of incidents that have no linkage will not be exported).

3.8 Exporting data

Once you have coded a data set, and you want to export the results, you can click the **Export linkages** button. This will open a new dialog, where you can select several export options (see figure 3.19).

The dialog allows you to assign properties to your incidents. These properties are simply the contents of the various columns of the data set that you imported. In fact, if you select all the properties, using the corresponding tick boxes, one of the files exported will simply be a near-identical copy of the original data set that you imported, the only difference being that every incident will be assigned a unique ID, numbered from 1 to N, where N is the total number of incidents in your data set. These incidents, and any properties that you select for them, will be written to a file that is called **"Linkages_Nodes.csv"**. As the name of the file suggests, this is a node list, which in this case is structured

Figure 3.18: Location of memo field.



Column with incident descriptions: Description

Linkage direction: Present to past

Linkage description: responds to

The dog enters the yard at his house.

Linkage type:
--[responds to]-->

The owner lets his dog out of the house, to play in the yard.

Memo:

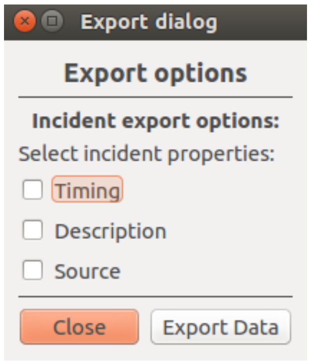
10:01

Left additional column: Timing

Right additional column: Timing

10:00

Figure 3.19: The export dialog.



such that it can be immediately imported into Gephi, my favourite network visualisation program[3].

The program will also export a file called **"Linkages_Edges.csv"**, which contains all the linkages between incidents that you have identified, as well as any memos that you might have assigned to these linkages. A third file that the program exports is a file called **CYPHER.txt**, which is a list of Cypher commands, which can be used to import your coded data into a neo4j database. To understand more about what you could possibly do with these files, please read chapter 4.

[3]See https://gephi.org. I offer no instructions on how to import data into Gephi here. Various guides for that are available elsewhere. Indeed, the data can also be imported into other software, although you might have to make small changes for that to work.

3.9 Log files and exiting the program

It will probably come as no surprise to you that the default way to close the program is to click the **Exit program** button. Alternatively, you could close the dialog of the program in the same way you would close any dialog in your OS.

Just before the program closes, it will export a log file to the **"logs/"** folder that is located in the folder from which you run the program (if this folder does not exist yet, the program will create it). The log file will be time stamped with the date and time at which it was created. In the log file, you will find a lot of details of various operations that you have performed while using the program, such as navigating to other incidents, creating or removing linkages between incidents, and several other things. These logs are created silently and automatically.

The logs are designed to help you make the coding process more transparent. For example, they may help you retrace your steps if at some point in the coding process you run into some kind of problem, and do not remember how you got there. I do not expect that anyone will be terribly interested to ever inspect these log files, but it may be reassuring to know that they are there if you need them.

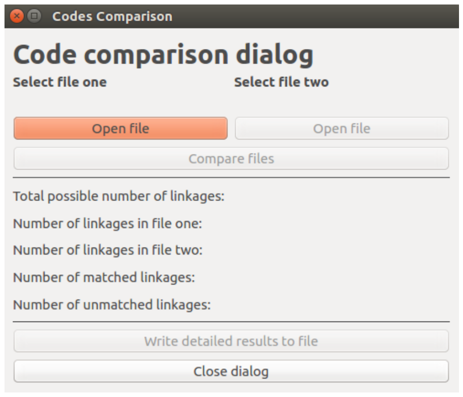
3.10 Comparing codes

The ***Linkage Coder*** program also allows you to do some basic inter-coder reliability tests. In the top right of the screen, you will find a button named **Compare coding**. This will open the dialog visualised in figure 3.20. The dialog requires you to select two different save files that include the codes that you wish to compare. These can be selected by clicking the **Open file** buttons, which will open file dialogs with which the save files can be selected. The names of the files that are currently selected will be shown in the dialog as well (see figure 3.21).

After selecting two files, one can compare the codes in the files by clicking the **Compare files** button. The results of the comparison are then shown in the dialog, as illustrated in figure 3.21. The dialog will report (1) how many linkages could in theory exist (if all pairs of incidents considered would be assigned a linkage, (2) the number of linkages that have been coded in file 1, (3) the number of linkages that have been coded in file 2, (4) the number of linkages that match between the two files, and (5) and the number of linkages that do not match between the two files.

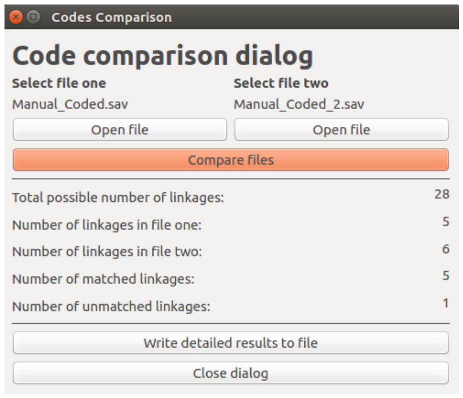
The last two items should give you a clue of how closely the coding of the two files matches. If you find that there are unmatched codes, a text file with a more detailed report can be exported using the **Write detailed results to file** button. This file contains additional information that can be used to identify the matched and unmatched linkages in both files. For an example of

Figure 3.20: The code comparison dialog.



Total possible number of linkages:
Number of linkages in file one:
Number of linkages in file two:
Number of matched linkages:
Number of unmatched linkages:

Figure 3.21: Displaying results of comparison.

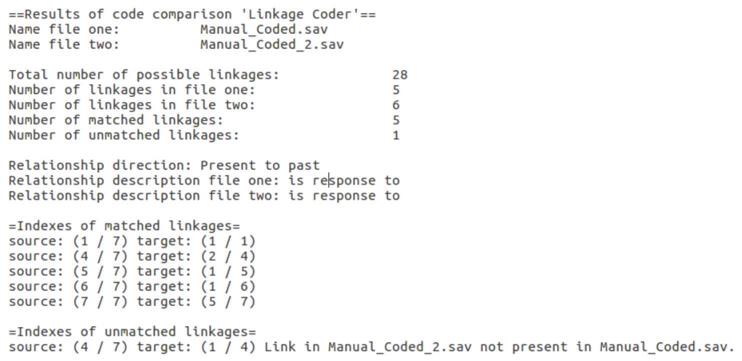


Manual_Coded.sav
Manual_Coded_2.sav

Total possible number of linkages:	28
Number of linkages in file one:	5
Number of linkages in file two:	6
Number of matched linkages:	5
Number of unmatched linkages:	1

the kind of information provided, see the screenshot in figure 3.22. This file can be useful in resolving any differences in the coding of two different coders.

Figure 3.22: Screenshot of file with details of code comparison.



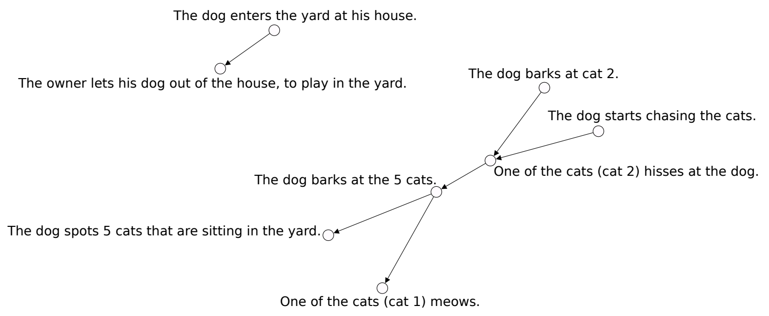
```
==Results of code comparison 'Linkage Coder'==
Name file one:          Manual_Coded.sav
Name file two:          Manual_Coded_2.sav

Total number of possible linkages:              28
Number of linkages in file one:                 5
Number of linkages in file two:                 6
Number of matched linkages:                     5
Number of unmatched linkages:                   1

Relationship direction: Present to past
Relationship description file one: is response to
Relationship description file two: is response to

=Indexes of matched linkages=
source: (1 / 7) target: (1 / 1)
source: (4 / 7) target: (2 / 4)
source: (5 / 7) target: (1 / 5)
source: (6 / 7) target: (1 / 6)
source: (7 / 7) target: (5 / 7)

=Indexes of unmatched linkages=
source: (4 / 7) target: (1 / 4) Link in Manual_Coded_2.sav not present in Manual_Coded.sav.
```

Chapter 4

What is next?

After you have coded your data (see section 3.7), and exported your files (see section 3.8), you will end up with three files in your **"export/"** folder. What can you do with these files now? The node list and the edge list that are exported are designed such that they can be immediately imported into the data laboratory of Gephi[1]. If you have the *event layout plugin* for Gephi installed[2], then you will be able to quickly create a basic event graph in Gephi (see section 4.1 for more details). In event graphs nodes represent events, and the edges represent relationships between events[3]. Figure 4.1 visualises an example of a basic event graph that can be created in this way. Creating simple event graphs may already help researchers in the further interpretation of their data. However, I believe a much more powerful way to work with the coded data is to use them to create a graph database.

Figure 4.1: Screenshot of basic event graph.


The dog enters the yard at his house.
The owner lets his dog out of the house, to play in the yard.
The dog barks at cat 2.
The dog starts chasing the cats.
One of the cats (cat 2) hisses at the dog.
The dog barks at the 5 cats.
The dog spots 5 cats that are sitting in the yard.
One of the cats (cat 1) meows.

[1]See https://gephi.org/.

[2]See http://www.wouterspekkink.org/?page_id=93.

[3]For an example of a publication in which event graphs are used, see Spekkink, W. A. H., & Boons, F. A. A. (2016). The Emergence of Collaborations. *Journal of Public Administration Research and Theory*, 26(4), 613âĂŞ630.

4.1 Graph databases and event graphs

At the time of writing this manual, I am experimenting with the use of graph databases, which are, in very simple terms, databases in which all data are either stored as **nodes**, as **relationships between nodes**, or as **properties** of nodes or relationships. For me, storing data this way is ideal, because I usually like to present event data in some "networked form", and with graph databases there are very few steps required between storing data and presenting data in such a form. As discussed in section 1.2, I intend to integrate the ***Linkage Coder*** program in a bigger program at some point. My intention is for this bigger program to interface directly with a graph database, without the user ever having to worry about how this works. For now, however, the user will need to use third-party solutions for creating such a graph database, and import the coded data into the database manually.

The graph database technology that I currently work with myself is the Neo4j Community Edition[4], because (1) it is open source, which is something that I encourage, and (2) because it uses a query language and interface that I find very useful[5]. I will not go into the details of how to use Neo4j here, because that would take me well beyond the scope of this manual. There is a wealth of information that can be found on the website of Neo4j[6], as well as on numerous other online forums. One of the files exported by the ***Event Coder*** program (**"CYPHER.txt"**) is a list of Cypher commands (Cypher is the language used by Neo4j) that the user can simply copy and paste into the Neo4j interface to recreate the complete, coded data set as a Neo4j database.

Figure 4.2 shows a screenshot of a Neo4j database that I created in this way, using some made-up data. In this screenshot, the yellow nodes are our incidents, and the green nodes are linkages between incidents. The relationships between the nodes indicate whether a given incident is a source or a target in the corresponding linkage[7]

We could add other information to the database we created this way. Another program I recently created is the ***Event Coder*** program. The ***Event Coder*** program and the ***Linkage Coder*** program are intimately related. In fact, I plan to integrate both of them in a larger program in the future (see section 1.2). While the ***Linkage Coder*** program specialises in the facilitation of coding linkages between incidents, the ***Event Coder*** program can be used to

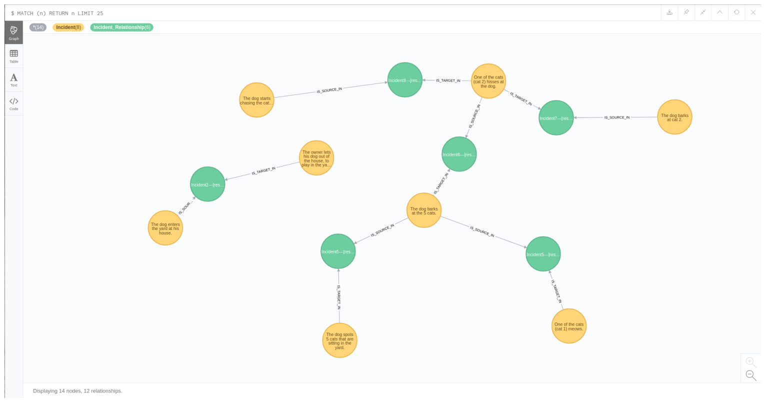
[4]See https://neo4j.com/.

[5]I also see important drawbacks to Neo4j. In my opinion the configuration of Neo4j on your system, and the creation and management of new databases could have been made more intuitive. If you are completely new to Neo4j, it might take a while for you to get a solid grasp of how it works.

[6]See, for example, https://neo4j.com/docs/.

[7]This is a somewhat inefficient way of representing the data, because the linkages between incidents could be represented more efficiently by storing them as relationships, without creating separate nodes for linkages. However, by treating the linkages as nodes, this would keep open the possibility to link these nodes to yet other entities in the database, which is something I may want to do in future versions of my programs.

Figure 4.2: A coded version of our fictional data set stored in a graph database.



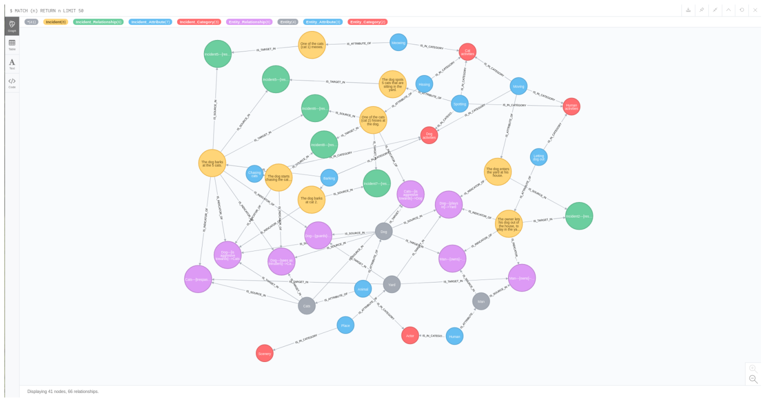
```
$ MATCH (n) RETURN n LIMIT 25
```

Displaying 14 nodes, 12 relationships.

assign attributes to incidents, and to identify networks of relationships that are indicated by the occurrence of incidents. In figure 4.3 we added additional information to our database, based on codes assigned to our data set with the ***Event Coder*** program.

Once we have all this information in our graph database, it is easy to export files that we can use to create event graphs in Gephi, but now we can also include additional information about our incidents in the event graph. Figure 4.4 shows Cypher commands that we can use to export a node list and an edge list from our database, where we also include attributes that were assigned to our nodes (if more than one attribute was assigned per incident, you would need a more complex command). Figure 4.5 shows an event graph that we could create with the information contained in these files. In this case we chose to highlight which types of activities occurred in response to each other.

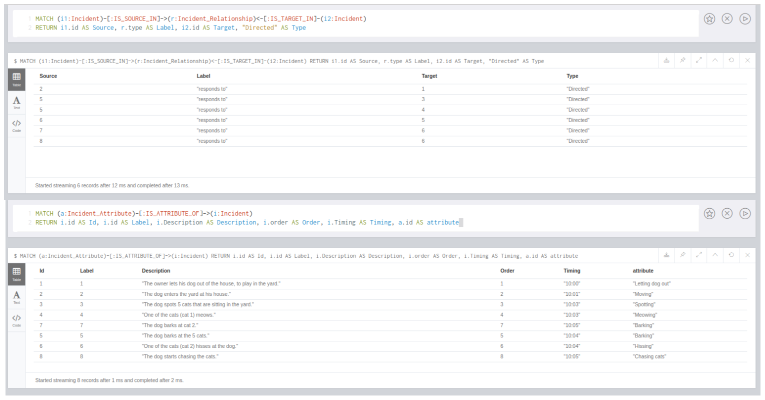
Figure 4.3: The graph database expanded with additional details.



```
$ MATCH (n) RETURN n LIMIT 50
```

There are many other possibilities with the Cypher query language. This does

Figure 4.4: Exporting our node and edge lists from the Neo4j database.



```
MATCH (i1:Incident)-[:IS_SOURCE_IN]->(r:Incident_Relationship)<-[:IS_TARGET_IN]-(i2:Incident)
RETURN i1.id AS Source, r.type AS Label, i2.id AS Target, "Directed" AS Type
```

```
$ MATCH (i1:Incident)-[:IS_SOURCE_IN]->(r:Incident_Relationship)<-[:IS_TARGET_IN]-(i2:Incident) RETURN i1.id AS Source, r.type AS Label, i2.id AS Target, "Directed" AS Type
```

Source	Label	Target	Type
2	"responds to"	1	"Directed"
5	"responds to"	3	"Directed"
5	"responds to"	4	"Directed"
6	"responds to"	5	"Directed"
7	"responds to"	6	"Directed"
8	"responds to"	6	"Directed"

Started streaming 6 records after 12 ms and completed after 13 ms.

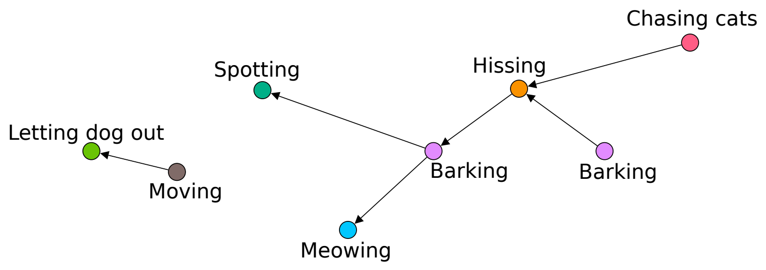
```
MATCH (a:Incident_Attribute)-[:IS_ATTRIBUTE_OF]->(i:Incident)
RETURN i.id AS Id, i.id AS Label, i.Description AS Description, i.order AS Order, i.Timing AS Timing, a.id AS attribute
```

```
$ MATCH (a:Incident_Attribute)-[:IS_ATTRIBUTE_OF]->(i:Incident) RETURN i.id AS Id, i.id AS Label, i.Description AS Description, i.order AS Order, i.Timing AS Timing, a.id AS attribute
```

Id	Label	Description	Order	Timing	attribute
1	1	"The owner lets his dog out of the house, to play in the yard."	1	"10:00"	"Letting dog out"
2	2	"The dog enters the yard at his house."	2	"10:01"	"Moving"
3	3	"The dog spots 5 cats that are sitting in the yard."	3	"10:03"	"Spotting"
4	4	"One of the cats (cat 1) meows."	4	"10:03"	"Meowing"
7	7	"The dog barks at cat 2."	7	"10:05"	"Barking"
5	5	"The dog barks at the 5 cats."	5	"10:04"	"Barking"
6	6	"One of the cats (cat 2) hisses at the dog."	6	"10:04"	"Hissing"
8	8	"The dog starts chasing the cats."	8	"10:05"	"Chasing cats"

Started streaming 8 records after 1 ms and completed after 2 ms.

Figure 4.5: Event graph with incident attributes.



not actually tell you anything that you could do with these data in terms of analysis, but I hope it does make clear that it offers a lot of flexibility in how you store, present, and structure your data. Once you manage to put your data into a graph structure, it is easy to convert your data in the various forms required by different software packages.

Chapter 5

Contact details

At the time of writing this manual, I work as a Research Associate at the Sustainable Consumption Institute (SCI) of the University of Manchester.

Wouter Spekkink
The Sustainable Consumption Institute
The University of Manchester
188 Waterloo Place, Oxford Road
M13 9PL, Manchester
The United Kingdom

Email address (office): wouter.spekkink@manchester.ac.uk
Email address (personal): wouterspekkink@gmail.com

Website: http://www.wouterspekkink.org
Github page (where I upload source code): https://github.com/WouterSpekkink