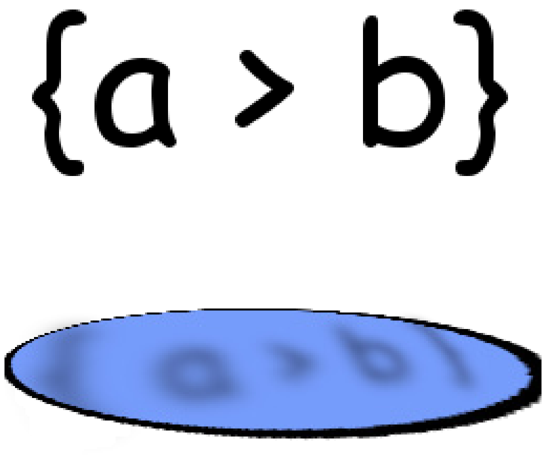
Qualitative Analysis for Beginners
(using TAMS Analyzer)



Published by May Day Softworks
Tacoma, WA

Introduction

This document provides an overview and tutorials for starting to work with qualitative data using TAMS Analyzer (TA).

The first chapter provides a birds eye view of TA and the way it fits into qualitative research. It is a personal account of doing qualitative research and using TA to track data and answer research questions. Other people may use the tools in TA very differently, but this should provide at least one vision of how you *might* use this digital tool with your project. To be clear the research I discuss, in the abstract, assumes a single researcher. TA has multi-user capabilities. Using TA in a multi-user environment is not simple and is discussed in a different document. The point of part one is not to teach you "how" to do anything, merely to outline the ways that TA is used to solve different problems at different stages in a research project.

The second chapter moves to specifics and talks about the way that coding works. Coding is the process by which passages in documents are assigned meanings. It also talks about initially setting up a project. Coding is very flexible in TA. A code (i.e., a "word" indicating what a passage of a text means) can mark zero characters, a few characters, or a whole document.

The third and final chapter of this primer covers analysis, the process of mining a document that has been coded for information. Some of this information may be quantitative (e.g., How many times did I use code X?) but most will be qualitative (What are the examples of codes X, Y, and Z organized by gender of the interviewee?).

These chapters are in no sense to be taken as a comprehensive description of TA or its capabilities. When you have completed this introduction you can read through the user guide and release notes for descriptions of the full extent of TA's abilities.

Living the qualitative life in a digital age…
…with TAMS Analyzer

What does it look like to do qualitative research in a digital age? For many qualitative researchers computers with their number crunching capacities seem anathema as research tools for uncovering textual meanings, cultures as symbol systems, and practices as defined in anthropology, ethnomethodology, cultural studies, and other fields engaged in qualitative studies. Yet computers have been involved in analysis of language since the 1950s and the dawn of artificial intelligence research. There are now numerous programs available for qualitative researchers including NVivo, NUD*IST, MaxQDA, Atlas/ti, HyperResearch, Qualrus, Transana and many others. I developed TAMS Analyzer (TA) because there were no packages oriented towards textual analysis and qualitative research for Macintosh OS X when I desperately needed one in 2002. I have continued to extend and develop the program over the years. Note that I have not used any other qualitative packages, and, as a result, TA is a bit sui generis. However, I did use computers for qualitative research while working on my dissertation in the early 1990s (Weinstein, 1998) I used a programmable word processor called Nisus, an outliner called More for analysis, and Panorama, a very flexible database good at manipulating large chunks of text quickly. Mixing and matching programs was important for me then. In 2002, none of the programs I had used had been updated for the new Macintosh operating system. TA was my answer. In many ways TA's design with open, in-text codes, reflects the heterogeneous way I dealt with data early on. The codes are naked, ugly, visible for anyone to see, and any program to use. It's a virtue and a handicap.

Somewhere I read that unlike quantitative software every qualitative package is different. Skills developed on one package don't easily transfer to others. The language different software programs use to describe their operations varies as well. Qualitative research is rather non-paradigmatic in Kuhn's sense of the word: we don't share a common language/problem model/solution model/world view. That said, when I met with the creator of MaxQDA recently, we were able to compare features and assess our software's various strengths and weaknesses almost instantly. So what I discuss here will be different, at least slightly, from what someone would learn with other software, but there are commonalities as well.

This first chapter introduces how TA works at the most conceptual level, i.e., how it links to the qualitative research process. This chapter will not have any screen shots and few tricks. It is about how to think about using qualitative research tools in the course of a research cycle.

To tell this story I have flattened out the research process. But I think this pattern and growth in the research cycle will be recognizable to those that have done qualitative studies with TA and probably similar packages.

I. **Phase 1: In the field and on the computer**

Most qualitative data sets are defined by 3 types of objects: interviews, observations, and artifacts. While these can't be considered comprehensive (consider surveys, for instance), these forms of data still dominate those fields engaged in qualitative research projects. Obviously, depending on the question being asked, the balance of these types of objects will vary. For the studies I've recently become involved in, interviews have often provided the bulk of data, earlier work was more observational.

Digitization starts in the field. Field notes are often typed straight into laptops these days. Digital video is increasingly what is analyzed rather than the observation notes themselves. TA is a text oriented package. This doesn't mean that I can't use other forms of data like digital audio and video, but that they are not analyzed directly. Instead they are linked to a log or transcript using time codes and it is the log/transcript that is coded and analyzed. TA is very smart about this and can use the time codes to leap to the important spot in the video or the other direction (move the scrub bar and leap to the nearest time code in the log).

For my projects I've been "taping" my interviews using an Olympus™ DM-10 digital recorder. I'm a real convert to this equipment. The sound quality, if I use the highest setting, is crystal clear. It's also smaller and quieter (100% quiet) than the old recorders, even those little micro-cassette ones. Once the interview is over I can connect the recorder to my computer (standard USB cable) and drag the file to my Mac. Now this is a bit technical but I feel the need to share: the only hitch with what I've said is that the DM-10 saves the files in WMA format (Microsoft™) which is only readable by Microsoft programs. Luckily there is a tool called MPlayer which is free and which if used with the right settings converts this to WAV that then can be turned into an MP3 file using iTunes.[1] Both of the last two formats are TA friendly, it's just that WAV files are huge, and I have no audio complaints about the MP3s which are 1/10 the size.

Writing field notes can be done straight into TA. Document-artifacts need to be converted to RTF format to be useful. Microsoft™ Word is the best way to do this, though other conversion tools exist. As for multimedia files they need to be converted to some format that Quicktime™ supports (AIFF, WAV, MP3, etc.).

The first phase of my projects involve frequent scene shifts between "the field" and the computer. At this point in the research cycle I'm using TA to transcribe interviews—TA has a little transcription machine built into each document window—or to make logs of video. I've also entered any codes, i.e., words or short phrases that name themes, up front. I'm also adding codes as I see themes emerge while transcribing.

There is another type of decision I'm making as I enter codes at this point. This concerns the types of independent variables that I will use later on in analysis. These include who is being interviewed/observed, the times of particular observations or points in the interview (aka time codes), SES/geography/gender/race or other sociological factors I'm interested in. Some of these should be done as data is entered (the time codes for instance). It's just easier that way, and TA has some tools that automate the process. Others I can enter later, though if I know that a particular type of data is going to be of interest, I can enter it as I go along. TA has ways of saying 'in this file the gender is "male"' or 'the city is "NY"' etc, so I don't have to mark every instance of "coded data" with that information. I can also automate this information by telling TA that if "Bob" is the speaker he is from NYC, is male, Latino, and bald. Any time something

1 I have been using the command line version of MPlayer which is available through the Fink project. I type "mplayer -ao pcm myfile.wma" and a wav file is generated from the wma.

is identified as from "Bob" that other information will be entered automatically. The types of information I am talking about here I call context codes, since they provide information about the context for the data (they describe who the speaker is more than what s/he said).

This identifying of context and data codes continues until I have collected the body of interviews and observations in my data set, or until the interviews and observations are not generating new themes. Then it's time to go deep.

II. **Phase 2: Going deep**

Phase 2 begins by reading and rereading my data and checking and applying the themes throughout the database. In this package, and most others like it, codes/themes are attached to portions of the transcript/log. This chunk of text is an example of this theme, etc. I am adding themes, and every time I do, I need to return to every document to see where those themes apply. While not as physically taxing as transcription, this is in some ways the most tedious work in a qualitative study. The work ends when I've stopped adding themes.

I've found two tools in TA are absolutely essential in this part of the process. The first is the definition button. When I tell TA about a new theme I also define it. Lists of these themes appear in just about every window TA displays and researchers can always click a particular theme and hit the definition button to see how the theme was defined.

This is usually not enough, however, as more code/themes are added. The definition button has to be supplemented by searches for particular themes/codes. In other words I need to go back and see each instance where I've used a code before so that I know what I meant by the code. "Searches" return a window that lets me page through each instance of the code applied to my interviews, fieldnotes, etc. Reading the passages I've coded a particular way allows me to inductively recall the meaning of the code. For complex themes, more than the definition, this gives me clues as to tone/topic/theme the code was intended to identify.

Once I have coded all of my documents and identified the major themes it's time to deepen those themes. Given a theme like "negative behavior" (TA actually doesn't allow spaces in code names, so it would probably be negativeBehvior or negative_behavior), I would want to further classify each passage as to the type of negative behavior. This part of the process I call reanalysis. Reanalysis involves looking at the instances of a particular code to alter my original coding scheme. To do a reanalysis, I do a search for the code I am interested in breaking down and then lock out most of my editing capabilities by throwing the software in "reanalysis mode. This will keep me from doing damage to my source files that would throw them out of synch with my search results. Next, I'll sort through the results of that search and try to identify the key types of "negativeBehavior". If one type is name calling then I will "mark" the examples of name calling and "recode" them as "negativeBehavior>nameCalling". The ">" is TA's way of indicating that nameCalling is a subtype or subcode of negativeBehavior. I can add subcodes to subcodes by just naming my codes with additional levels of the ">" sign. I would then continue marking the next type of negative behavior (throwingSpitWads) and doing the same thing. Eventually "negativeBehavior" turns into a whole family of types of "negativeBehavior" all under a top level with that name.

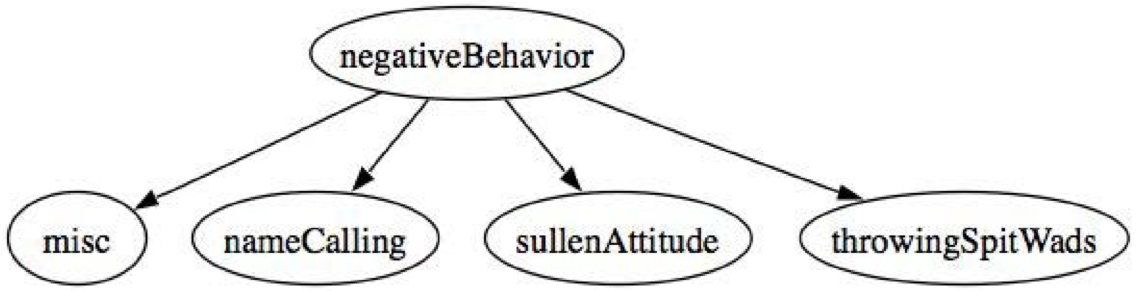


Figure 1: A "graph" of the negativeBehavior family

The real challenge here is to try to categorize each instance that I come across of my top level code of interest. Usually some are too vague to be classified and are thrown into miscellaneous subcode (negativeBehavior>misc), but I want to limit the number of these, if possible.

For some projects this may be enough. I have now created a typology for the themes I'm studying. By repeated reanalysis I can add layers of nuance to each code. I can also create very sophisticated counts of these codes and subcodes for reports. The counts are designed to be exported to Excel and to be analyzed using pivot tables and other advanced features.

However, for many projects, there is a need or desire to think big again, or to think outside of the initial coding scheme, and that is where phase 3 begins.

III. **Phase 3: From deep to broad**

After breaking down codes into subcodes and subcodes of subcodes, I am ready for the next analytic phase. By now I have moved from "raw" data to "cooked" data, data that signifies. The third phase involves massive searching of the data and, using a unique feature of TA, searching searches, winnowing down the results of searches until I find exactly the data I need and the pattern of codes in my data that answers my research questions. TA provides for both search for codes and search of data (i.e., string searches). It also supports a very powerful, standardized formatting language called regular expressions (regex) for specifying complex lexical patterns. Note that while TA provides tools to bookmark data, search for significant patterns of code and text, and to combine searches in a variety of ways, e.g., to find the intersection of searches, the real art of finding patterns within the data largely depends on the imagination.

Eventually, I may come to find that families of codes also have relations to other families of codes, or codes to codes not in the same family. These non-familial groupings of themes are called "code sets" in TA. Code sets are simply arbitrary groupings of codes, ones that are genetically related and unrelated. Code sets can be named with full sentences; I don't have the "no spaces" restriction of code names.

In my research, I have often found that I use code sets to represent ideas that I want to test my "well coded" data against. For example, in a recent study of pre-service teachers using science fiction to teach science content, I started to identify which of the codes I created were related to teaching science for citizenship (rather than as a way to make more scientists). These codes were in various code families, but all related to issues like empowerment, equity, health and environment. I created a set of those codes, and from that was able to determine that there was no particular gender/sex bias to embracing those themes. I had not done gender analysis of this data prior to this, but it took about a minute to tell TA what the sex/gender of each of my interviewees was (this was done using an "if" command in TA: "if" the interview is with "Bob" sex is male, "if" the interview is with Sally, sex is female). Then, having created the code set, I searched for my coded passages (about 10 seconds), and then asked for a concept map (a "graph") of the relation of "sex" to "social issues". I also looked at how social issues broke down by each of my interviewees. The point is that I can ask questions about these large, ad hoc, cross familial categories by getting either graphic representations or counts of codes or other variables (context codes) I create against these code sets.

This has described a pretty typical research cycle using TA. There is one more part of this that deserves a comment or two, and that is how do I get images out of TA that communicate what I have discovered.

IV. **From analysis to print**

Having done the analysis, reanalysis, and code set analysis I will want to share my findings with the "world" (i.e., the other 5 people as fascinated by my topic as myself). TA has a wide variety of reports including code counts, cross counts of codes (how many code X and code Y apply to the same data), counts or concept maps (graphs) of the relation ships among codes, context codes, and code sets. Most of the table style graphics can be saved as a tab delimited text files, perfect for Excel™ or Word ™ , the code set report is generated as an html file which can be copied and pasted right into Excel™ to generate sums, averages, etc. The concept-map style graphs are generated through a program called Graphviz.app which can "export" graphs in many graphic formats (tiff, jpg, etc.).

To be honest, most of the graphs generated by TA I use for analysis not for presentation. I do count on TA to find the exact quotes I want to use in my papers, and TA allows me to copy the quotes without all of the coding information, which my readers don't necessarily want to see. When I have needed a graphic (a pie chart of code counts, for instance), I've saved the counts to a text file and used Excel™ to do the actual drawing rather than TA. Remember that one of the inclinations behind TA's design was to make sure its data was available for other programs to do some of the heavy lifting.

V. Conclusion

My idea behind this short piece was been to show how a researcher might use software to make sense of qualitative data in a digital age. For TA it means keeping track of lots of themes, and sub-themes, and uber-themes, and cross themes. As with many digital issues, the payoff comes with scale. If I'm analyzing one short interview, Post-Its™ are probably the way to go. If I'm dealing with 3-100 interviews, Post-Its™ are out of the question, and it's time to use TA or one of its cousins.

Recap:

Phase	**What I'm doing**	**TA Feature**
1	Creating a project Generating initial themes Transcribing Importing documents	New project Define codes New & import document Transcriber On-the-fly code creation
2	Coding Recoding	Define codes On-the-fly code creation Search for codes Selection of codes in searches Near selection in searches String searches Code definition Add code Recode
3	Code sets Reporting	Code sets Search for codes and code sets Graph reports Code counts Summary reports

Coding for Beginners with TAMS Analyzer

This is not comprehensive documentation of the TAMS Analyzer. It is more of a tutorial, or actually the necessary nuts and bolts to get going on a qualitative research project. From this documentation, users can read the documentation for the program and the coding system.

I. **What is coding?**
   Coding is simply a way of transforming raw information into data.
   In qualitative research (i.e., research relying primarily on interviews, observations, document collection) analysis proceeds by sifting through these raw (or if transcribed, semi-raw/slightly baked) pieces of information and deciding what each portion represents. In even modest size projects this can produce what is known as a data burden, i.e., too much information to be comfortably handled without some sort of mechanical system. Coding is one way to handle this. Using computers, sticky-notes, or scribbles in margins (and much more complex and Rube Goldberg-esque systems have been devised) relevant passages are "coded," i.e., labeled as to what that passage represents. Single chunks of text should be able to receive multiple codes, and in most modern coding systems they can receive various refinements to those codes, i.e., they can be subcoded (this isn't just an example of X--whatever that may be--but a subtype of X called Y; in TAMS the code would be X>Y; the ">" symbol is used to indicate various levels of sub-coding). An example would help....

   Consider a project involving studying the sounds that children think animals make. We might begin to collect songs and rhymes that have animal sounds in them as well as interviews about animal sounds. Of course, Old MacDonald would be part of our data. Take just this verse:

   Old Macdonald had a farm EIEIO
   and on his farm he had a pig, EIEIO
   with an oink oink here and an oink oink there
   here an oink, there an oink, everywhere an oink, oink
   Old Macdonald had a farm EIEIO

   We would like to mark "oink, oink" as the sound a pig makes. Now we may have a lot of codes and collect a lot of different information, including a lot of peripheral information about what children think animals are, so we need to design our code system carefully so we'll be able to keep track of all the information we'll have coming in. For our study, whenever we have an example of a sound we'll mark it with the code "sound" and then a subcode for the type of animal it is. So we would want to mark "oink, oink" with (and I'm using TAMS syntax here; if you use a different system this would be different) with "sound>pig". If we get information about what children know about pigs we will mark it "idea>pig". How we do that will be described later, but in theory (wihout TAMS) it could be done a lot of ways: sticky's hanging off the side with idea>pig on it, for instance. Computer coding usually involves selecting the text and somehow picking the code from a list.

II. What is TAMS?

TAMS, which stands for text analysis markup system, is simply a way of indicating in texts what the codes you're using are. It looks a lot like html and xml, which are languages used for making web pages, and I certainly was influenced by those ways of marking up text. The idea was to make a system that was easy to use; easy to see; and flexible enough that it could be done with any number of tools. Before I wrote TAMS Analyzer, for instance, I used a word processor for coding and a small program (still available at the TAMS website) to pull out the information I needed; then I used programs like Excel to do the actual analysis of that data.

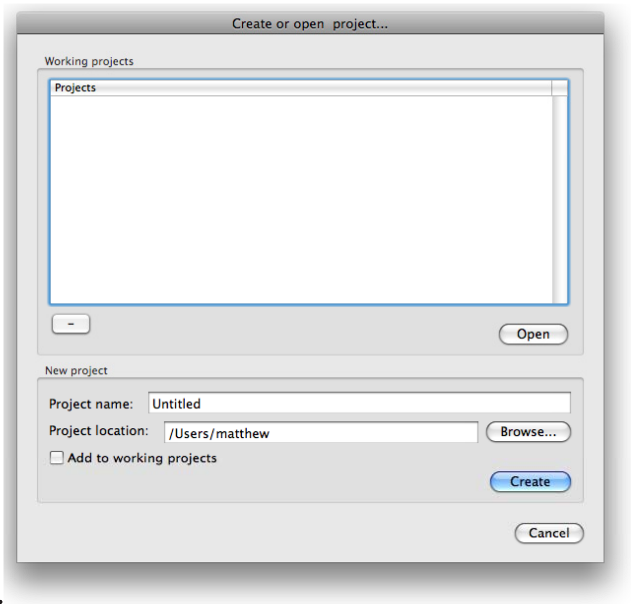
To mark up text you surround the part of the text you are interested in by "tags" which have in them a "code." The tag with a code in it looks like this: {mycode}. To indicate the end of the section of text you're interested in you put another tag with a "/" in front of the code name: {/mycode}. So in our animal example we would "type" into the text "{sound>pig}" before the words "oink, oink" and "{/sound>pig}" after those immortal words (oink, oink). Now that text has been coded! It would look like "{sound>pig}oink, oink{/sound>pig}" This means in TAMS that "oink, oink" is an example of sound subtype pig. You could do it with any wordprocessor!

III. What is TAMS Analyzer (aka TA)?

TAMS Analyzer was my attempt, after using TAMS for a while, to create a more complete application for coding, searching for codes, and recoding (going back through and adding levels to the codes). TA is still not the whole megilla, it doesn't have graphing, for instance. For many projects you will still want to save the results and use Excel, Neo(aka Open)Office or other programs (Panorama is a wonderful database for this sort of analysis) to do the more refined counting and graphing of results; but TA can take you at least 4/5 of the way there! For most projects it will be all you need.

IV. **Starting TA and understanding the parts**

When you first run TA you will see the "New project" creator:



.

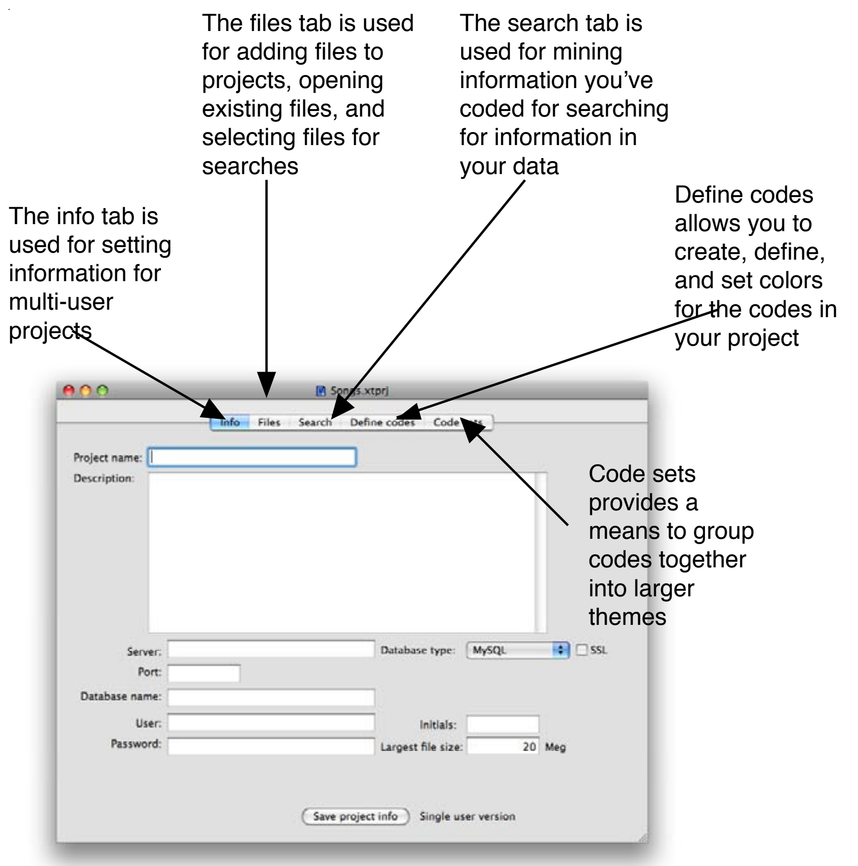
Figure 1. New Project Creator

Using this you pick the location and give the name to the new project. Do not use any extension when entering the name. (1) For this tutorial call it "Songs". (2) Before you click **create,** determine where TA should save this information by using the **browse** button to pick the folder where you want your project saved. Finally, because this is a file we will work with throughout the tutorial, let's add the project to the work menu. (3) Click the "Add to work menu" check, and (4) then click the **create** button.

You will notice that you have one very complicated window (See figure 2). It is called the workbench (aka project window). Every research project you have has to have one of these. It is what you have to open to do any further work. **Never open the data, result, or other files directly**. Open your project and use this window to open the data and results files in your project. The file associated with this window (it has the extension xtprj which stands for XML Tams Project) contains project wide information like the location of your interview and other files as well as housing your codes (which will be explained later) and their definitions. Note that the Project window will save itself as changes are made!

A. The Workbench window

The workbench or project window ties your files together. The purpose of the workbench is multiple. This is the window you use to define your codes. It is where you create and add data files. It is also the window where you put together multi-file searches. Finally, it is the window that allows you to access your whole project .


The files tab is used for adding files to projects, opening existing files, and selecting files for searches

The search tab is used for mining information you've coded for searching for information in your data

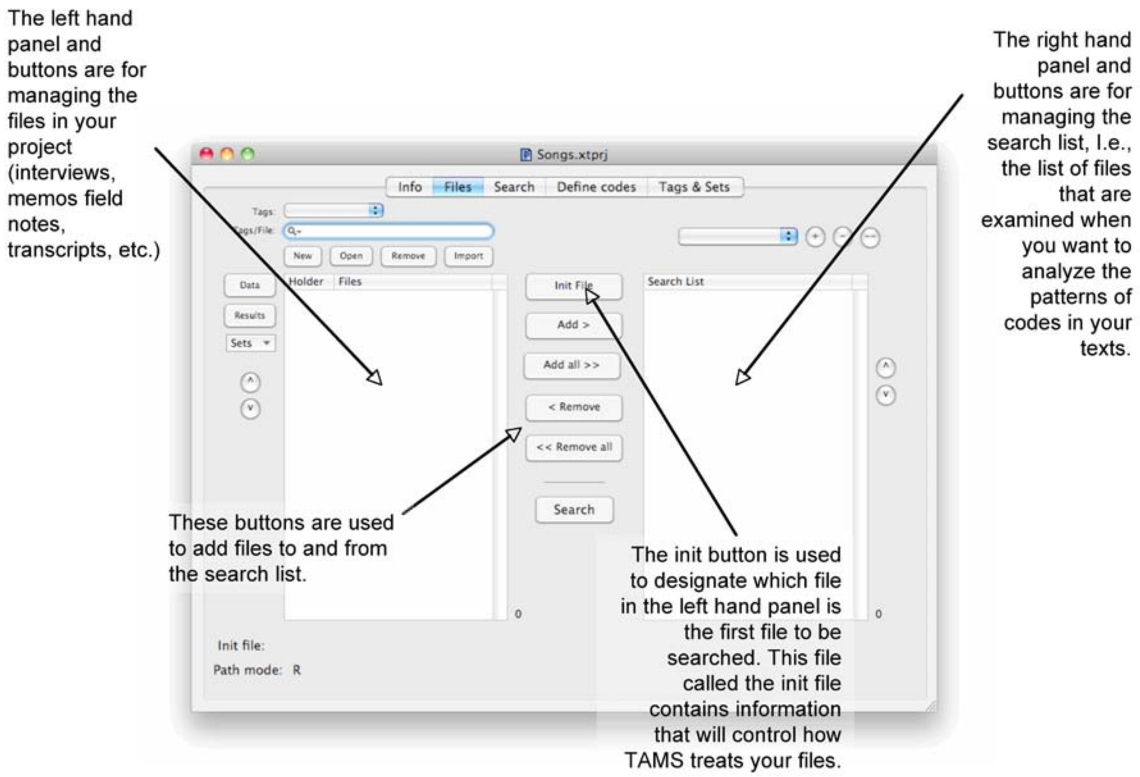
Define codes allows you to create, define, and set colors for the codes in your project

The info tab is used for setting information for multi-user projects

Code sets provides a means to group codes together into larger themes

Figure 2. The Workbench

We will begin by clicking on the file tab so we can create a new document in our project. The workbench will then look like this:


The left hand panel and buttons are for managing the files in your project (interviews, memos field notes, transcripts, etc.)

The right hand panel and buttons are for managing the search list, I.e., the list of files that are examined when you want to analyze the patterns of codes in your texts.

These buttons are used to add files to and from the search list.

The init button is used to designate which file in the left hand panel is the first file to be searched. This file called the init file contains information that will control how TAMS treats your files.

Figure 3. The Files Tab

B. Document Windows

To make a document window you need to click the new button on the files tab of the project window. The new button is right over the "Files" list view on the left side of the workbench window under the tab buttons:

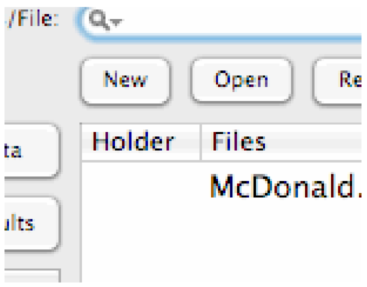
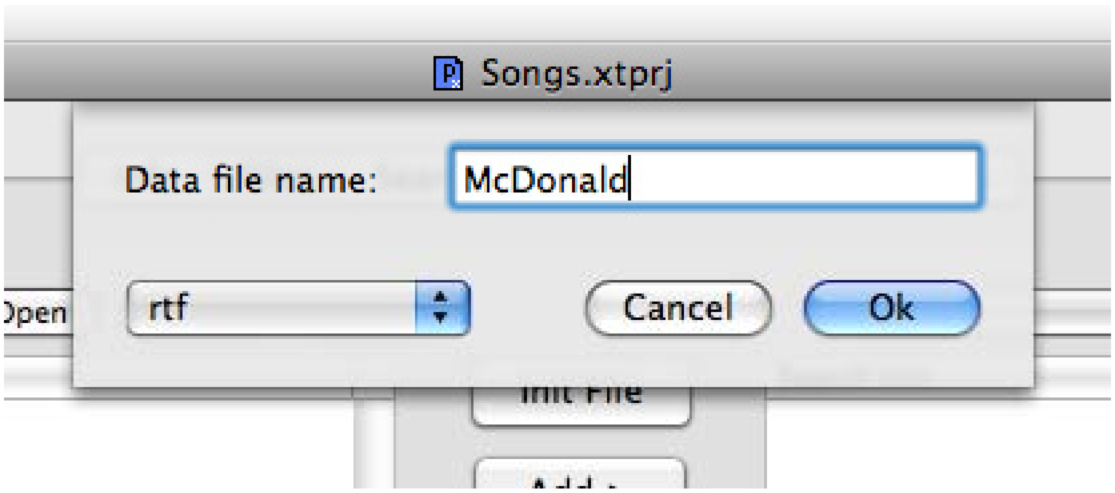


Figure 4. The New Button On The Workbench Files Tab
Makes Blank Documents Windows

Click the new button, and you will be prompted for a name. Type McDonald:



Data file name: McDonald

rtf

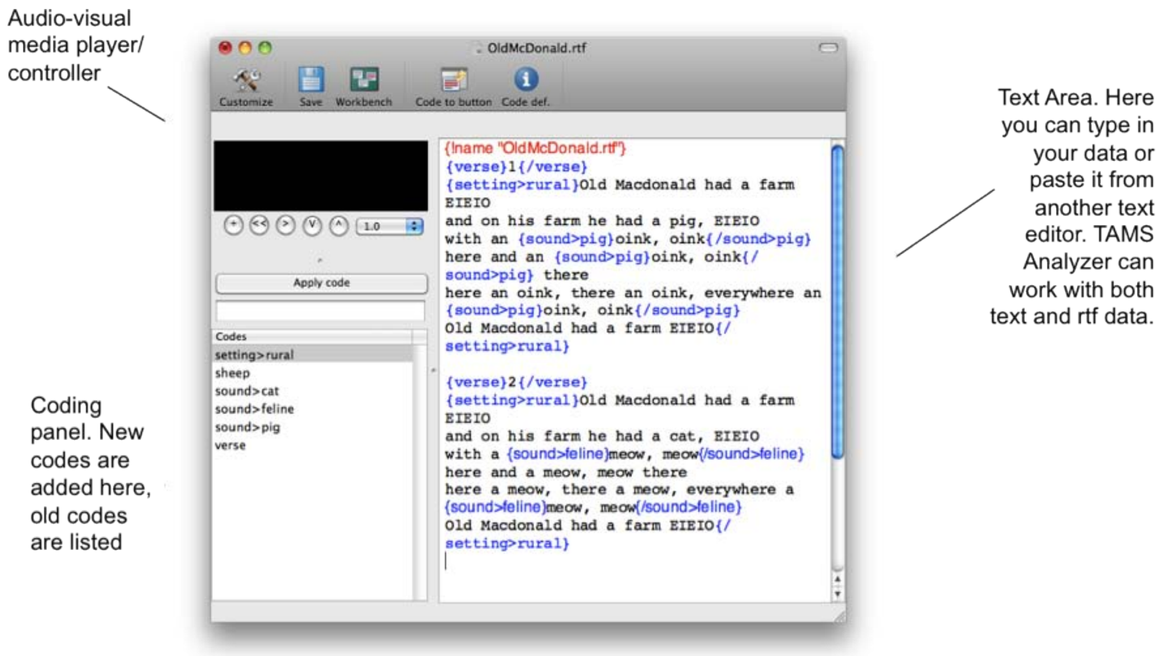
Figure 5. Naming Data Files

Once you click “Ok” you will see an almost blank document window. **Starting with TAMS Analyzer 3.0, the window actually starts with a little bit of helpful text at the top of the window. Leave it there and start your data below the text in braces.**

The document window is a sort of specialized word processor (or text editor) designed for coding data.

Here is my guided tour of the document window: In document windows there is a toolbar running across the top which is where you can put text and codes you use a lot. The document window toolbar also has a button to move the workbench to the front and to save the current document. Underneath this are two tabs, like those for a rolodex. One says edit, the other says search. We’ll just work in Edit mode for now, which is the one that the document window starts in. The other tab is for searching for data in this particular file.

The Edit pane is divided into three parts. On the right side is a big pane which is where you can enter your text. On the lower left side there are buttons and fields for managing codes (well, there’s one button here which toggles the ruler, but everything else is about codes on the left side of the window.) On the top left side there are buttons for adding and controlling video and audio files that are attached to this document (this might be the transcript or log of an interview you have as an mp3 file, for instance). These buttons (labeled +, <<, >, ^, and v) act as a built in transcription machine. (See the Audio-Visual How To in the How Tos folder.)



Audio-visual media player/ controller

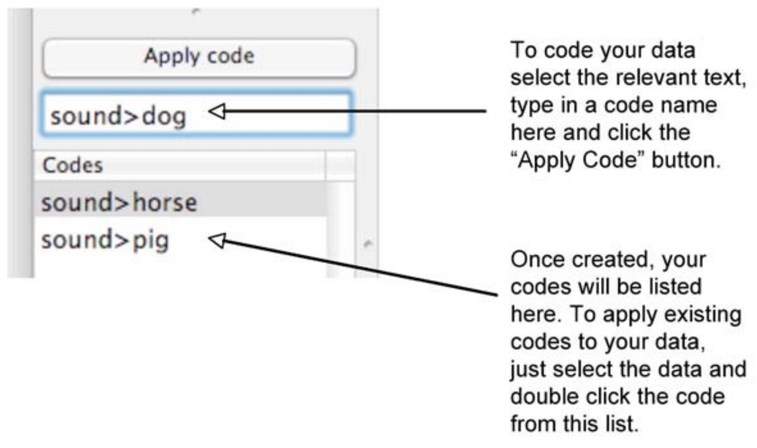
Text Area. Here you can type in your data or paste it from another text editor. TAMS Analyzer can work with both text and rtf data.

Coding panel. New codes are added here, old codes are listed

```
{!name "OldMcDonald.rtf"}
{verse}1{/verse}
{setting>rural}Old Macdonald had a farm
EIEIO
and on his farm he had a pig, EIEIO
with an {sound>pig}oink, oink{/sound>pig}
here and an {sound>pig}oink, oink{/
sound>pig} there
here an oink, there an oink, everywhere an
{sound>pig}oink, oink{/sound>pig}
Old Macdonald had a farm EIEIO{/
setting>rural}

{verse}2{/verse}
{setting>rural}Old Macdonald had a farm
EIEIO
and on his farm he had a cat, EIEIO
with a {sound>feline}meow, meow{/sound>feline}
here and a meow, meow there
here a meow, there a meow, everywhere a
{sound>feline}meow, meow{/sound>feline}
Old Macdonald had a farm EIEIO{/
setting>rural}
```

Figure 6. The Document Window And Its Parts (Edit mode)

The “Apply code” button takes what you type into the code field (see top arrow figure 7), checks to see if it’s a new code; if it is it asks for a definition; in either case it applies the code to whatever text is selected in your editor pane.



To code your data select the relevant text, type in a code name here and click the “Apply Code” button.

Once created, your codes will be listed here. To apply existing codes to your data, just select the data and double click the code from this list.

Figure 7. Adding A New Code Using The Button Panel

The buttons on the tool bar are worth a little explanation.

Code to toolbar: This button will take whatever code is selected in the code list and make a toolbar button for it... Makes it easy to access codes you use a lot.

Code def.: This button will pop up a definition of the code in a window, to remind you what your codes meant when you first used them

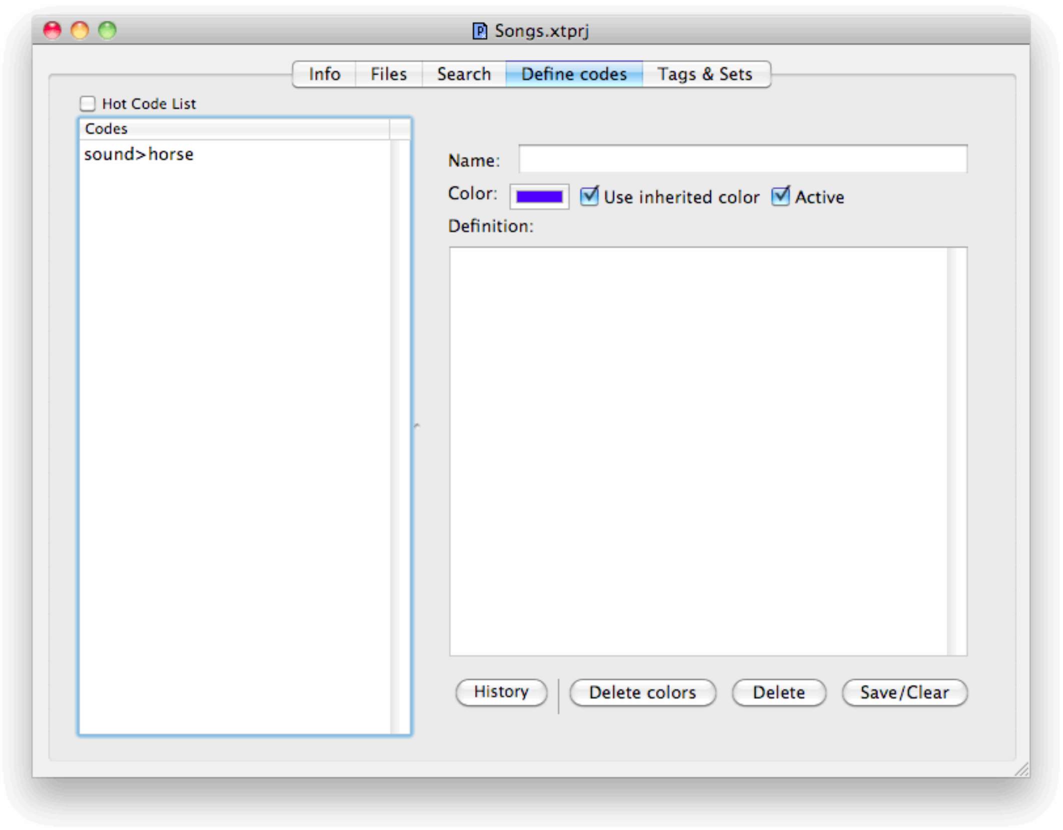
Workbench: This brings up your project window.

Once a new code is added it will appear in a list under the word Codes (in the figure 6, no codes have been added to the project yet). As we will discuss in the next section, the general way you code text is by selecting the text you want in the right side of the window and double-clicking the code you want from the list on the left side. But this will be handled in the next section: my purpose here is just to point out the anatomy of this window.

C. Other windows
There are a couple of other important windows which I'm not going to talk about yet... The most important of these are the windows that contain your "results", by which I mean the results of searches for relevant data. This will be discussed in the third part of this tutorial: Analysis for Beginners with TAMS Analyzer.

V. **How do I code in TA?**

A. Adding codes with the project window
We've seen one way to add new codes: select some text, fill in a code into the code field, and then click the new button. Sometimes, however, a researcher comes to a project with codes already in mind and needs to enter them, even before adding a document or transcribing an interview. One way to add such "a priori" codes is through the define codes tab on the Workbench. To find this, move your workbench to the front and click on the tab called "Define codes". The workbench should now look as follows:



sound>horse

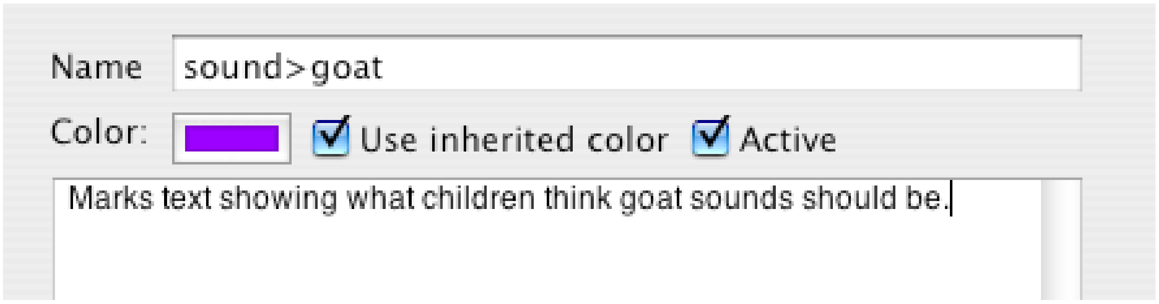
Name:

Color: ✓ Use inherited color ✓ Active

Definition:

Figure 8. Creating New Codes

The name of our first code will be sound>goat. We'll enter that next to "Name." In the big box underneath the name box enter the definition: "Marks text showing what children think goat sounds should be"



Name sound>goat

Color: ✓ Use inherited color ✓ Active

Marks text showing what children think goat sounds should be.

Figure 9. Filling In New Codes

Let's enter a second code. Click the "Save/Clear" button and add a code "sound>dog" with the definition "Marks text showing what dogs sound like to children".

When you are done typing the definition, either click a different tab, or press the “Save/Clear” button. The program will save the definition in the project window,

B. Adding a new code from the document window
Let’s code “Old MacDonald’s farm,” or at least one verse so that we get the idea. Make sure you have an open document window. If you don’t already have a document window, **click the “New” button on the <u>files</u> tab of <u>workbench</u>**.

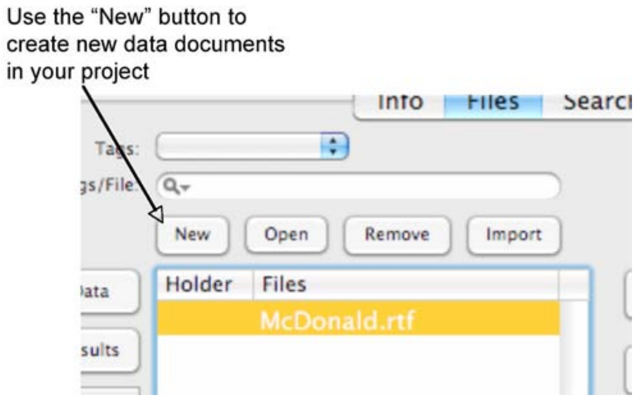
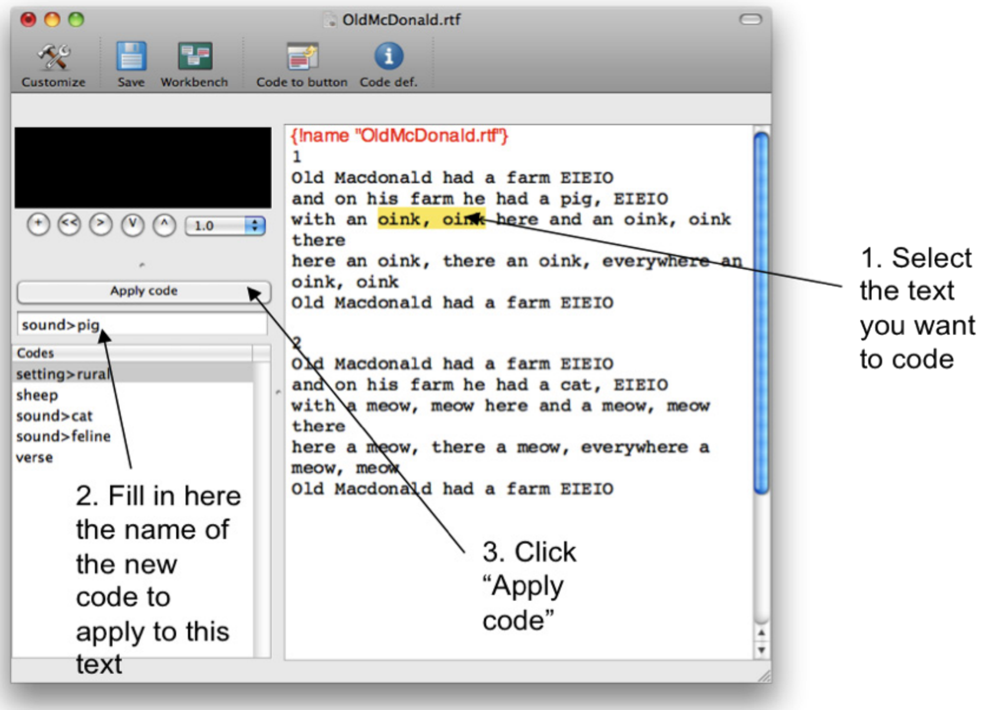

Use the “New” button to create new data documents in your project
Info
Files
Search
Tags:
New
Open
Remove
Import
Holder
Files
McDonald.rtf

Figure 10. The new button to make a new document

Fill in a name (e.g., McDonald) and press the Ok. This should provide a nearly blank window. Under the !name tag type the following verse into the window:

Verse 1:
Old Macdonald had a farm EIEIO
and on his farm he had a pig, EIEIO
with an oink oink here and an oink, oink there
here an oink, there an oink, everywhere an oink, oink
Old Macdonald had a farm EIEIO

This will go on the right hand side. Let’s now assign the code “sound>pig” to the “oink, oink”. **Select “oink, oink”** and then on the left hand side type in “sound>pig”:



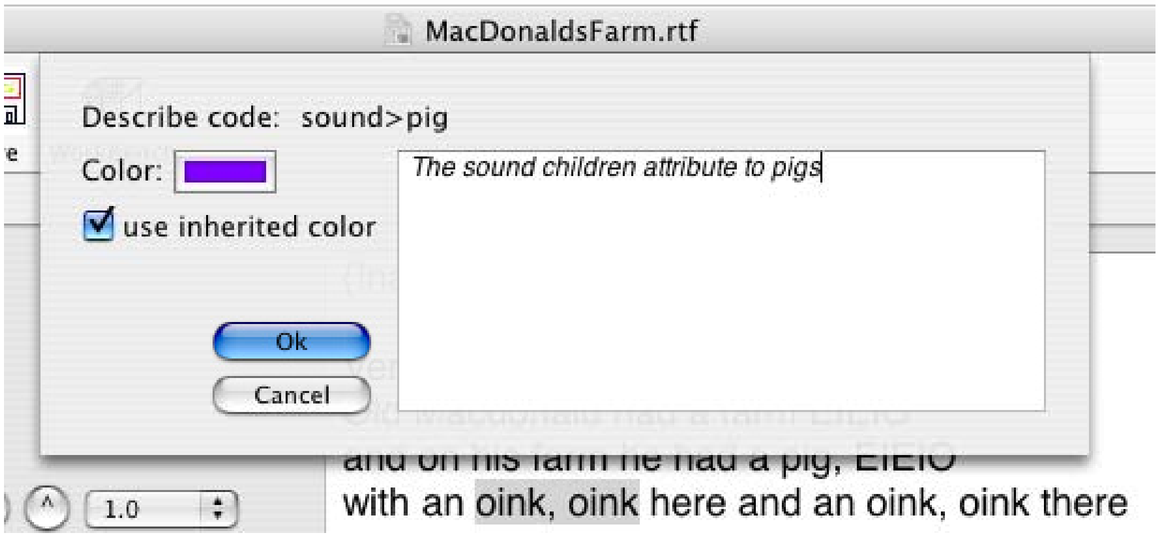
```
{!name "OldMcDonald.rtf"}
1
Old Macdonald had a farm EIEIO
and on his farm he had a pig, EIEIO
with an oink, oink here and an oink, oink
there
here an oink, there an oink, everywhere an
oink, oink
Old Macdonald had a farm EIEIO

2
Old Macdonald had a farm EIEIO
and on his farm he had a cat, EIEIO
with a meow, meow here and a meow, meow
there
here a meow, there a meow, everywhere a
meow, meow
Old Macdonald had a farm EIEIO
```

sound>pig

setting>rural
sheep
sound>cat
sound>feline
verse

Figure 11. Creating a new code

Then click the “Apply code” button. A dialogue will drop down asking for the code’s definition.



Describe code: sound>pig

Color: 🟪

☑ use inherited color

The sound children attribute to pigs

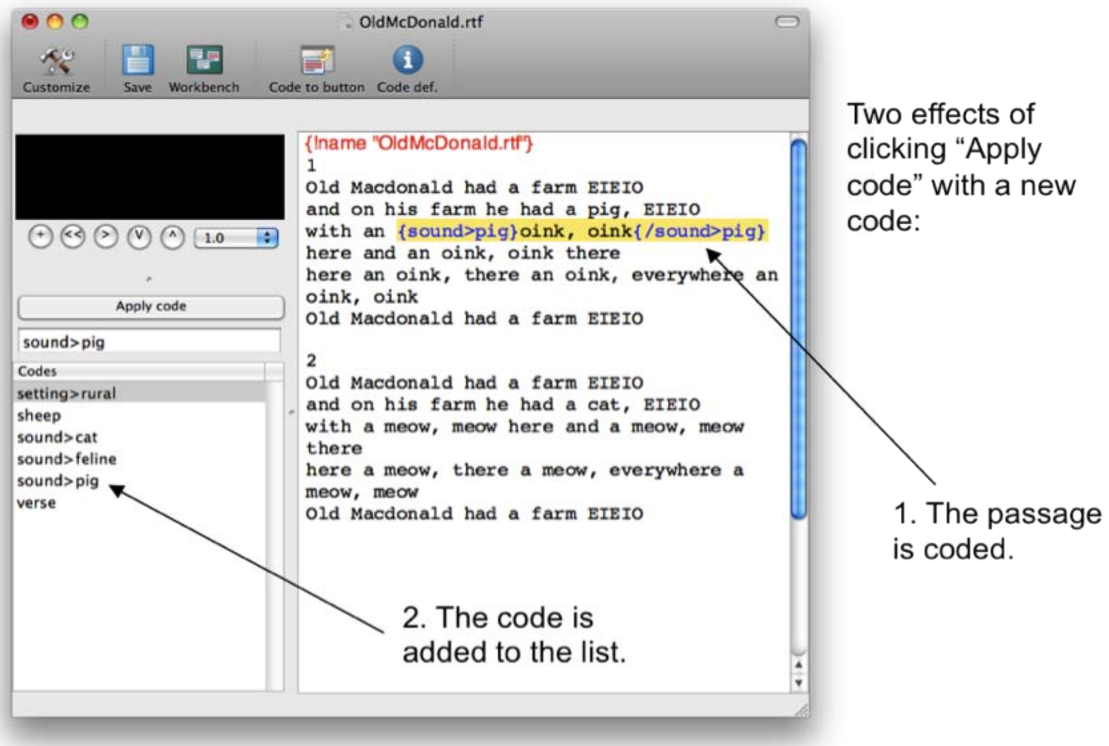
and on his farm he had a pig, EIEIO
with an oink, oink here and an oink, oink there

Figure 12. Adding a Definition

Clicking the Ok button will have 3 effects:

1. This code and definition will be added to the code list in the workbench
2. The code list on the left of your document window now has a new code

in it (sound>pig) and
3. “oink, oink” has now been coded:



```
{!name "OldMcDonald.rtf"}
1
Old Macdonald had a farm EIEIO
and on his farm he had a pig, EIEIO
with an {sound>pig}oink, oink{/sound>pig}
here and an oink, oink there
here an oink, there an oink, everywhere an
oink, oink
Old Macdonald had a farm EIEIO

2
Old Macdonald had a farm EIEIO
and on his farm he had a cat, EIEIO
with a meow, meow here and a meow, meow
there
here a meow, there a meow, everywhere a
meow, meow
Old Macdonald had a farm EIEIO
```

```
sound>pig
Codes
setting>rural
sheep
sound>cat
sound>feline
sound>pig
verse
```

Figure 13. The Effects Of The New Button

C. Using existing codes
Once you’ve taught your project a new codes applying them is a breeze. Simply select text and double click the code from the code list (that’s the list of codes on the left side of each document window). We could just keep selecting “oink”’s for instance and double click sound>pig from the left side of the window.

With a second code “sound>cat” the document might look like this:

Old Macdonald had a farm EIEIO
and on his farm he had a pig, EIEIO
with an {sound>pig}oink, oink{/sound>pig} here and an {sound>pig}oink, oink{/sound>pig} there
here an oink, there an oink, everywhere an {sound>pig}oink, oink{/sound>pig}
Old Macdonald had a farm EIEIO

Old Macdonald had a farm EIEIO
and on his farm he had a cat, EIEIO
with a {sound>cat}meow, meow{/sound>cat} here and a meow, meow there
here a meow, there a meow, everywhere a {sound>cat}meow, meow{/sound>cat}
Old Macdonald had a farm EIEIO

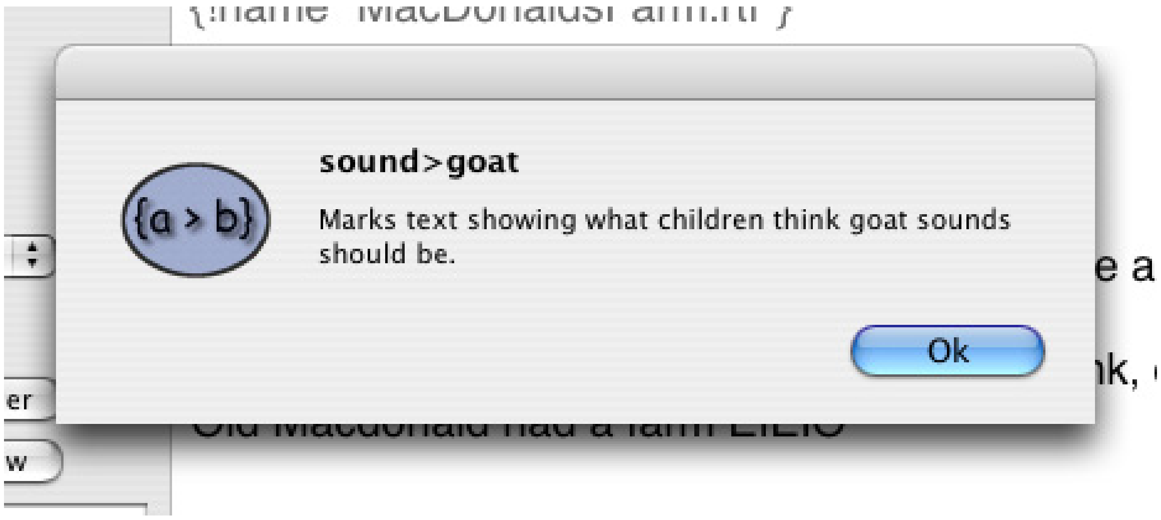
While this will be dealt with later: coded sections can be overlapped and nested. Both "{a}{b}some text{/b}{/a}" and "{a}some{b}text{/a}{/b}" just fine (notice how the first is an example of nested codes, the second is overlapped).

In the following example I've also coded the whole verse as the place that children think animal lives with a "setting" code, here subcoded to setting>rural:

{setting>rural}Old Macdonald had a farm EIEIO
and on his farm he had a cat, EIEIO
with a {sound>cat}meow, meow{/sound>cat} here and a meow, meow there
here a meow, there a meow, everywhere a {sound>cat}meow, meow{/sound>cat}
Old Macdonald had a farm EIEIO{/setting>rural}

Here, the sound code (sound>cat) is twice coded inside a stretch of setting>rural.

D. Recalling the definition of a code
When you have a lot of codes and sub-codes (pig is a subcode of sound in our example) it is often hard to remember the definition you gave a code. No problem. Click one time on the code you're curious about from the code list and click the "Code def." button on the document's tool bar. There are also "def" buttons in various workbench windows that list codes. Your definition will pop up:


sound>goat

Marks text showing what children think goat sounds should be.

Figure 13. Code definition

E. Working with codes

As has been noted codes can be overlapped and nested. Furthermore, <u>related</u> codes can be nested and overlapped. There are no problems for instance with

{a}This is my {a>b} text {/a>b}{/a}

even though a>b is a subtype of a. That's fine. From TAMS point of view they are as different as pickles and tomatoes. That doesn't mean that codes can't cause trouble; they certainly can. See the section on "Problems with codes."

There are several tools that make working with codes easy. These are on the Coding menu. For example, while coding I find that I want to relocate the beginning or end tag. I'll read the next paragraph and realize that should be included in the coded passage. Start by selecting the tag. One way to do that is to click in the middle of the tag and pick "Find current code" from the menu. That will select the whole tag. Now you can use the mouse to drag and drop the end tag to a new location.

You may see one end of a coded passage and want to find the other end. In other words you want to find the other end of the code pair. By a code pair I mean the front code (that is the one that looks like {a}) and the back or end code (the one with the slash: {/a}). Just click in the middle of one end and pick “Find paired code” from the Coding menu; the other tag will then be selected and scrolled to.

Sometimes you may want to just move through the document code by code. This is easily done with repeated use of “Find current code” and “Find next code” Picking “Find next code” repeatedly will move you through the document.

Deleting code pairs is also something that TA makes easy. If you have just inserted the code by double clicking or using new, you can pick undo from the edit menu. Alternatively you can click on one of the tags (i.e., anywhere inside the braces) and pick “Delete code pair” from the Coding menu. (For new codes, this will not do anything to the code list, only to the document you’re working on).

Finally, all of the codes in a selection of text can be removed by (1) selecting the portion of text you want stripped and (2) picking “Remove codes from selection”.

F. Problems coding

There are a number of problems with coding that can crop up; and TA provides two tools to help you catch these problems.

1. Broken up codes: sometimes the mouse slips and tags can end up in tags: {setting>ru{sound>cat}ral}. Here {sound>cat} has accidentally been inserted inside of {setting>rural}. This will not make any sense to TA. If you pick “Check for pairs” this will select problem tags, basically tags that don’t seem to have an end or beginning. The one it shows you probably is not the problem tag, but it will be near the problem tag. It is a clue as to where the problem is. The program is telling you that TA can’t find the other end.

2. Incomplete codes: Sometimes while working with a document, a tag at one end or the other will get deleted. The solution is the same as for problem #1. Choose “Check for pairs” off of the Coding menu. A tag will be selected if there are problems (i.e., if there are not an even # of beginning and ending tags). This is a clue to the problem; for some reason, TA did not find a match for this.

3. Nested codes: Sometimes the same codes can end up inside each other. This might be represented by the following situation:

{a}Some text{a} that I'm {/a} trying to code {/a}.

This is not the sort of nested code that works with TA. It would be fine if the inner code was any code including a subcode of a; if it were a>b, for instance. The problem is that TA can't figure out where the passage ends, and it will choose the shortest passage. The phrase "trying to code" is not seen by TA as having been coded. These problems can be found by picking "Check for nested" from the Coding menu.

The moral of the story is clear, run "Check for pairs" and "Check for nested" often.

G. Saving

TA does **<u>not</u>** automatically save document windows. Save often. Remember: result and document windows you must save on your own. The project window takes care of itself

VI. Concluding comments

This is only the "A's" of the ABC's of coding. Once codes are added as described, you will find them available to every file in your project.

TAMS provides all sorts of additional tools for working with codes. There are also different types of codes, to start with. There are codes that describe entire documents (this file is an interview), there are codes that describe a section of a file (this is bob talking), and the codes we've been working with, which are called data codes, and which identify themes. See the users guide.

Some of the things that you might pursue through the users guide, exploring the program preferences, and continuing to work with your data are:

1. How to assign colors to different codes
2. How to group codes together into code sets

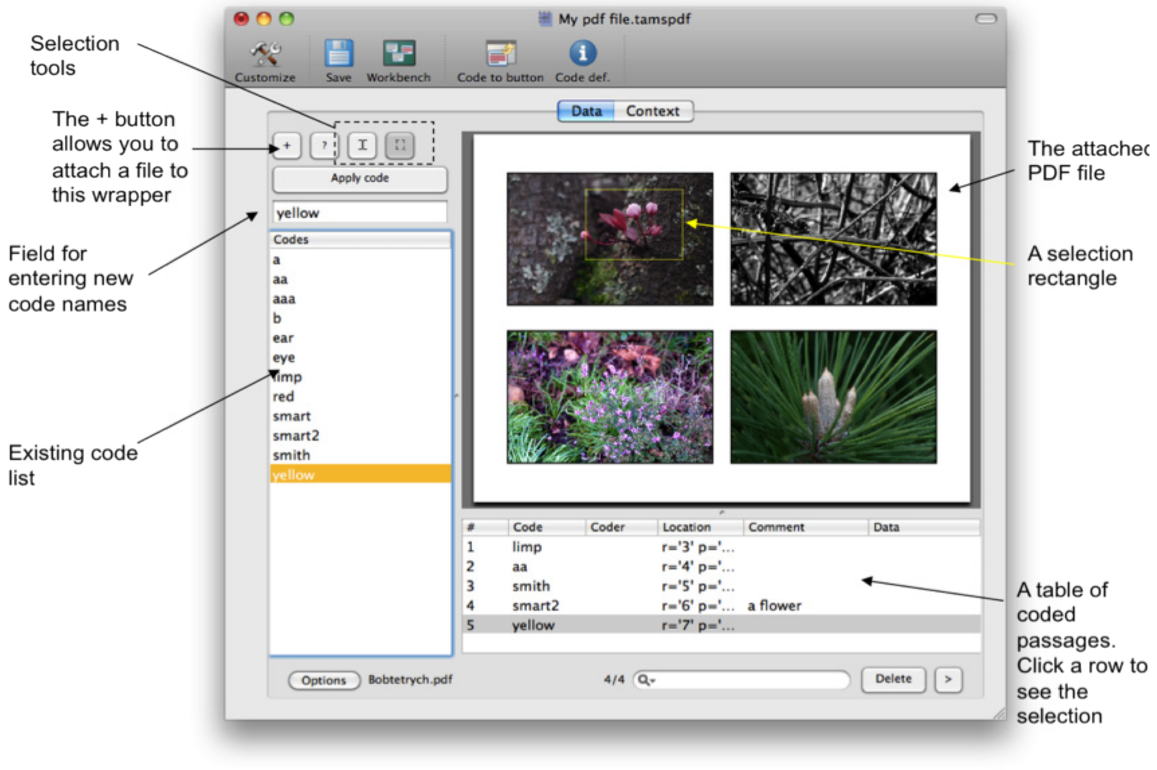
3. How add comments to a particular coded passage
4. How to use results windows to add layers of codes to already coded passages.

VII. Addenda: Coding PDF and pictures with TAMS Analyzer 4.0

TAMS Analyzer 4 adds support for coding PDFs and images. The process is very much the same one you saw above. You select the part of the image/document you want to assign a code to; you fill in the name of a code; you click "Apply code" or double-click the code name from the list. With PDFs there are two different ways of selecting portions of a document. You can use a rectangle tool to pick different areas of an image or page, or a text tool to select text (if your specific PDF document has selectable text; many do not).

One difference between the above example and these other file types is that initially setting up your PDF and image files involves an additional step: step 1 involves creating a "wrapper" file and step 2 involves attaching the PDF or image to this wrapper. To create a "wrapper" file (which TAMS will use to hold your coding information) you follow the process shown in figures 4 and 5 above. You pick "new" above the file list on the file tab (figure 4). When prompted for a name, pick "pdf" or "image" from the pop-up menu (see figure 5).

What you see is the pdf/image wrapper. It is a file of type tamspdf or tamsgaphic (see figure 14 for an example of a pdf wapper window). Before you can start coding you have to attach your PDF to this wrapper. Just click the "+" button and select the file you want to code. This is a a one time process. This will copy the file into your project folder and connect the file to the wrapper. Once loaded, save your wrapper file. Now you can start selecting and coding. Each coded passage will be represented in the table below the pdf/image.



yellow

Codes
a
aa
aaa
b
ear
eye
limp
red
smart
smart2
smith
yellow

#	Code	Coder	Location	Comment	Data
1	limp		r='3' p='...		
2	aa		r='4' p='...		
3	smith		r='5' p='...		
4	smart2		r='6' p='...	a flower	
5	yellow		r='7' p='...		

Bobtetrych.pdf 4/4

Figure 14. The PDF Window Wrapper

Analysis for Beginners with TAMS Analzyer

I. What you need to use this documentation

At this point you should have read the document called Coding for Beginners. If you have, hopefully you know the difference between workbench windows and document windows, you know that I use project windows and workbench windows interchangeably, and you can now create documents, enter data, create data codes, and apply those codes to sections of your document.

We'll be using a modified version of the Old McD. from *Coding for Beginners* for this analysis (which is merely to introduce you to the basic procedures for analysis):

```
{!context verse}

{verse}1{/verse}
{setting>rural}Old Macdonald had a farm EIEIO
and on his farm he had a pig, EIEIO
with an {sound>pig}oink, oink{/sound>pig} here and an
{sound>pig}oink, oink{/sound>pig} there
here an oink, there an oink, everywhere an {sound>pig}oink,
oink{/sound>pig}
Old Macdonald had a farm EIEIO{/setting>rural}

{verse}2{/verse}
{setting>rural}Old Macdonald had a farm EIEIO
and on his farm he had a cat, EIEIO
with a {sound>cat}meow, meow{/sound>cat} here and a meow,
meow there
here a meow, there a meow, everywhere a {sound>cat}meow,
meow{/sound>cat}
Old Macdonald had a farm EIEIO{/setting>rural}
```

This version of our old data introduces some very "not for beginners" ideas. The first of which is the metatag. A metatag is an instruction to the tams program. It's the way the researcher specifies how TAMS should treat the data. Metatags are always surrounded by braces "{" and "}", just like codes, but begin with an exclamation mark (!). Our new document begins with a critical metatag: **context**.

II. Context metatag

The context metatag tells TA that the following list of codes are different than normal data codes. These codes are used to indicate information that gives context to the data codes rather than the data codes themselves. In interviews the speaker's name might be a good context code. In field notes the time index, or location, or observer might all be context codes. We are telling TA that when it returns information about our data we want it to include this other information (who is speaking, the time code for field notes, or in this example the verse #) with the relevant data. This ability to attach information to each other is one of the real strengths of TA, but you can see that it is not exactly coding for beginners. Note that the context values have to precede the data that they are coding (actually if you read the user guide you can structure a document so that you can assign the context codes at different points—but for unstructured documents what I have said is true: the context values need to be set before the data codes.) Also, all of your context variables need to be "declared" in the first document that TA searches through. It's often helpful to have a document designated for holding these "up front" metatags. In TA this is called the init (as in initial) document.. **Note: For people who have read earlier versions of this documentation, !context is synonymous for the older code !repeat.**

Examples:

```
1.

{!context speaker}

{speaker}bob{/speaker}: {identity}this is me{/identity}
{speaker}mary{/speaker}: {identity>negation}no it's not{/
identity>negation}

2.
{!context speaker, time}

{time}110{/time}
{speaker}bob{/speaker}: {identity}this is me{/identity}
{speaker}mary{/speaker}: {identity>negation}no it's not{/
identity>negation}

{time}225{/time}
{speaker}bob{/speaker}: {identity}I can prove it{/identity}
{speaker}mary{/speaker}: ok do so
```

III. What is analysis?

Analysis is a process of finding out what information is present in your data and what that information means. Practically, it is the process of taking codes and finding out what pattern they form and testing meanings for that pattern (through searching for negative instances, for instance). To find these patterns in TA you ask the program to turn your interviews into tables that you can use to count instances of particular codes (or collections of codes) etc. In essence, what TA is all about is taking those codes and turning your interviews, field notes, etc., into a database (or spread sheet) that you can browse, sort or further search in a variety of ways. It is this culling through the data which is the heart of analysis.

IV. Searching, sorting and selecting: the grand plan

At the beginning of your analysis, after you have coded your documents, there are three essential operations you need to master: searching, selecting and sorting. That is what this tutorial will concentrate on. When you have a comfort with these three operations you should go on to explore data sets, autosets, and set operations. But those topics are not for beginners.

A. Searching

Searching is the procedure by which you take your coded data and turn it into a nice table-like database/spreadsheet. In this tutorial we will use the workbench for searching; though each document window has an individual search tab that you can use to find information just in just that document. The steps for searching for information are easy enough: specify a search list (which documents should the program look through?), specify search criteria (do you want to find just one code or a series of codes, or all your coded passages?) and press the search button. This will give you a "results window," i.e., a database/table of the passages that meet the criteria you established.

1. Creating a search list

First we have to indicate what documents our program should search through. Often we have lots of documents in a project that may represent searchable aspects of our work as well as other items that we do not want to search. To create a search list we use the "Files Tab" which contains a list of files in our project, and use the buttons in the center of the workbench to move items from the file list (the list of the documents in our project) to the search list (those that TAMS should search). If you want all of your documents brought over use the >> button. If you want to empty the search list of all documents use the << button. Otherwise click on one file in the file list (that 's the left hand list at the bottom) and click on > to add it to the search list. If you want to remove a file click on it in the search list and click the < button.

You can also change the order of files to be searched by using the ^ and v buttons on the left hand side of the screen.

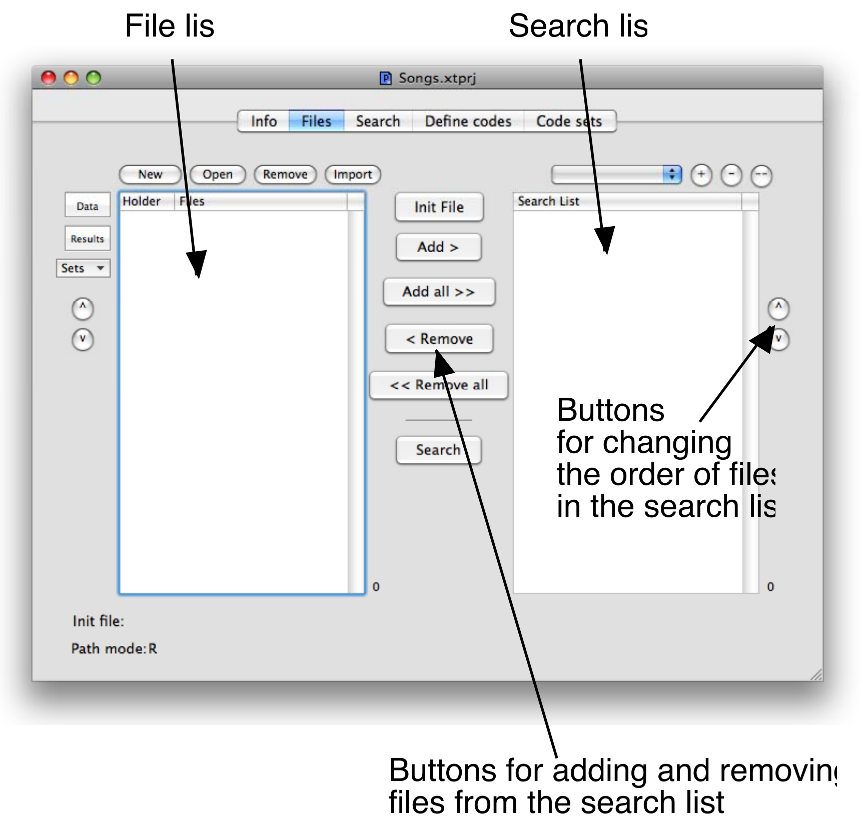


Figure 1: Assembling A Search List On The Workbench in the Files Tab

For our project, we will have one file in the file list. We need to add that file, let's say it's called "oldMacD.rtf," to the search list. Click on file name on the left side of the project window and press the button labeled "**Add >**" to append it to the list of files to be searched. You should see the name appear on the right side.

2. Executing a search

We have our search list now we need to tell tams what to search for. This we do through the "Search" tab of the workbench window. Our first search will be what is called an "unlimited" search, meaning we're not putting any limits on what data is returned: we want it all. This will demonstrate how TAMS turns raw stuff (interviews) into tables. To do an unlimited search make sure that the field marked "Search" is empty, that the pop-up menu underneath the check boxes says "Simple" and leave the "raw" box checked and the other two unchecked and press the button marked Search.

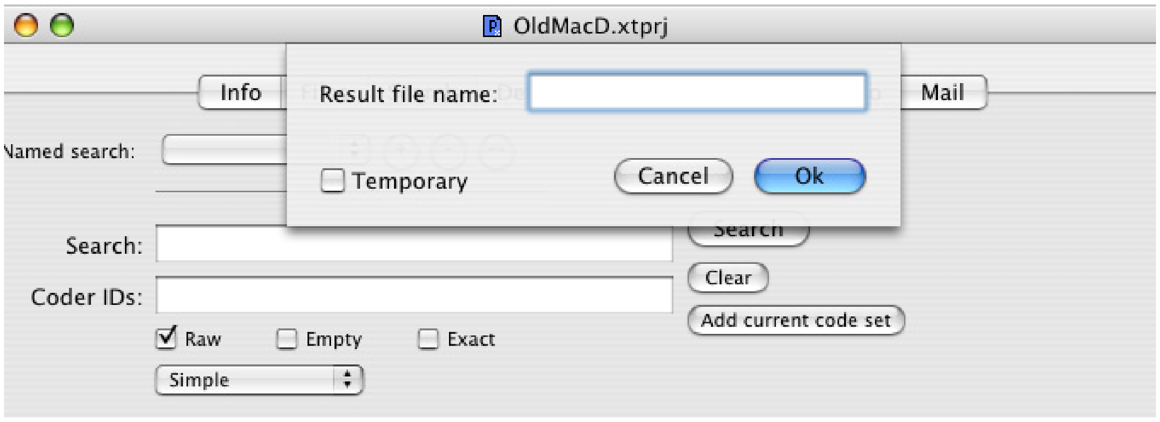
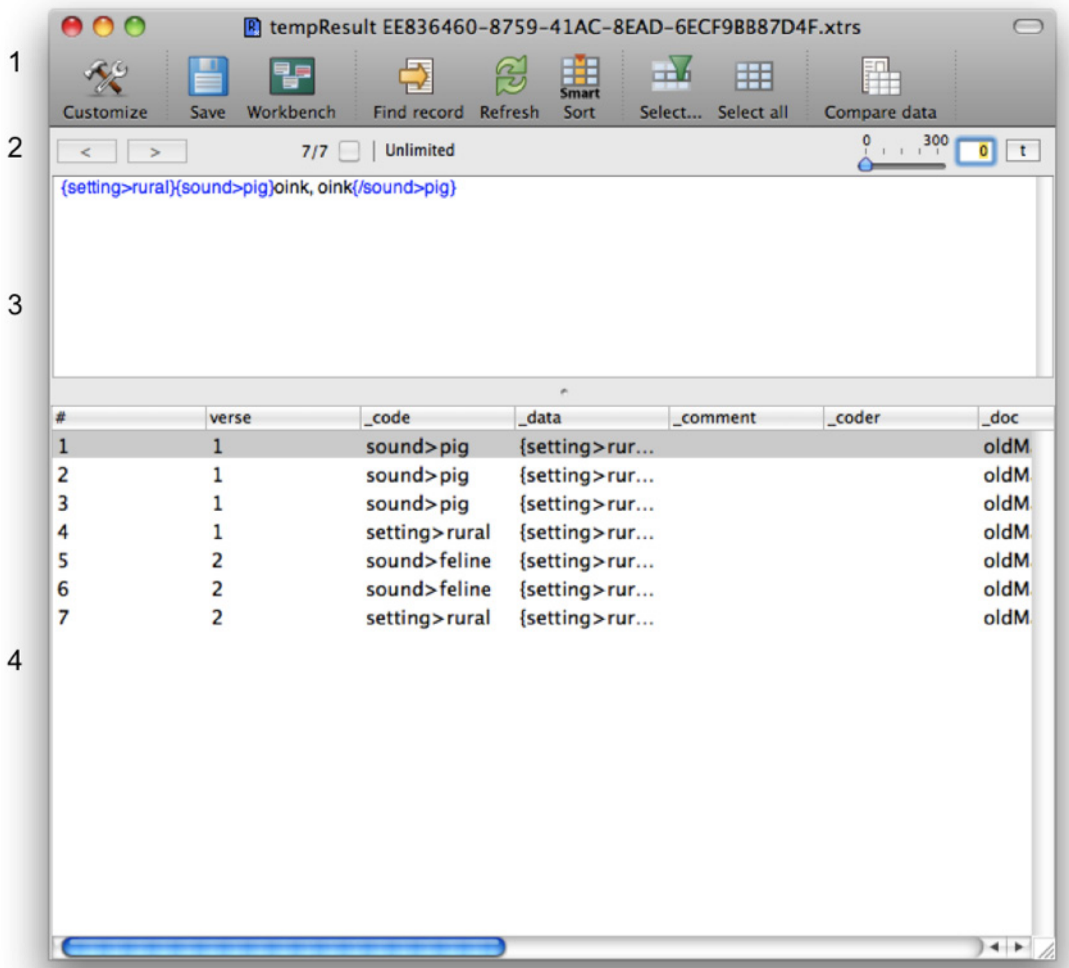


Figure 2. TAMS Prompting for a Result File Name

TAMS will now prompt you for a name for your search. Just leave the field blank, don't touch the "Temporary" check box, and hit the "Ok" button. This will automatically create a temporary result file, i.e., the next time you run TAMS it will throw out these results. This is typically what you want. Most results are exploratory and are not intended for permanent storage. If later you do want to save these results permanently you can use the File->Save as… option to give the file a real name and making sure that "Temporary" is not checked. On the other hand, if you know you want to keep this file for further analysis, fill in a name and make sure that temporary switch is unchecked. This is a typical results window:



{setting>rural}{sound>pig}oink, oink{/sound>pig}

#	verse	_code	_data	_comment	_coder	_doc
1	1	sound>pig	{setting>rur...			oldM
2	1	sound>pig	{setting>rur...			oldM
3	1	sound>pig	{setting>rur...			oldM
4	1	setting>rural	{setting>rur...			oldM
5	2	sound>feline	{setting>rur...			oldM
6	2	sound>feline	{setting>rur...			oldM
7	2	setting>rural	{setting>rur...			oldM

Figure 3. A Results Window

What you get is a results window: This is the window that lets YOU analyze your data. There are a lot of parts of a result window, so briefly:

1. There is a panel of buttons on top that you can configure and by default includes buttons for taking you back to your project window (Workbench), back to your document windows (find record), and for refreshing this window if one of your documents changes among other functions

2. Then is a status panel which gives you information about your results. It tells you how many records are showing, a check box which tells you whether any of your documents have changed (which means you should hit the refresh button) and a description of the data you're seeing. In this case it tells us that we have done an unlimited search. At the far right side is a little button that is used for doing "Select near" analysis: i.e., give me all examples of people talking about pigs near (within 5

lines or any definition of near you develop) people talking about goats.

3. Below the status panel is a large pane that shows us the data of the selected record. To select a different record, click on a different row of the table below. This is the browser pane.

4. Finally, occupying the whole bottom of the window, there is the table of records: one row represents one passage of text that met the criteria we asked for.

A simple, unlimited search returns a row for every coded passage in our data. If a section of text has two codes: "{code1}{code2}This is my text{/code2}{/code1}", for instance, that text will appear in two different rows in the table of records, one for code1 and one for code2 (yes, this might mean that there is redundancy. A non-simple search removes the redundancy and returns one row for every stretch of text surrounded by codes (no how many codes are embedded in that text. There is a price for this; you actually lose information searching that way).

By clicking on one of the rows you can use your up and down arrow to browse through your data, examining the passages in the browser pane. If you want to look at one row of the data in its original context double-click the row.

Finally, it is worth examining the columns given to you in the table of records. The first column is a simple numbering of the shown records. Next will appear all of your variables (repeats, context, universal, etc.). Here we can see which verse each coded passage came from. If we had an interview we could see who was speaking.

Next comes the code of the passage put into a column TA creates called _code (all of the columns TA creates start with an underscore), followed by the data (called _data), followed by comments we've attached to the codes (another non-beginner function), the name of the coder if you are using multiple coders, the name of the document (in *doc),* followed by the number of characters into the document that the passage started, the length and location of the passage if there were no tags in the document (_bare_??) and finally the initials of the person who has checked out this document, which only applies to multi-researcher projects (_holder). Again, the columns that TA creates always start with the "_" (underscore) character.

Sorting is very easy at this point! We can sort the data just by clicking on a header of the column we want to sort and pressing the sort button. This is not the best way to sort information though, a subject we will address in a later in this tutorial.

3. Searching for specifics

There is a lot that can be done in unlimited searches, and more often than not that's what I use to explore my data. Often however, the researcher wants to see only a single code or a few codes. While that is possible to do in an unlimited search (see the section on "selecting data") it's easier just to search specifically for the codes of interest.

The easiest way to search for a specific code is to go to the search tab and simply double-click the code from the list of codes on the bottom left of the workbench. So if we want to see all of the sections of text coded as sound>pig we could double click sound>pig from the list and hit the search button. Then only the sound pig records would appear in the results window. To clear the search field to clear it of "sound>pig" so you could search for a different code, click the button called "Clear" on the work bench.

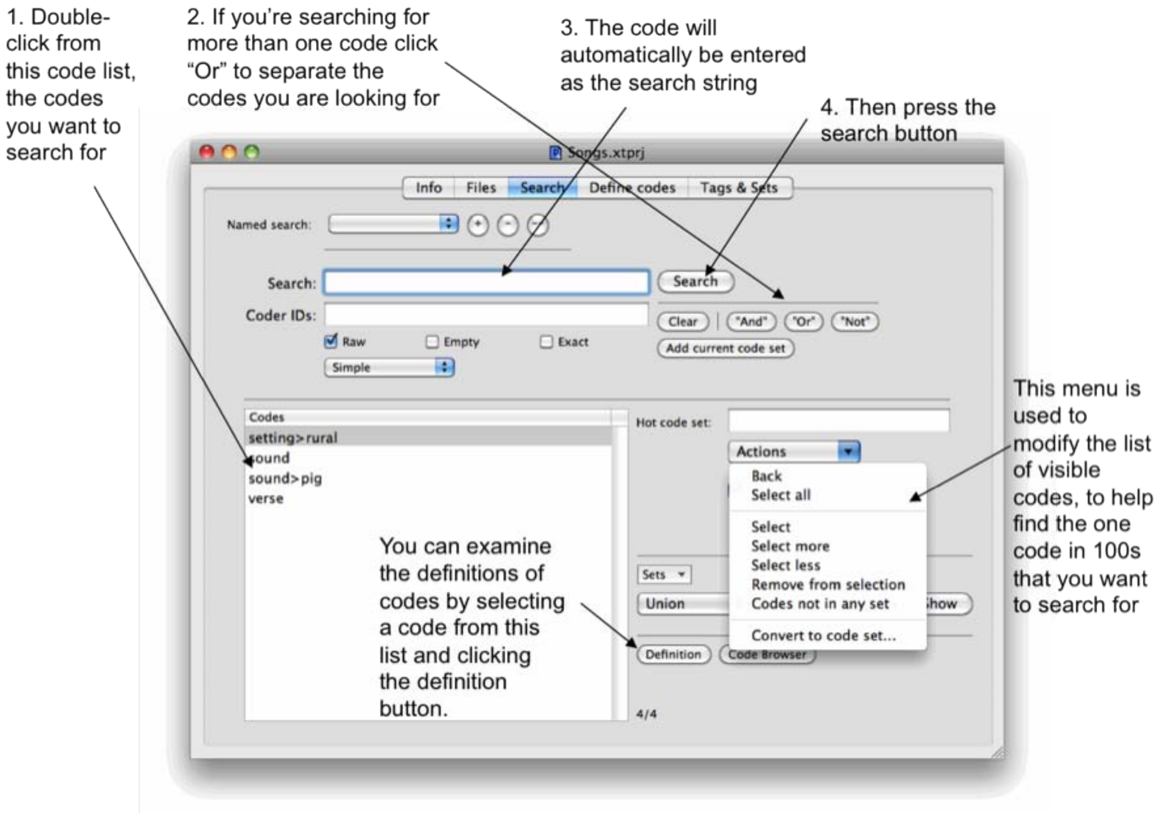

1. Double-click from this code list, the codes you want to search for
2. If you're searching for more than one code click "Or" to separate the codes you are looking for
3. The code will automatically be entered as the search string
4. Then press the search button
This menu is used to modify the list of visible codes, to help find the one code in 100s that you want to search for
You can examine the definitions of codes by selecting a code from this list and clicking the definition button.
Songs.xtprj
Info
Files
Search
Define codes
Tags & Sets
Named search:
Search:
Coder IDs:
Search
Clear
"And"
"Or"
"Not"
Add current code set
Raw
Empty
Exact
Simple
Codes
setting>rural
sound
sound>pig
verse
Hot code set:
Actions
Back
Select all
Select
Select more
Select less
Remove from selection
Codes not in any set
Convert to code set...
Sets
Union
Show
Definition
Code Browser
4/4

Figure 4. Searching for a Single Code

You could have also just typed the name of the code into the box labeled "Search:"

What If you were looking for either sound>cat or sound>pig? Simple:
1. Make sure that the Search: box is empty by pressing the "Clear" button.
2. Double click sound>cat off of the list
3. Type in a comma and a space after sound>cat in the Search: field box.
4. Double click sound>pig.
5. Press the search button

Alternatively you could have typed in "sound>cat, sound>pig" directly into the search box with the keyboard, and then pressed the search button. You could keep adding commas and different codes. If you wanted to search for three codes you could type "sound>cat, sound>pig, setting>rural", for instance.

The codes sound>pig and sound>cat are part of a family of codes called **sound**. If we just look for sound we will get everything that is in the sound family including anything labeled sound, sound>pig, sound>cat, sound>cat>persian, etc. If I only want to look for those things that are sound and not a subcode of sound, e.g., sound>cat, I have to click the “Exact” box under the search. Otherwise TA assumes I want the whole family.

Finally, and this is almost a beyond beginner item. If I want passages that have two codes, i.e., the intersection of coded passages, I would need to use a plus sign between the codes. In other words if I want to see the part of my document that is sound>pig and setting>rural at the same time I would type “sound>pig +setting>rural” into the search field.

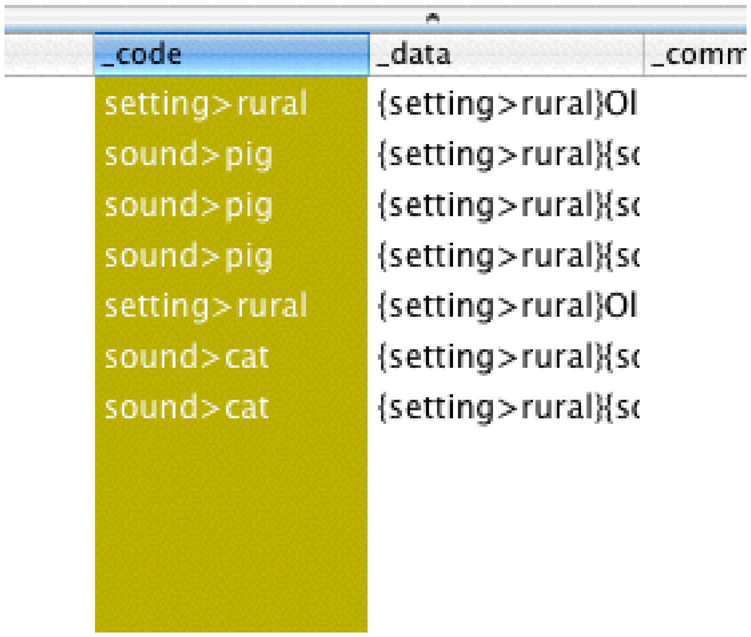
If you feel comfortable with all of this you can learn a few other tricks to searching by clicking the search tab on a document window and looking at the Help box.

B. Sorting

Finding information is the first part of analysis. We’re half done with the topic of finding information, but before continuing I want to offer as an interlude some notes on arranging information. This is the art of sorting, and TA can do very complex sorts. While beyond this tutorial, TA can also remember sort orders and use them in macros that are very powerful for finding key information across searches.

I’ve already noted that a simple way to sort is to click on a column in a results window and hit the sort button. The problem with this method of sorting is you have no control over the type of sorting that the program is doing. The more powerful sorting tools are under the “Results->Sort up” and “Results->Sort down” menus. These allow you to control the direction of the sort (Should A or Z be first? If A is first you are sorting up; if Z is first you are sorting down) and allow you to nest sorts, e.g., first sort A-Z on one column and then sort A-Z on a different column. That way you can sort by file name and then sort by the codes; the result will be all of the codes sorted by file name!

Alternatively, you may put three or four documents in your search list and then want to see how many documents use each code. So you want to sort by the codes and then sort within each code by document. Specifically you would click on the column with the header _code and then pick “Results->Sort Up->alpha” (i.e., alphabetically) Then you would click on the header of the column labeled _doc and pick “Results->Sort up->alpha within” The word **within** means keep the first column sorted appropriately, but rearrange items that match in the first column according to the currently selected column.” You could sort by any number of columns **within** each other. If you want to start over, just sort NOT within the others. Just pick “Results->Sort up->alpha” or whatever data format is appropriate.



_code	_data	_comm
setting>rural	{setting>rural}Ol	
sound>pig	{setting>rural}{s(	
sound>pig	{setting>rural}{s(	
sound>pig	{setting>rural}{s(	
setting>rural	{setting>rural}Ol	
sound>cat	{setting>rural}{s(	
sound>cat	{setting>rural}{s(	

Figure 5. Column _code is Now Selected for Searching or Selecting

TAMS can sort alphabetically, numerically either by integer or real (floating point) comparison, by date (though you must pick the format first by picking it from the Results->Sort options…->Date format menu item first—it's best if there's only one date format in a project), and by code. This latter type of sort is really not part of a beginners tutorial, but just so that you know, if I set the code level to 1 (by picking the "Results->Sort options->Code level" dialogue) sound>cat and sound>dog will be taken to be identical for the purposes of sorting since they match at the FIRST (this is the meaning of code level 1) level of codes. Picking code level 2 means both level 1 and 2 must match. Ok, that's an advanced topic.

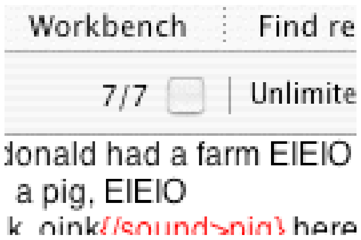
Now you can organize your data as well as find it!

C. Selecting

The third type of operation you need to know about is selecting. Selecting can be thought of as finding information inside of a search. It's a way of searching searches; or better stated searching results files. Remember that a results window is a fairly complete database, and a good database should be searchable. The terminology is confusing because selecting also refers to the way that we pick things on the screen with our mouse: select column _code, select the first field, etc. I'll try to use "pick" instead of select for the latter type of operation.

Let's take as an example trying to find the number of sound>cat codes there are out of the total number of coded passages. To start with, do an unlimited search: Go to your workbench and press the clear button and then the search button. Don't fill in a name, just click **OK**. You should now have a result window with one line for every coded passage in the document: i.e., one line per code.

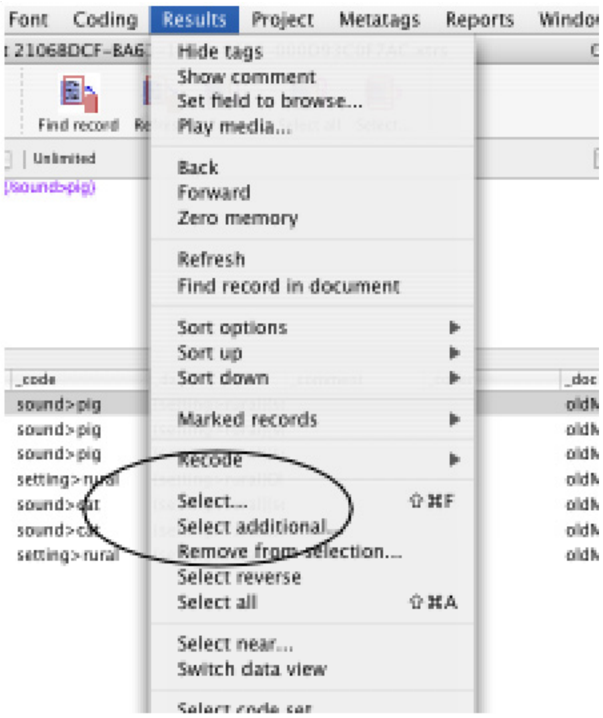
We can see that we coded 7 passages not counting our repeat codes by looking right under the buttons on the results window:



lonald had a farm EIEIO
a pig, EIEIO
k oink{/sound>pig} here

Figure 6. We found 7 coded passages (and all 7 are showing)

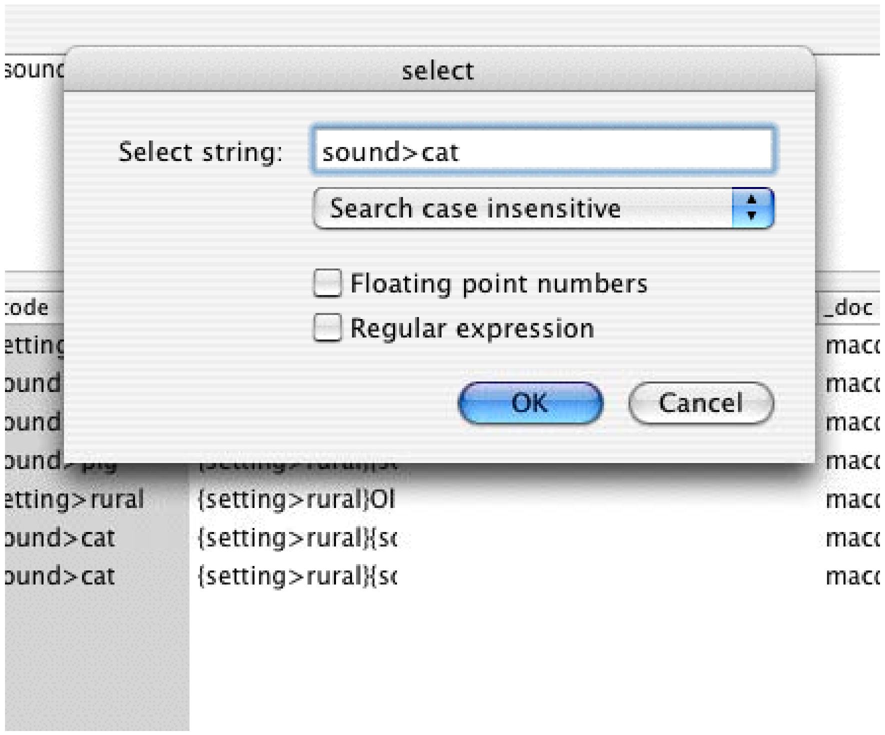
Let’s find the sound>cat codes. Pick the _code column by clicking on its header. Picking the column of interest is always step one of a selection. Then pick the Results->Select… menu option:



_code
sound>pig
sound>pig
sound>pig
setting>rural
sound>cat
sound>cat
setting>rural

Figure 7. Picking Select…

This provides a dialogue box into which you can type the phrase you are looking for: sound>cat.



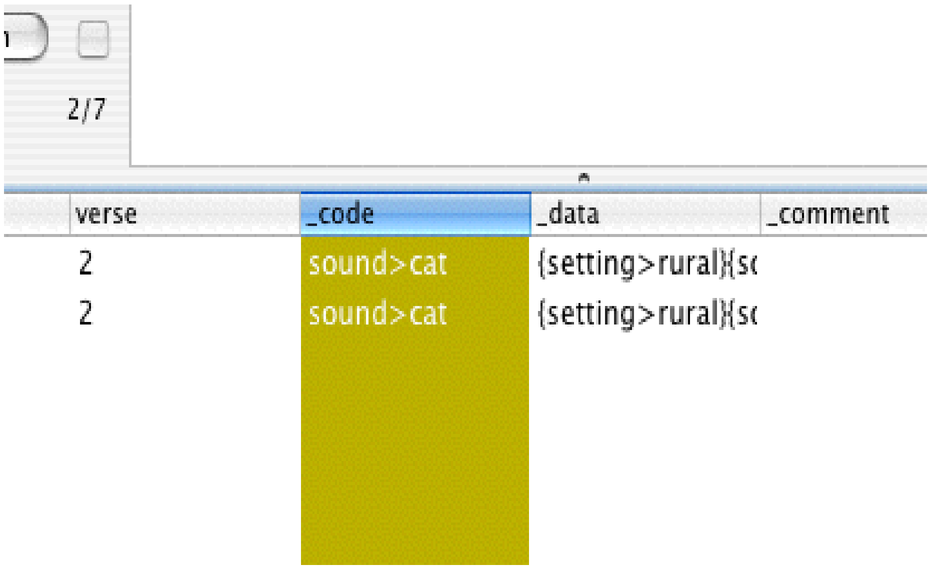
Select string: sound>cat

Search case insensitive

:ode		_doc
etting		macc
ound		macc
ound		macc
ound>pig		macc
etting>rural	{setting>rural}OI	macc
ound>cat	{setting>rural}{sc	macc
ound>cat	{setting>rural}{sc	macc

Figure 8. Selecting “sound>cat”

After clicking OK only those records that have sound>cat will be showing!

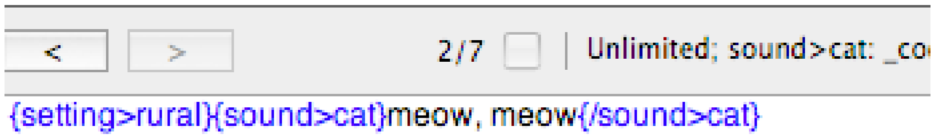


2/7

verse	_code	_data	_comment
2	sound>cat	{setting>rural}{sc	
2	sound>cat	{setting>rural}{sc	

Figure 9. After selecting “sound>cat”

Now only two records are showing. And the count indicator under the buttons shows us that 2 out of 7 total records are sound>cat.



Unlimited; sound>cat: _co

{setting>rural}{sound>cat}meow, meow{/sound>cat}

Figure 10. The results of selecting “sound>cat”

To see all of the records again pick “Results->Select all”

That’s the heart of selecting. We could modify this in a whole variety of ways mostly shown in the menu displayed in Fig. 6. We could switch the records that are showing and the ones not showing (so only records that were not sound>cat were visible), by picking “Results->select reverse”; or we could select additional records from the total pool of found records by picking a column and picking “Results->Select additional.” We could keep whittling down the records showing by picking more columns and picking “Results->Select…” again. Finally we could whittle down our findings by picking a column and picking “Results->Remove from selection…” By combining these and using different columns: your _code column, your repeat columns, and your _data column you can find many patterns inside your data.

V. A more complicated example… Less for beginners

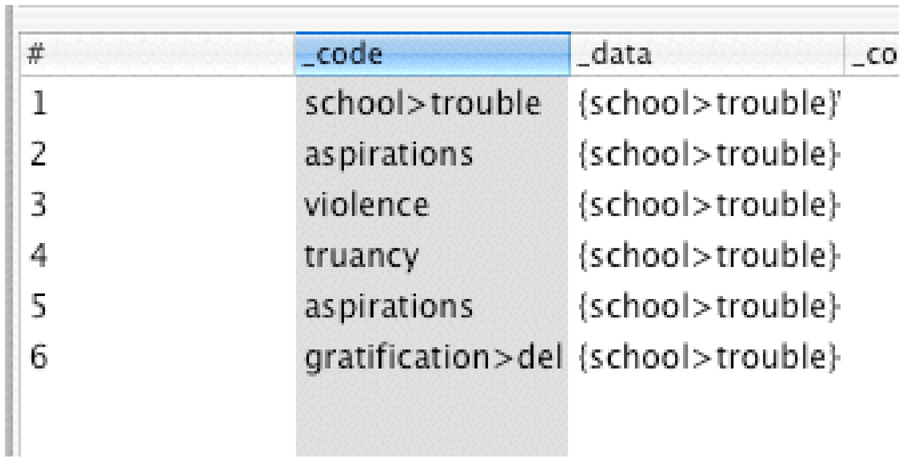
Before concluding this tutorial I want to work through a more advanced example of selecting. This is an alternative way of finding the intersection of coded passages by using Result->Select… It also shows some of the issues that can result from such searches. For this example I’m going to use this small passage:

```
R:  Well, my high school was known as a trouble school.
There were a lot of fights, and kids, uhm wandering around,
and most of us worked in factories on the [city’s] east
side. Most of us partied rather than worked.
```

We’ll code this as follows:

```
R: {school>trouble}Well, my high school was known as a
trouble school. {aspirations}We weren’t going anywhere. {/
aspirations}{violence}There were a lot of fights{/violence},
and {truancy}kids, uhm wandering around{/truancy}, and
{aspirations}most of us worked in factories on the [city’s]
east side{/aspirations}. {gratification>delayed}Most of
us partied rather than worked.{/gratification>delayed}{/
school>trouble}
```

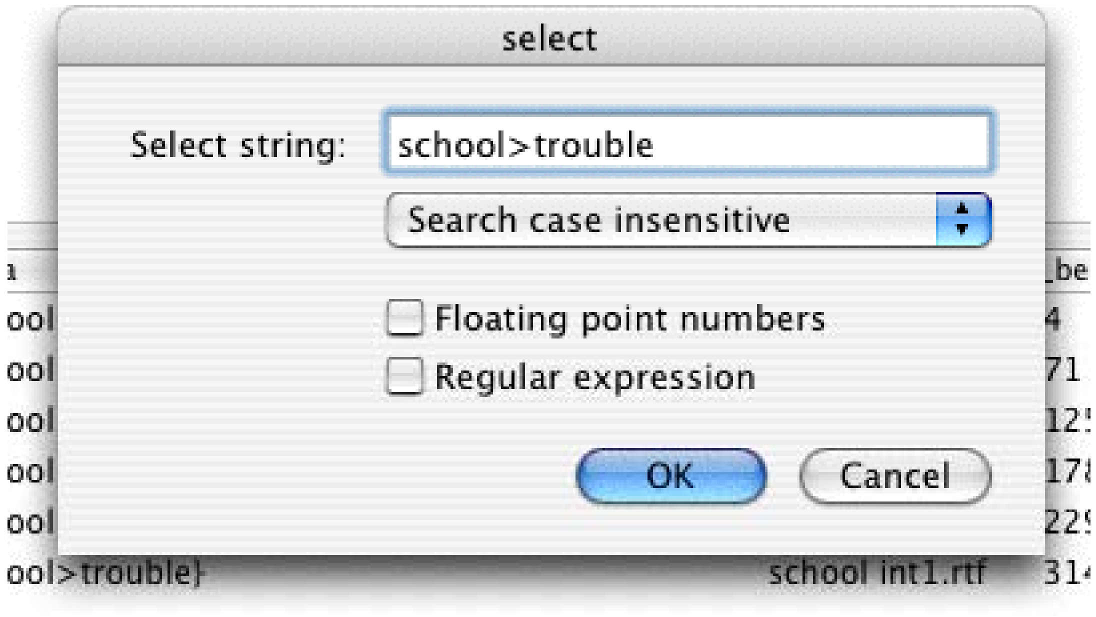
Now let’s imagine that we are searching for the intersection of discourses we’ve labeled “school>trouble” and “aspirations.” Maybe we want to see how high school image and occupational aspirations correlate. One way to find these intersections is through searching for “school>trouble+aspirations.” You can also find this from an unlimited, simple search, however. Reasons for doing so include the ability to do “set mathematics” i.e., the ability to look at how groups of coded passages intersect with each other. To do this we’re going to work from an unlimited, simple search; that means we hit the search button with no criteria filled in.



#	_code	_data	_co
1	school>trouble	{school>trouble}'	
2	aspirations	{school>trouble}	
3	violence	{school>trouble}	
4	truancy	{school>trouble}	
5	aspirations	{school>trouble}	
6	gratification>del	{school>trouble}	

Figure 11. Picking _code

Once we have done a search we are going to look for records that have both. As a first cut, let’s work with just those records that have “school>trouble” Select the _code column by clicking it’s header, and pick Results->Select… Fill in “school>trouble” and hit “ok.”



Select string: school>trouble

Search case insensitive

Floating point numbers

Regular expression

Figure 12. Selecting _code

Now we need to find the records that are left that have "aspirations" (in our example there is only one record to search, but that just points to the artifice of the example). Pick the _data column this time. Then pick Results->Select and enter "{aspirations". As the search string: voila: what you see are records that have both codes.

WARNING: In our example we would get a different number of records if we first searched for "aspirations" in the _code column and then "{school>trouble" in the _data column. We would eventually see the same text as we browsed through the records, but you should know that the number of records would be different. Finding how many passages coded with A have code B turns out not to be the same question as how many passages coded with B also have code A. Why? The first time we would get the passage marked as orange as the result of the first and second search:

```
R:  {school>trouble}Well, my highschool was known as
     a trouble school. {aspirations}We weren't going
     anywhere. {/aspirations}{violence}There were a lot of
     fights{/violence}, and {truancy}kids, uhm wandering
     around{/truancy}, and {aspirations}most of us worked
     in factories on the [city's] east side{/aspirations}.
     {gratification>delayed}Most of us partied rather than
     worked.{/gratification>delayed}{/school>trouble}
```

The second time we would get the following two orange passages returned separately.

```
R:  {school>trouble}Well, my highschool was known as
     a trouble school. {aspirations}We weren't going
     anywhere. {/aspirations}{violence}There were a lot of
     fights{/violence}, and {truancy}kids, uhm wandering
     around{/truancy}, and {aspirations}most of us worked
     in factories on the [city's] east side{/aspirations}.
     {gratification>delayed}Most of us partied rather than
     worked.{/gratification>delayed}{/school>trouble}
```

If you want a count of intersections of the two codes rather than a count of how many of coded passage X also has some of Y, you should do a search from the workbench with the + sign: aspirations+school>trouble.

VI. Concluding remarks

At this point you have some idea of how to search, sort, and select. You should also be able to designate some codes as context codes to identify contextual information. From here some simple additional things to learn include personalizing the toolbar on results windows, adding and viewing comments in results windows, and exporting data to the clipboard so that it can printed in Microsoft Word, Apple Pages, or the word processor of your choice. An even more advanced step would include learning about “reanalysis,” i.e., transforming your data files based on your results files. This includes recoding your files (actually changing the code of a given passage) and adding codes around given passages. Finally, there is a lot to be explored in terms of reporting, including simple reports (counts of codes) and complex reports (data summaries and dot graphs).These are all described in the user guide. Have fun exploring this complex but powerful software program.