

TAMS Analyzer 4.0

User Guide

The screenshot displays the TAMS Analyzer 4.0 interface. The main window shows a code graph with three nodes: "positive reason", "negative reason", and "4s reasons". A node "M" is connected to these three nodes with arrows and numbers: 8 to "positive reason", 7 to "negative reason", and 48 to "4s reasons".

Below the graph is a table with columns "#", "time", and "code". The table contains 18 rows of data:

#	time	code
1	0:00:21	
2	0:00:21	
3	0:00:21	
4	0:01:28	
5	0:01:28	
6	0:01:28	
7	0:01:28	
8	0:01:28	
9	0:01:28	
10	0:02:51	
11	0:02:51	
12	0:02:51	
13	0:02:51	
14	0:04:10	
15	0:04:10	amy F Amy
16	0:04:40	amy F Amy
17	0:04:40	amy F Amy
18	0:05:47	amy F Amy

Two comparison windows are overlaid on the main interface:

- Comparison of gender by Code sets:** A table comparing "gender" against "Code sets".
- Comparison of FileName by Code sets:** A table comparing "FileName" against "Code sets".

The "Comparison of gender by Code sets" window contains the following table:

gender	negative reason	positive reason
F	19	48
M	7	8

The "Comparison of FileName by Code sets" window contains the following table:

FileName	negative reason	positive reason
Amy	and it takes a long time for kids to read books	goofie chemistry songs?
	My coop teacher right now tried to ask her administration if she could do one. They wouldn't let her, it was a really low reading level too. It would have been real quick on nuclear chemistry. They said, what are	amy: yeah but they remember them get stuff out look at nasa done, couple years uhm like I used october sky. my actually teacher I saw in inner city used that as intro to phys unit, sh e questioned them on diff things they got into it ii think that shows scientific method, how kids could do it. but

Revision 14—Updated for 4.00
August 14, 2010

This documentation is really just to get people started and give them an overview of what is necessarily a complex system. This is not comprehensive documentation. The best source of documentation of every feature is the release notes. However, after many complaints I've been convinced that the existing and out of date documentation needs to be supplemented.



(cc) 2008, 2009, 2010 Matthew Weinstein; Some rights reserved

Icons:

(cc) 2010 Lukas Birn; Some rights reserved. Creative Commons Attribution 3.0 Unported License

TAMS Analyzer User Guide by Matthew Weinstein is licensed under a Creative Commons Attribution-Share Alike 3.0 United States License.

Published by Mayday Softworks

Contact information:

Dr. Matthew Weinstein
Assoc. Professor of Science Ed.
University of Washington-Tacoma
1900 Commerce St.
Tacoma, WA 98402-3100
mattheww@u.washington.edu
<http://faculty.washington.edu/mattheww/>
<http://tamsys.sourceforge.net/>
aim: allemandel3ft

Chapter 1: What is TAMS Analyzer

TAMS Analyzer (TA) is a software program for coding and analyzing qualitative, textual and audiovisual information such as interviews, observations/field notes.

A. TAMS

The analysis of a document is done by you, the reader-ethnographer, in this program. TA just keeps track of (actually embeds) the information you indicate. You read the document, select sections and indicate what such a selection represents.

TAMS stands for text analysis mark-up system. It's sort of HTML-ish or XML-ish, but it is very distinctive. People have asked why I'm not using XML, and my initial response is that multiple independent ways that we (qual. researchers) have to analyze texts doesn't work easily with XML which, for instance, doesn't allow overlapped sections. To just make clear that I am not using XML or any other standard, I use “{“ and “}” to mark my tags. At some point someone (maybe me, maybe you) will create a TAMS to XML converter.

B. Coding

TA's first job is to help you code, that is to mark sections of documents as to their significance. Whether importing documents (TAMS can work directly with rtf, rtfid and text documents) or creating them in TA your first job is to select text or parts of images and indicate what they mean.

C. Analyzing

TA's second job is to extract information from a marked up document. Basically TA just compiles a table of the selections meeting specified criteria. This is called analysis. After compiling this table TA let's you search the table and generate summary statistics (counts of how many records meet such and such criteria) and cross comparison tables. It also lets you use this table to change the codes in your original source document.

D. License issues

TAMS is released under the GPL license, the text of which is available

at www.gnu.org. At some point I'll do the tedious work of including a statement in every source file regarding it; there are also some parts of the program which are released under Apple's License which is not as liberal as GPL; so be careful (in particular the parts of the program concerning the find text dialog box, the selection rectangles, and PDF management).

Chapter 2: Getting started with a project

When you first launch TAMS Analyzer, you are presented with the “New Project” dialogue. This permits you to name and locate your project. **Starting with TA3 the location of files is tightly controlled by TAMS Analyzer.** In previous versions, new projects, documents, and result files would start out as “Untitled” and you would save the documents in the typical way. With TA3 there are no untitled documents (ignoring some report windows). Instead, before you are presented with the window you are asked by TA3 to name the file. With document and results windows the project saves them automatically in appropriate locations. This is necessary for future improvements as well as for the multi-user capacities that TA3 adds to the program. This is what the “New Project” dialogue looks like:

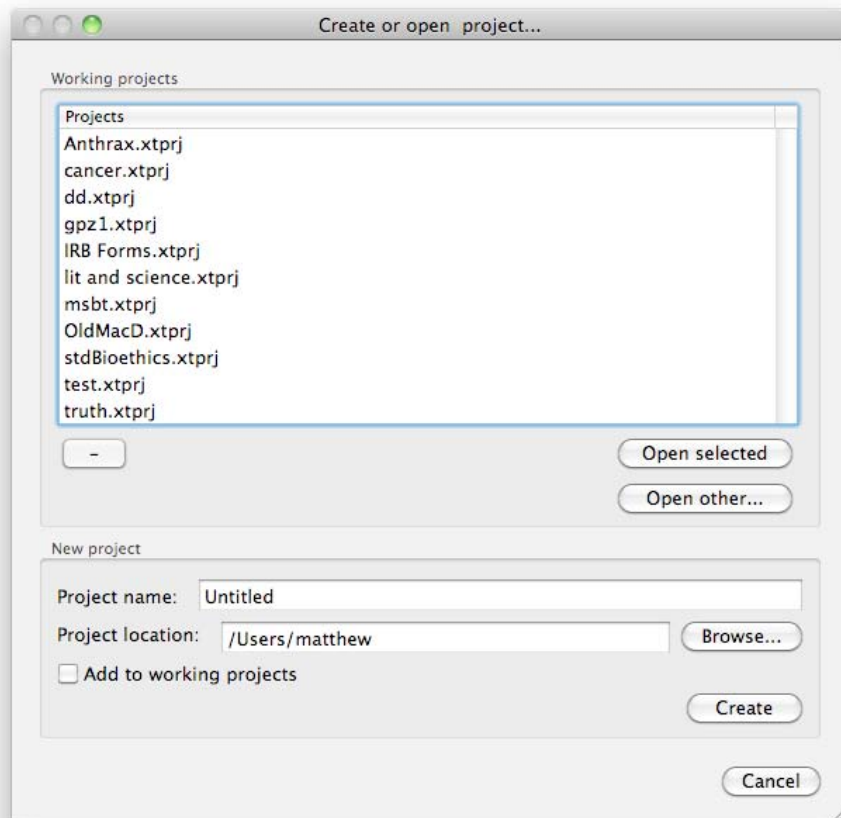


Figure 2.1. New Project dialogue

Notice that “working projects” are listed above. You can add your projects to

4

the working list by checking the “Add to working projects” checkbox when you create the project, or by using the “ File->Work->Add project to work menu” menu item when your project is the front window.

To create a new project replace “Untitled” with a project name in the above window: “Cooperative learning” or “Cancer narratives” or whatever is appropriate.

The second line indicates in what folder the project should be located. Use the Browse button to pick the appropriate folder. Once you have you can see the folder’s location next to the browse button. But you should know that TA3 will create a folder (with a .tams extension) and put the project file and create the necessary subdirectories all within the chosen folder.

As noted, if you wish to automatically add this project to the “work” menu, check the “Add to work menu” box before hitting create. Note that this will add the project both to the list of projects on the New Project dialogue (next time you start the program) and to the File->Work submenu.

Once you hit the create button you will see the project window also known as the workbench.

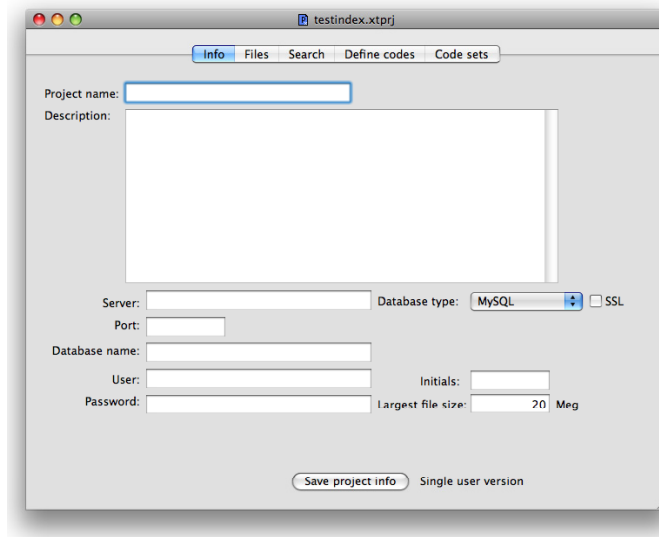


Figure 2.2. The project window (aka a workbench) for the single user version

A project consists of a series of files each of which usually represent one thing: an interview, an observation, a memo, etc. It is through this window that you will add files to your project, remove files from your project, and open files you have already added to the project, as well as do multi-file searches. This window also holds all of the codes and definitions that are in your project. The key functions for managing these things are distributed across the tabs that line the top of the workbench. For example, clicking the file tab reveals functions and information for adding and removing files from the project as well as for selecting the files that you will search when you go to mine your data.

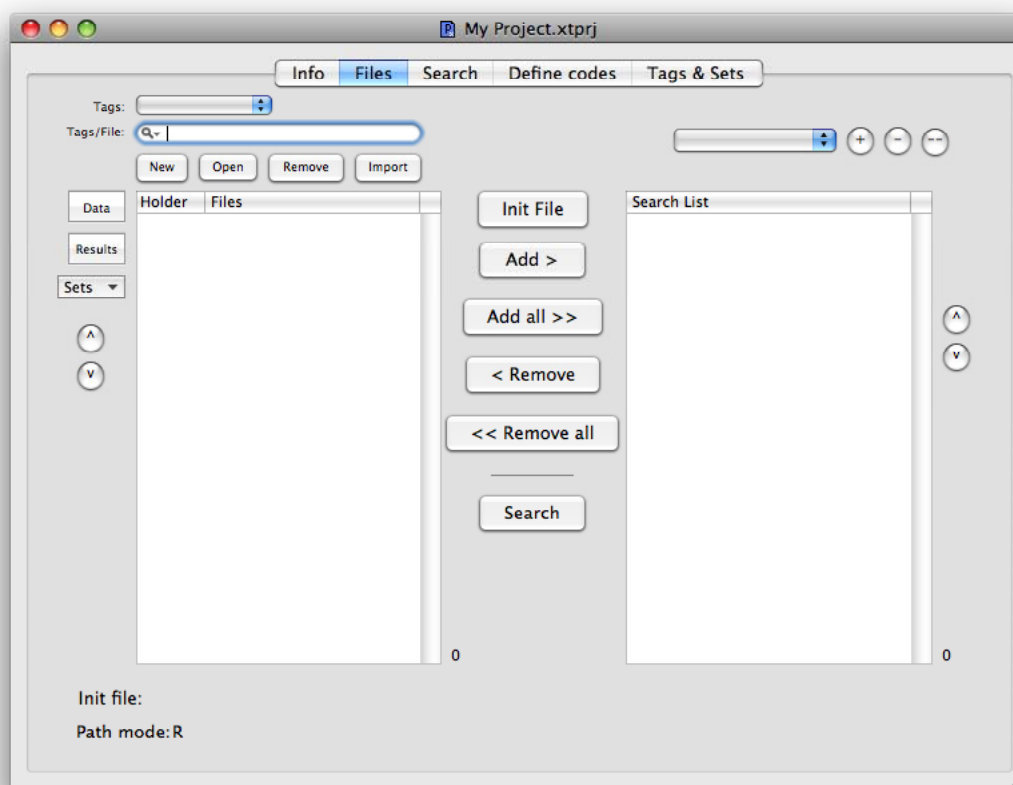


Figure 2.3. The file tab of the work bench

The file tab provides the interface for adding new data files to the project. Here are some ways of putting together a project:

A. Case 1: the files already exist

If you have already typed in your interviews, save them as RTF documents. Click the small “Import” button over the “Files” list view on the left

6

side of the project window and use the standard open file dialog to select the files you want to add. The files will be copied into the project's data directory and then appear in the "File" list. They will not be open. You can add text (.txt), RTF and RTFD (.rtf and .rtfd) files to your workbench. Extensions are necessary for TAMS to know if it can open the files.

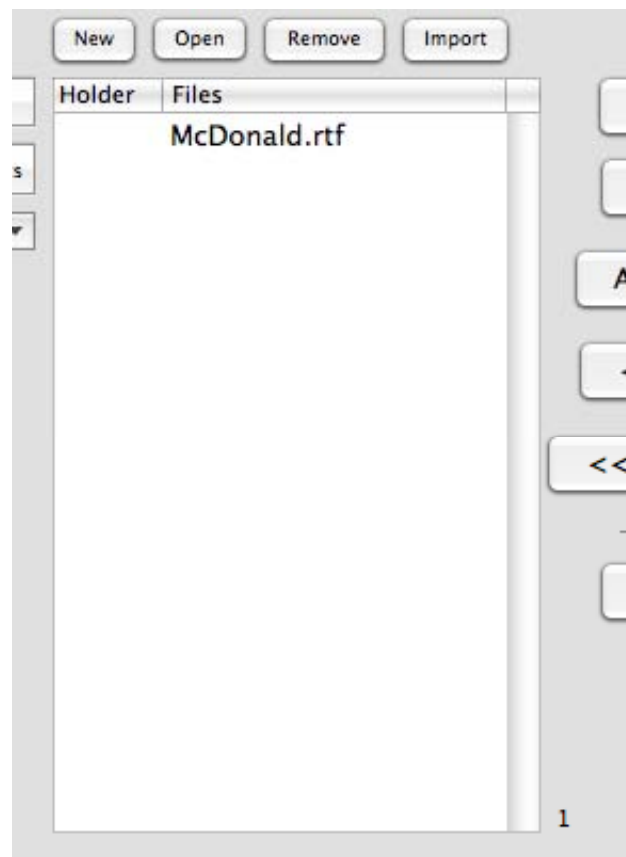


Figure 2.4. File added to a project

B. Case 2: creating data files in TA

When you create new files in TA, you will be saving them as RTF files by default. You can also save them as text files or rtfds by picking the file type from the small pop up menu on the dialogue where you give the file its name.

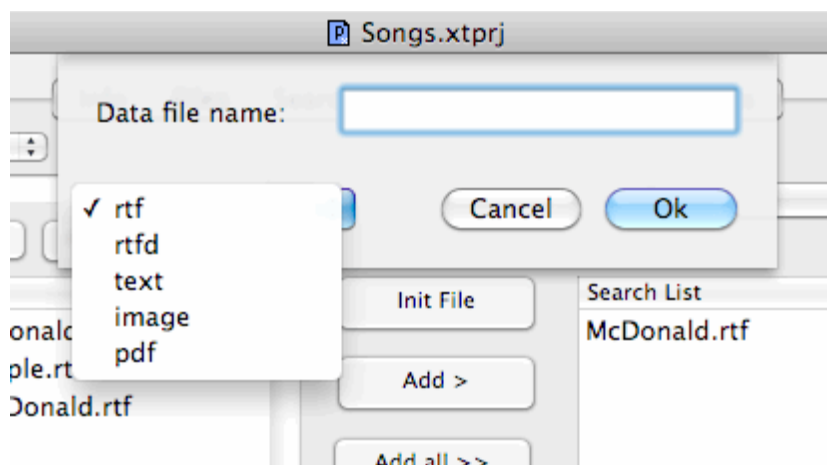


Figure 2.5. Naming and selecting the file type for new data files

To create a new file, simply click on the “New” button over the file list, give the file a name (no extension), choose the file type, and a new document will be opened for your immediate use.

C. Saving & restoring projects

The overall details of a project are saved in a project file: a saved representation of the project window. This file includes, among many other things, the location of the files that comprise the project, a list of files to be searched through for information, and, vitally, the codes that you have created. Unlike all the other windows that TAMS provides, project files automatically save themselves as new information is added.

Never open a document (i.e., an interview, observation, etc.) or results file directly (i.e., through the finder). Instead open the project and then double click the document you want to see from the file list.

Open a project by simply double clicking a project file, a file which has the extension “.xtprj.” To add, delete, and create source/document files (interviews, memos, etc.), use the menu items on the *Project menu* (not the ones on the File menu), or the buttons above the file list.

Chapter 3: Coding

A. What is a code

A code is a name that identifies the meaning or significance of a passage of text or part of an image or PDF page. My focus here is going to be on text files. Working with images and PDFs is a fairly straightforward extension of the text idea. In TAMS the text passage is surrounded by tags that have the code and other information with it. Codes can be nested and overlapped without problem.

1. Valid characters

The names of **codes can have letters, numbers, and underscores (“_”)**. **They cannot have spaces.**

Codes can be hierarchical, i.e., you can create a whole family of codes, indicating the various levels with “>”. For instance, to create a “food” family with carrot, parsley, and cilantro in it, you would name the codes

```
food>carrot  
food>parsley  
food>cilantro
```

carrot, parsley, and cilantro are subcodes of food. Note that TAMS is case sensitive. Also you can still use food (no subcode) as a code.

You could specify further levels of coding such as

```
food>parsley>curly  
food>parsley>italian
```

2. From codes to tags

In your text to indicate that something is coded you surround the passage with “tags” which contain the “code”. Showing is easier than telling in this case. Say that you are going to code the following passage in your document.

Parsley makes me sick.

To do so in TAMS you just surround it with tags containing the code:

```
{food>parsley}Parsley makes me sick.{/food>parsley}
```

The end tag must begin with a slash, the front tag must not have a slash, just like HTML. Every open data tag must have a matching close tag. Your Coding menu has a couple of diagnostic tools to help you find “bad” or missing tags. Note here you can see that tags *contain* codes but are not the same thing as codes. They have that other stuff (“{“,”}” and “/”) as well.

Note you could just type all that junk in, but what would be the purpose of my program? In TA you select the text and either pick the code out of a list or type it in a box on the side of the document (if it’s a new code). More on this below.

3. Signed tags

To support multiple coders, TA 1.0 introduced a new syntax that added a signature to a tag. A signature is a group of letters (no spaces) that are your handle for coding. These are stuck in brackets after the code inside the tag. If my handle is “mgw” then I could sign the passage by coding it as

```
{food>parsley [mgw]}I hate parsley.{/food>parsley [mgw]}
```

note the code and the signature must match in both tags!!!

Again, it would be silly for you to type all that. You indicate that you want to sign your tags in TA’s preferences dialog on TAMS Analyzer menu and by filling in initials on the work bench Info tab.

In TA3 and later signed tags take on additional significance as the program has added a whole new layer of support for multiple coders, which allows researchers to share their work. **Your initials in TA3 are also your login to the database that allows you to share your research.**

4. Tags with comments

Sometimes you want to leave yourself a little memo about the passage.

TAMS does this by allowing you to leave it in the close tag after the signature (if there is one). The memo is offset from the code or signature by a space.

```
{food>parsley [mgw]}I hate parsley. {/food>parsley [mgw] This
guy's crazy!!!}
```

You can also insert it with a colon after the signature (or code if there is no signature)

```
{food>parsley [mgw]}I hate parsley. {/food>parsley [mgw]:
This guy's crazy}
```

which makes it look a little nicer.

TA can facilitate adding comments in text documents by holding down the option key while double clicking the code or clicking “Apply code” Now you can see that tags have a lot more than just the code, they also include signatures and comments.

Advanced note: You can also attach a comment to a region of your document that effects all codes within that region through the `{!setcomment MYCOMMENT}` and `{!endcomment MYCOMMENT}` metatags.

B. Adding a new code

TA makes it easy to add a new code.

- First, select the text that will be coded.
- Second, just fill in the name of the new code in the box on the left side of a document window. **Do not hit return. That does something very special in TAMS, called creating a hot code list.**
- Third, press the button marked “Apply code”.

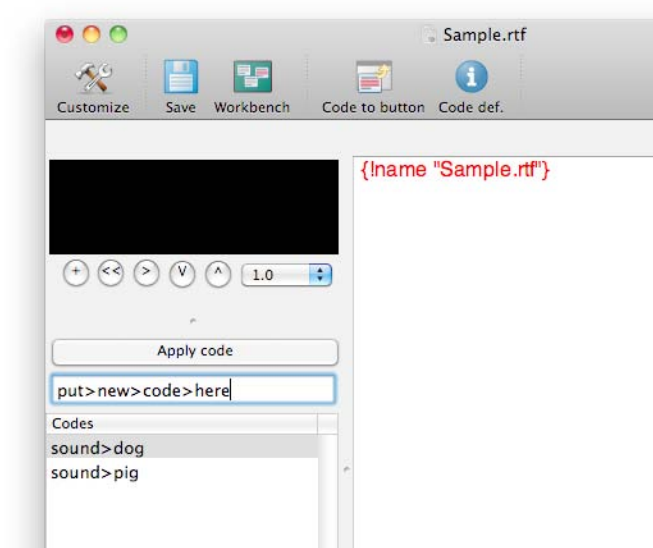


Figure 3.1. Entering new codes

Then you'll be prompted for the definition of your new code:

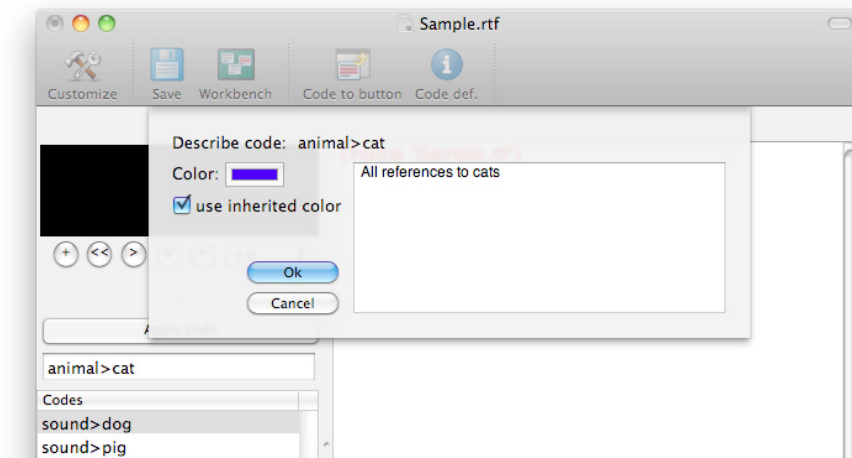


Figure 3.2. Code Definition

When you click ok, three things happen:

1. your code and definition will be added to the code dictionary in the project workbench (which you better save!)
2. your code will be added to the codes list under the box you typed your new code name into

3. as noted, your code will be applied (i.e., surround) the selected text.

C. Applying an existing code

Now things get really easy. If you already have the code in your list, just select a passage and double-click the code as it appears in the code list.

So in this example just double-click food>parsley to code the selected text.

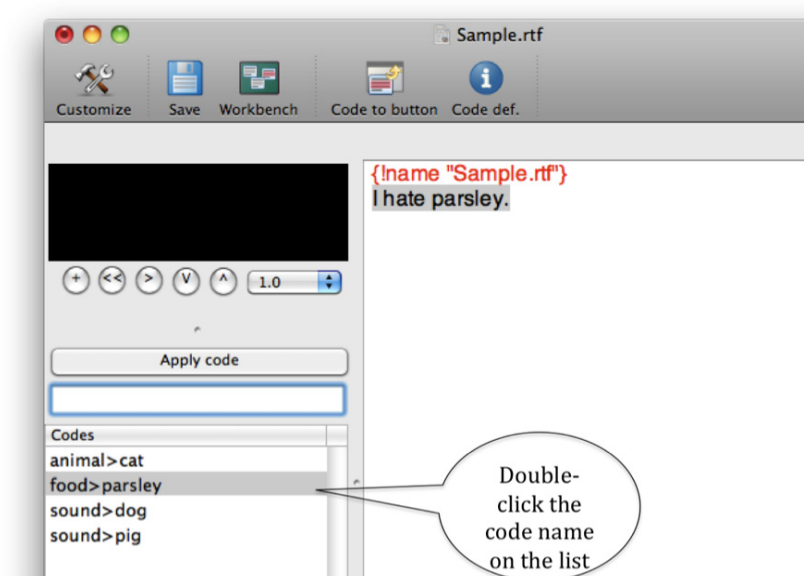


Fig. 3.3. Coding

After double clicking the choice in the “Codes” list on the left side of the window, this will look like this.

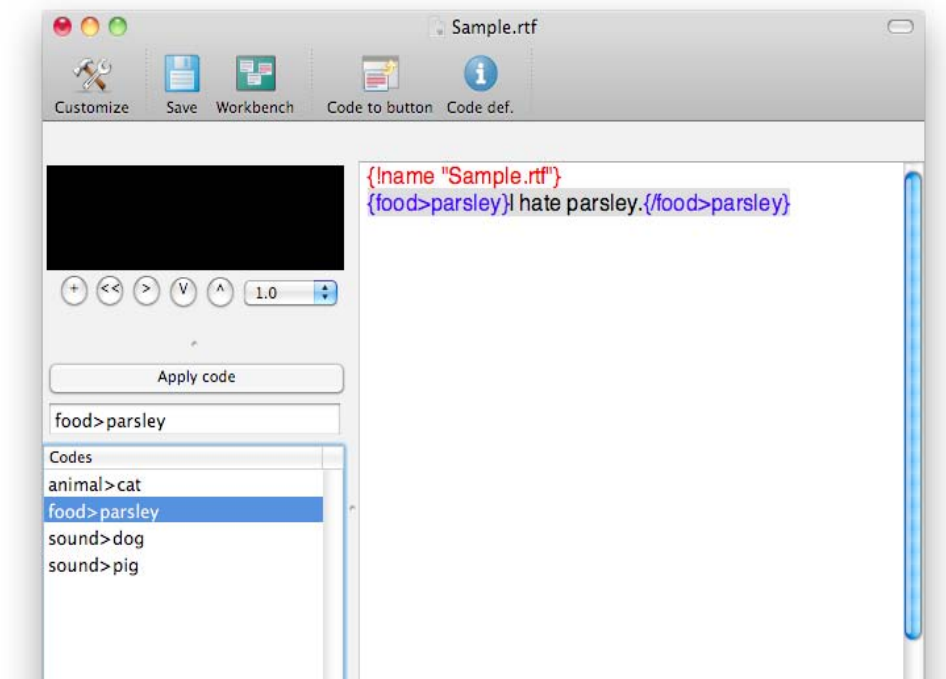


Figure 3.4. After coding

Notice that the text is still selected so you could keep applying codes to this section of text!

ADVANCED:

When you have a lot of codes, it's convenient to have a couple of frequently used ones on the tool bar. To put a code there select a code from the code list (click on it one time) and press "Code to button." This will add a button with that code

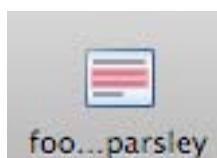


Figure 3.5. Code button

To use these buttons select some text and **single click** the button. Notice that the program abbreviates the full code name. Float the mouse over the button to reveal the full code.

REALLY ADVANCED:

These buttons are not permanent. Every time you close the window, they will go away. What if you want the button bar to come up each time with certain codes. Put a metatag at the top of your document that lists the codes you want on the button bar, you can also have text and insert vertical bars as well:

```
{!button food>parsley, |, “{!end}”}
```

The first time you type this in you will need to pick “Build button bar” from the Coding menu, or you could close and reopen the file.

This example will create 2 buttons separated by a vertical bar. The left button will be a coding button that will code selections “food>parsley”. The other button will insert `{!end}` when clicked. Note that this second button has quotes around it. They signal that this is not a code. You have to use straight quotes, not curly quotes for this to work.

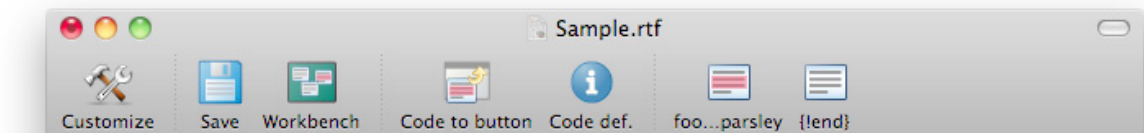


Figure 3.6. The resulting tool bar

This weird syntax with the ! is explained in III.E. below. For full documentation of the document toolbar see the Tool Bar HOW TO in the how to folder.

D. Working with tags and codes

To help you work with tags TA provides some very simple tools to select and move tags around as well as to delete tag pairs and leap from the open tag to the close tag of the pair (and vice versa). These are all on the Coding menu. I wont walk you through them, they should be pretty obvious.

The one practical piece of knowledge that I will share is that I often find

the need to move the end tag of a pair to a different location after I find that the next paragraph should also have been included. No problem. To Often you need to slightly readjust the location of a tag, move it over a word or sentence or two. To grab a tag for relocation, click anywhere in the tag, pick “Find current code” from the Coding menu. This will select the tag. Now you can easily drag it to its new location.

E. Using the Define Codes Tab of the Workbench

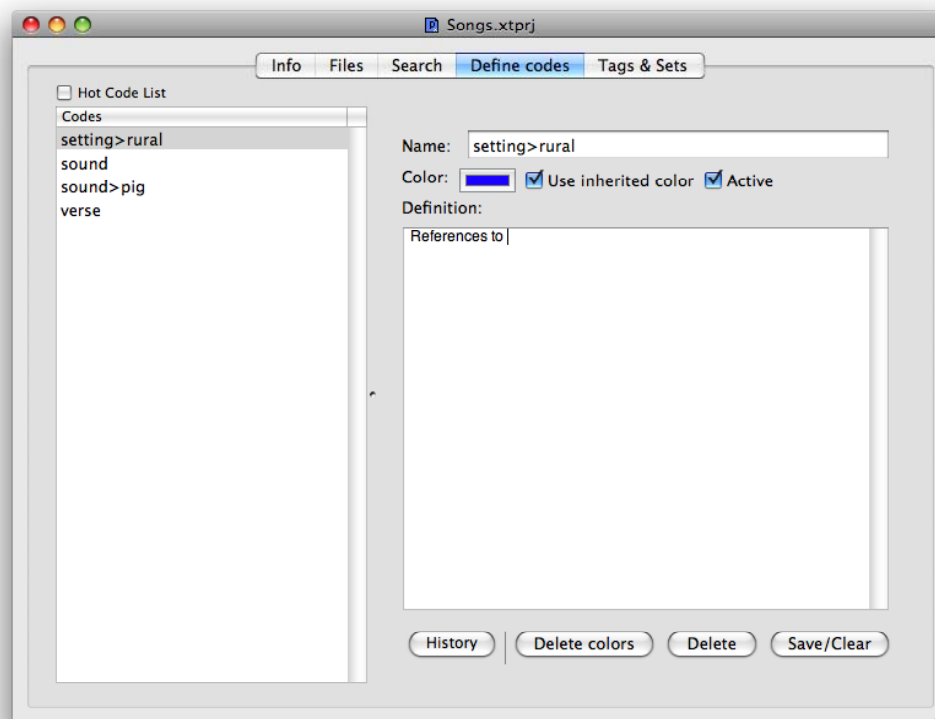


Figure 3.7. The define codes tab

To allow you to modify and amend, set colors for tags, as well as indicate which codes are active and inactive, TA3 has a special tab on the workbench for working with codes (a similar set of functions is available through a dialogue called the code browser which you can access through the “Project->Code browser” menu item). One thing you will notice is that there is no save button. The fact is things are saved every time you do anything whether it’s click clear, click a different tab, or select another code. The program is always saving your codes. So if you hit “Save/Clear” it will save the codes and definitions you have just entered. To get rid of a code, use the explicit “Delete” button.

F. Universal codes and metatags (sometimes erroneously called metacodes by me)

The types of codes we've been talking about are data codes. They specify the meaning of some portion of text. In comparison, universal codes describe a whole document rather than a section of it. For example you may want to indicate that the type of data you are dealing with in this particular file is an interview. You could put at the top of the document the following to remind yourself of this in the output:

```
{!universal dataType="Interview"}
```

This will produce one column in your output called "dataType" and for records from this document it will fill it with "Interview".

This type of tag, which starts with a "!" is called a metatag (rather than a coding tag). It conveys information to the program rather than marks information. There are a large number of metatags in TA all of which are listed in the Coding->Insert metatag submenu. They are described in Appendix 3.

G. Reminding yourself of a code definition.

At some point after 50 or more codes are added, it is useful to quickly see the definition of a code. To do this, pick the code off of the code list (i.e., click one time on the code list of your document window or the workbench) and press the "Code def." button on the tool bar. A window explaining the code will pop up! You can also click on the code lists and pick "Coding->Code definition" from the menus. Alternatively, you can select the code in your document (or even from a results window, which is getting ahead of myself) and pick "Coding->Code definition of selected text" from the menus.

H. Creating code sets

Some times it's easier to work with smaller sets of your codes than the whole body of codes in your project. These smaller groups of codes are

called code sets in TA. The easiest way to access code sets is from the Sets tab on the workbench. In that tab there are two modes (picked from a tab on the left side) file and codes. Click codes. To create a code set follow these directions:

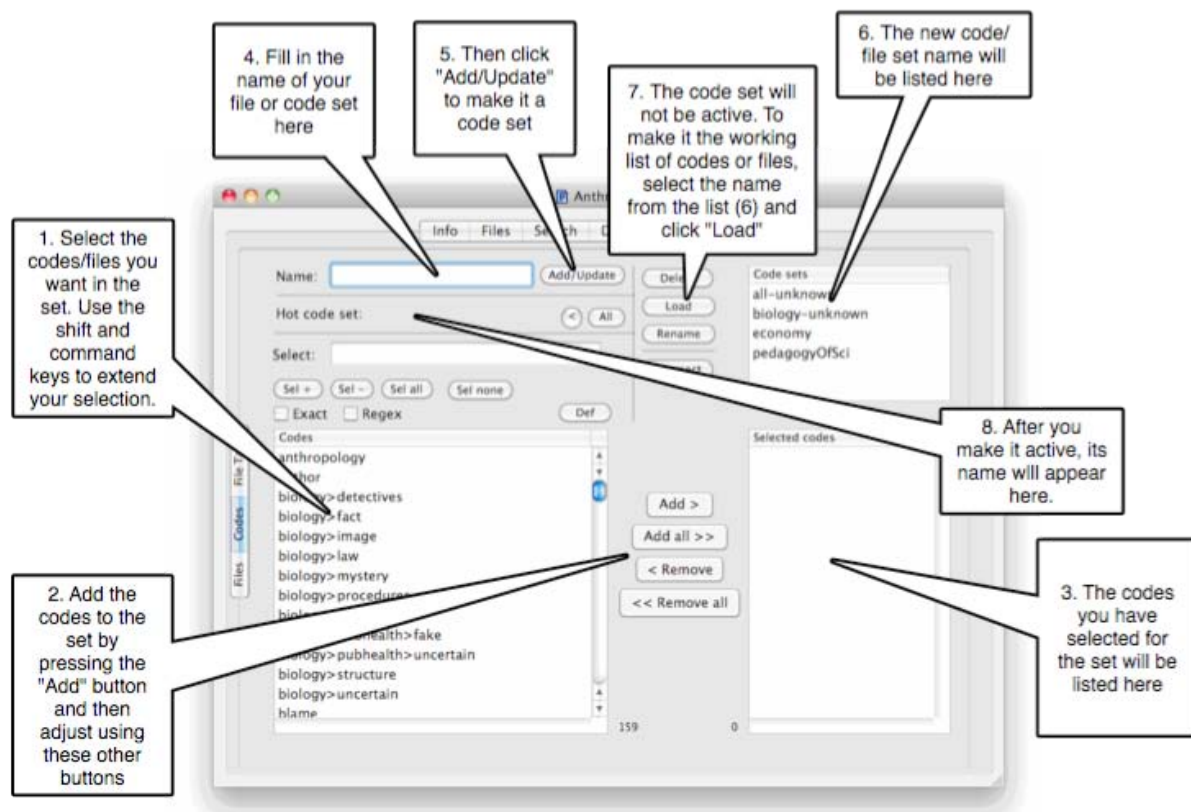


Figure 3.8. Code sets tab

Alternatively, you could create code sets by picking “Manage code sets...” from the Project menu. This provides a dialogue that also allows you to create subsets of codes. In this smaller palette, the procedure is to select the codes you want in the set by clicking, or extending the list by shift and apple clicking the codes; then name the code set by filling in the top blank of the form and register it with the “+” button.

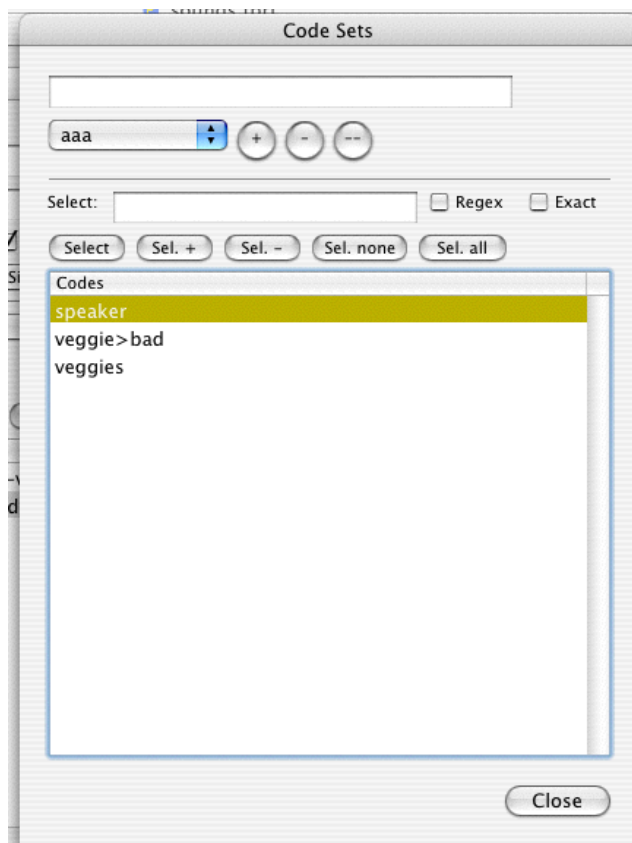


Figure 3.9. The Code Set dialogue

As an alternative to selecting the codes by hand you can use the search criteria to pick more or fewer codes. This way you can select entire code families in a single go. You can also revise code sets and delete them using the menu and the – button. The “- -” button on the dialogue deletes all current code sets. On the code sets tab, the button to delete the currently selected code set is marked “delete”. Be careful.

Creating a code set does not select it. You need to return to the code set menu on the file tab (or the “Project->Code sets” menu) and select your code set’s name or use the load button on the code sets tab.

Once you have code sets defined there are two ways to get information about code sets. First, with your workbench as the front window, you can get lists of which codes comprise your code sets by picking “Reports->Code and code set definitions”. From the dialogue it pops up you can click on the code sets you are interested in and then click “Refresh” to see a report of the code set names followed by the codes that comprise them. If you want to have the definitions for codes provided as well, click the

definitions check box before hitting refresh.

Note: To get a report of your codes and definitions use the “Code and code set definitions” report dialogue but leave all code set names unselected before clicking the Refresh button.

For a more graphic image of how codes are related to code sets and code sets are related to each other through the codes they share use the “Reports->Graph code sets” with the workbench in front. Dot graphs are discussed in section VII in great detail. Just know here that your options include a map (like a concept map) of codes to code sets or code sets to each other, indicating which codes overlap, showing all of the codes or only the non-overlapped codes in the boxes representing the code sets, and matching codes to code sets exactly or non-exactly. See section VII for more details on these reports.

Code sets are not just a convenience feature. Many elements of TAMS are linked to code sets. Count, co-frequency count, “Graph data” output all look at the current code set to generate their output. In addition you can look for records from a code set within your results window. See section VIII for more on analyzing your results using code sets.

I. Hot code sets

Hot code lists are temporary code sets created in document windows or on the search tab of the workbench. They allow quick access to relevant codes in projects that may have hundreds of codes total. They are stored in the project window, but are not saved with it. They can however be turned into regular code sets which are saved and restored with your project.

To generate a hot code set from a document window type in a key word or a part of a key word or code name into the code text field (above the code list) that you are interested in and hit return:

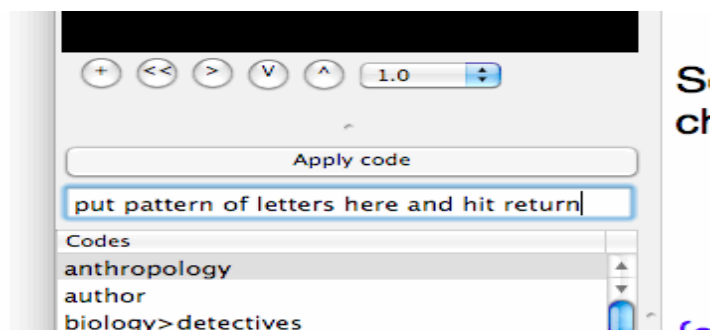


Figure 3.10. Creating a hot code set from a document window

The code list will show only those codes that contain this word and those whose definitions contain that word/phrase/pattern (the latter is true only if you have the “Check definitions when selecting codes” preference panel item checked).

Note 1: The “Select +” button allows you to type in other patterns of characters and add matching codes to the current hot code list

Note 2: If you hit select (or hit return) with the code field empty, you will see all of your codes.

Note 3: To refine your hot code list use the menu options under Coding->Select codes menu item:

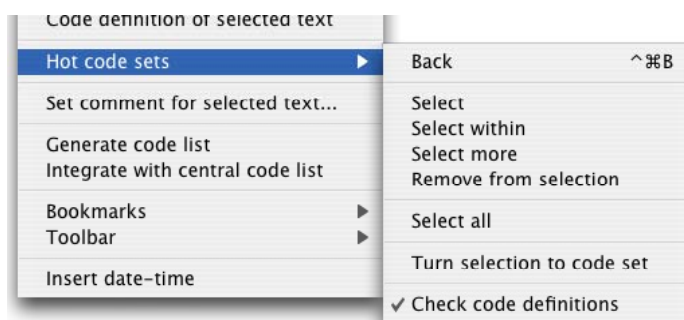


Figure 3.11. The hot code set menu

The group of four items under “Back” allow you to select (from the total pool of codes), refine the selection you have, add from the total pool of codes to the hot code list, and remove items from the hot code list.

The “Back” menu item restores the last hot code list you made and then as many previous ones as you have allotted with the preference panel item which indicates how many previous hot code lists to save.

The second to last item, “Turn selection to code set” allows you to turn your hot code list into a proper code set. You’ll be prompted for a name under which to save it.

The last item is a toggle that lets you indicate whether or not TA3 should check the code definitions as well as the code names.

Note that all of these same functions are duplicated under the search tab of the workbench:

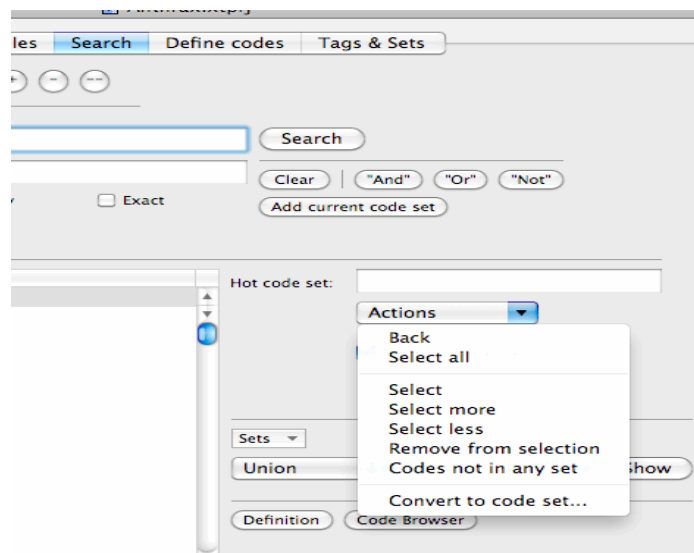


Figure 3.12. Creating a hot code set from the workbench search tab

Advanced user note: The searches that are used to compose hot code lists are regular expressions. This allows you to use the full range of regular expression technologies in making your selection. For instance if you want to find all codes with fruit or vegetable in part of the name (and definition) enter “fruit|vegetable”. The vertical bar above the “\” key is the “or” function in regular expressions. The searches are case insensitive and use the multiline mode provided by AGRegex (this assures that ^ and \$ work to mark the beginning and end of a line, so that you can find codes that start with a by typing in “^a” or that end with a by typing “a\$”).

J. Sectional coding

Structured documents allow you to indicate the codes that apply to an entire section. This is very useful, for instance in the following situations:

1. Tagging memos. Often in a memo, you want to just provide a list of tags that can act as codes for the entire memo.
2. Coding highly structured data. Often projects don't involve coding portions of a text but entire objects. Think of creating cards with items you are coding. Each card represents one something, and you want to code the cards, not the parts of the something on the cards. Again, you are basically tagging the cards.

To accomplish this two use the `!setcode` and `!setcodeinfo` metatags. The simple version is `!setcode`. The format is

```
{!setcode code1, code2, code3}
```

It will code the entire section starting at the point of the `!setinfo` tag with `code1`, `code2`, and `code3`, **none** of which should be context codes. These will be data codes applied to the sections. For example, a memo file might look like the following:

```
{!context date}
{!last date}

{date}1/12/2009{/date}
{!setcode a, b, c}
```

This is the text of my first memo.

```
{date}2/1/2009{/date}
{!setcode b, d}
```

This is the text of my second memo.

This is basically the same as

```
{!context date}
{!last date}

{date}1/12/2009{/date}
{a}{b}{c}
```

This is the text of my first memo.

```
{/a}{/b}{/c}
{date}2/1/2009{/date}
```

```
{b}{d}
```

```
This is the text of my second memo.
```

```
{/b}{/d}
```

Obviously, the first is easier to read, and serves as a simple way to apply a list of tags to your memos, and include them in your results windows.

!setcode takes its horizon (the point at which the codes expire) from the same horizon as !last and !inner. By default it is !endsections, but it can be changed through the !virtualend or !virtualendsection. There is no “coder” added to these codes.

To allow more control, there is a second tag !setcodeinfo which takes the form:

```
{!setcodeinfo codes="code1, code2, code3"
coder="myinitials" horizon="end or endsection"}
```

Here you can control both the coder’s initials (which allows you to have multiple people tag items) and set the end point for the tags (either end or endsection, not both!)

WARNING: There are limits to coding this way. First, addcode, recode, and all the other functions in the Results->Recode menu will not work or worst do damage to the documents. Also, setting codes in tags opens up all sorts of possibilities for mayhem, especially nested codes. There’s no way of having TA pre-check for nesting in this case, but you will get lots of error messages if the codes and horizons land up conflicting! There’s also the very dangerous possibility of using a context code in one of the tags which will lead to at best strange behavior and at worst crashes!!! I’ve tried to anticipate problems as best I can, but in my experience, TAMS users are more creative than my attempts to anticipate them.

To assist you in adding codes to your !setcode tag, you can drag and drop codes from the code list in document windows right into your transcript/field notes; this will insert the name of the code in the text. There is also

a menu item which does this: Coding->Insert code name, which will take whatever code is selected and insert it at the cursor. This way you can use your code list as the source of your tags.

K. 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. TA is saying that for some reason, it can’t find the other end. **Bug notice: Sometimes the program can’t see the end of a tag at the end of a document, i.e., if the last character of the whole document is a “}”. Save yourself headaches and put a return after the last character in your data documents!**

2. Incomplete codes: Sometimes in 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, or even if it was done by a different coder (with a different signature);. The problem is that TA can't figure out where the passage ends, and it will choose the shortest passage. In this example, the phrase "trying to code" is not seen by TA. These problems can be found by picking "Check for nested" from the Coding menu.

One type of error that TA is not good at catching is missing close braces. Often times if you're getting an error message while doing a search, this is the source of the problem.

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

L. Coding PDF documents

The PDF coding window is a wrapper file for PDFs. What this means is that you have one file that holds the coding information separate from the PDF file itself. When you create a new file and pick "pdf" you create one of these wrappers. The interface is shown in figure 3.13.

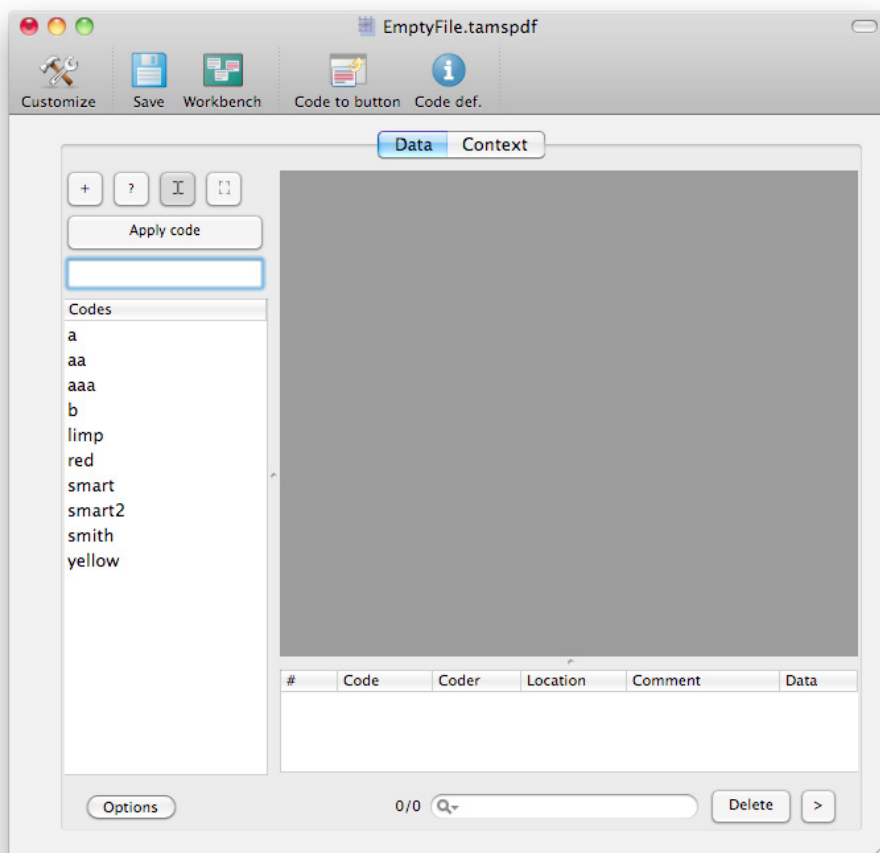


Figure 3.13

Most of this interface works the same way as text documents. There is an existing code list on the left, a field to enter new codes, and a large “Apply code” button. While the coded sections are shown in the pdf, they are stored in a database shown below the pdf itself. The large gray square will eventually show the PDF itself. To attach the pdf, click the “+” button above the “Apply code” button. You will then be able to locate the file and attach it to this wrapper. The PDF will be copied into the media folder of your TAMS project. Here is the window with an attached PDF, and with the shelf shown (opened with the small “>” button in the lower right corner).

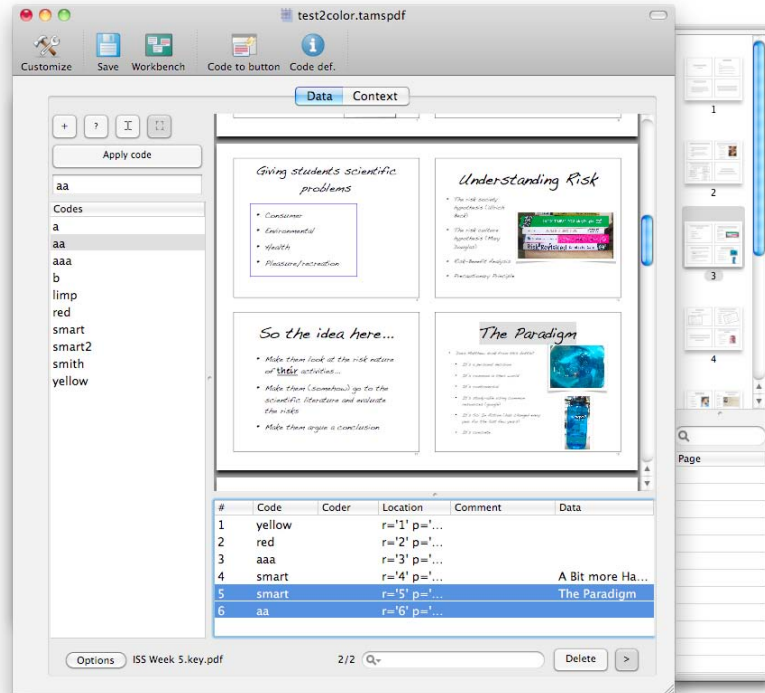




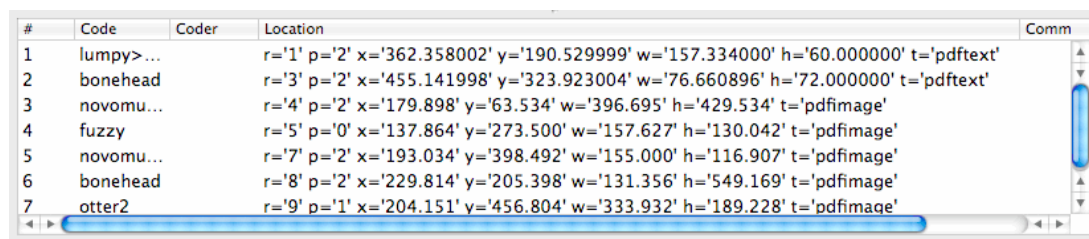
Figure 3.14. PDF document loaded

Below the PDF is a table that shows the current coded data, the coded selections table. Selecting a row (or rows) will highlight and scroll the PDF to the selected code (row).

There are 2 ways to code a PDF: by image and by text. The  button (or image selector) is used to select and code rectangular sections of PDF pages. The  button (or text selector) is used to select sections of text. Not all PDF files have support for text. Even PDFs that have selectable text may not have it layed out in a useful way. Some PDF text seems to jump columns, other PDF text returns each word as a separate line. You can edit/replace/adjust the text that TAMS finds by double-clicking the row. An editor will show you the content of that row and allow you to adjust/replace what TAMS finds as the selected text. Coding using the text selector returns the string that OSX provides for that selection as the relevant data. For image selections, TAMS provides a blank record. You should fill in some sort of description of what you have coded by clicking in the description column or double clicking the record. You will probably need to double click the new code row and edit the Data and Comment fields.

The PDF Wrapper interface provides multiple ways of searching your data. You can use the search field on the drawer to look for text in your PDF. You can use the search field at the bottom of the window to search through your coded records. In addition you can select a part of your PDF (using either text or image selection) and click the question mark (?) button above the “Apply code” button to list at the bottom of the window all records that intersect with the selection you have just made. This is important because it is not obvious looking at a pdf, unlike with text files, what codes apply to a given part (in this case a rectangular area) of your data. Note that you can also select all the records in the coded selections table and see what part of your document has been analyzed and which areas have not.

The location column of the coded selections table is worth examining. This field contains a string that TAMS uses to locate this coded selection in the pdf. It contains a unique record number, the page (or pages) of the selection, the type of selection (pdftext or pdfimage) as well rectangle information that describes the selection. The record number is indicated by the field named “r” and will look like `r='3'`, meaning this is record number 3. The pages are indicated in a p field, etc. See figure 3.15.



#	Code	Coder	Location	Comm
1	lumpy>...		r='1' p='2' x='362.358002' y='190.529999' w='157.334000' h='60.000000' t='pdftext'	
2	bonehead		r='3' p='2' x='455.141998' y='323.923004' w='76.660896' h='72.000000' t='pdftext'	
3	novomu...		r='4' p='2' x='179.898' y='63.534' w='396.695' h='429.534' t='pdfimage'	
4	fuzzy		r='5' p='0' x='137.864' y='273.500' w='157.627' h='130.042' t='pdfimage'	
5	novomu...		r='7' p='2' x='193.034' y='398.492' w='155.000' h='116.907' t='pdfimage'	
6	bonehead		r='8' p='2' x='229.814' y='205.398' w='131.356' h='549.169' t='pdfimage'	
7	otter2		r='9' p='1' x='204.151' y='456.804' w='333.932' h='189.228' t='pdfimage'	

3.15 Coded selection table with Location field showing.

Note that the coded selection table can be sorted. Click on the column and then pick Results->Sort up->Smart sort. You can also add a tool bar button for sorting. Selecting on the Location column will sort by page number. Selecting the # column will sort by record number. Selecting any other column will sort alphabetically.

The Context tab at the top of the window provides a text editor. This is where you can enter metatags that apply to this document. Specifically, use this to set the values of !context variables and !buttons. Do not use this to code actual data. While such data will be returned in a results

window, double clicking such results will not return you to the window and highlight the coded portion. It will not select anything, you will just get an error message (or maybe even a crash!). Notice that your codes are unavailable while the context tab is selected. The only coding you should do here is context coding (I repeat). If you want to code something (as a context code) you can select the code while the “Data” tab is selected, and then go to context and pick Coding->Apply code. I’m trying to make it difficult.

Finally, each individual row of the coded selection table can act like its own bookmark for the purposes of inserting references. Select a row and pick “Coding->References->Remember reference” (there is a button you can add to the toolbar for this). Then, in a text document, typically a memos document you can insert the reference to that row. Then pick “Coding->References->Insert reference link” and a `{!goto file=”filename” bookmark=”record #”}` format metatag will be inserted. Click in this metatag and picking “Coding->References->Go to reference” will bring you back to that row.

M. Coding image files.

Most of what I have said about working with PDF files applies to image files. The big difference is that an image is a single page, so there is no shelf, and there is no text to select (so there’s no choice of selection modes). You will always work with a rectangular cursor and double-click codes from the list (or fill in new codes and click “Apply code”). Like the PDF interface there is a context tab for entering !button and !context metatags. Like working with PDF files, you first create a wrapper window and then use the “+” button to connect that wrapper with the actual image (jpg, gif, etc.) file. This file is copied into your media folder.

The one unique feature of the Image Wrapper interface is the ability to describe your image. This involves first switching the tab on the right side of the window to “Description” from “Coding”, selecting portions of the window and then double-clicking in the Description column. This will allow you to enter some text that describes that selection. So if you were analyzing the Mona Lisa, you could select and describe the background, her ear, her smile, etc.

You could then code the image itself (by switching the tab back to

“Coding” and when you do a TAMS search, TAMS will include the descriptions of the parts of the image your coded rectangle intersects.

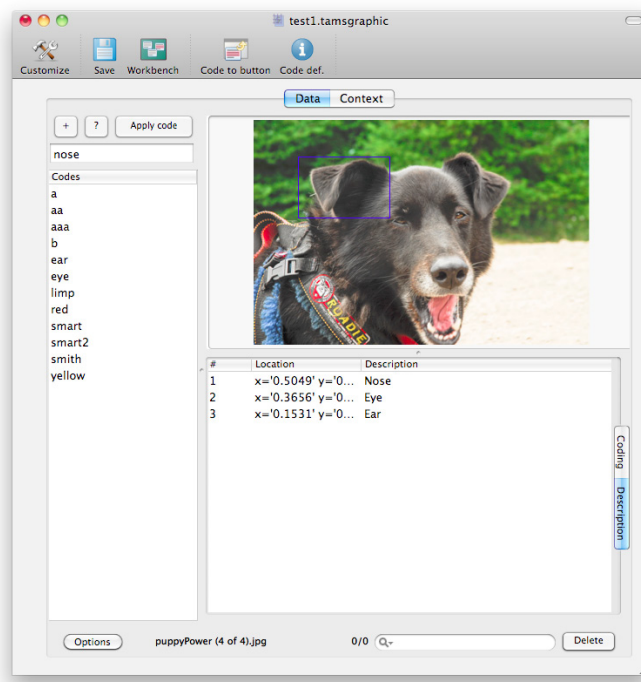


Figure 3.16. The description tab

One other difference between the image (or graphic) window and the PDF window is in how they sort the location column. If you choose to sort the location column in a PDF window, it will arrange your coded selections by page; if you sort your location column in a tamsgraphic file, it will sort the selections from top to bottom of the image.

N. Coding Video

This is nothing new. TAMS has for a long time supported the coding of video through transcripts which use time codes to link bidirectionally with the actual video. That's still the way it works in TAMS 4. The problem has been that the video player has been a tiny thing above the code list. Unless you owned a very large monitor, this was a pretty inconvenient arrangement. Now you can have your transcript open a much more video friendly arrangement by putting a `{!videowindow}` metatag towards the top of your file. When you open an rtf file, the window will look for that tag and provide you with an alternative pane arrangement, that has a large media player over a small transcription area. See figure 3.17.

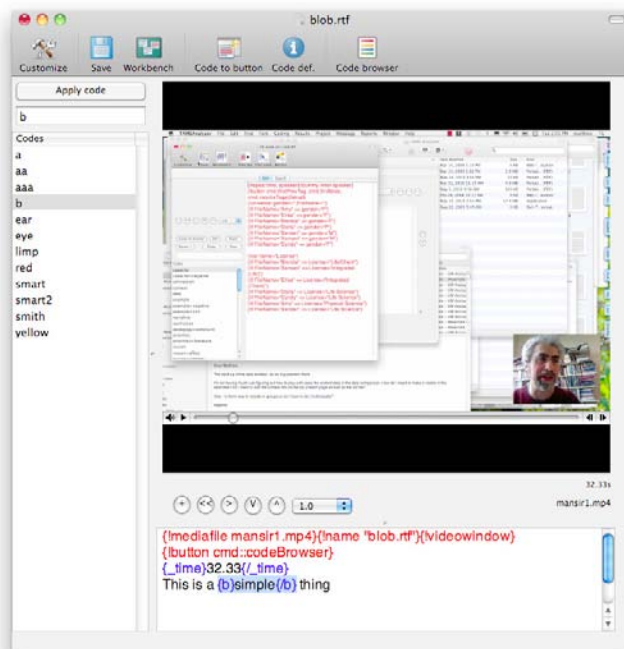


Figure 3.17 Using !videowindow to work with video

Note that after adding `{!videowindow}` to your file you will have to save and re-open your file for the change to take effect. Also note that there are “thumbs” to readjust the code and transcript areas to provide custom working room for video, transcript and code.

Chapter 4: Documents, sections, and context information

A. Context Codes and the Context tag

I will be using the example of an interview here, but we could be talking about field notes in which case time code would probably substitute. In an interview, I always want to know who is speaking when I look at results. (Unfortunately if a coded passage crosses speakers, only the first will be included). To include that information I need to take two steps: first, indicate who the speaker is, second indicate where that information is found.

To mark whom the speaker is just code it as you would any data. We might call the speaker “speaker” for instance:

```
{speaker}John{/speaker}: {food>parsley}I hate parsley.{/  
food>parsley} {!end}
```

Now, speaker is really a different sort of code than food>parsley. One indicates data, the other information you want attached to that data. To indicate that “speaker” is a special sort of code that isn’t data you put a metatag at the top that says that this is what I call a “context code”.

Context is identical to the keyword “repeat”. In earlier versions of TA I talked about Repeat metatag rather than the context. I find the word context more descriptive of what this does, and have edited this section (and the tams code base) to reflect that. Note that !repeat is still supported.

```
{!context speaker}
```

If you have time-code information, you could also add that like this {!context speaker, time_code}, and so on. But see the next section for problems that could arise.

B. Breaking a document into sections

Initially, starting with TA 2.47, TA starts by assuming that your documents are “unstructured” (Actually, this is only true if you have a new installation. Check the default by going to the TAMS Analyzer Preferences, and looking

under the Search tab for the “Documents treated as unstructured” check box. If it is checked then the assumption—unless you use one of the triggering metatags—will be that the document has *not* been structured, else it will assume it has). By structuring a document you enable additional features of TA, especially the section search. Using this search you can ask questions like “which turns in my interview with Bob involved the code food and the code parsley.” If your document is unstructured, TA will not know what “a turn” is in your document. A search will return either the entire interview (if it had those 2 codes in it) or any coded passages meeting the criteria, depending on how you have set up your program preferences. Similarly, empty searches will only work correctly if your document is structured. On the other hand, unstructured documents are very easy to work with and initially will give the results you want in most circumstances: Context codes set the environmental conditions for subsequent data codes through your documents.

If you are updating TA, you might want to turn off this new default unstructured behavior. Easily done. Put the metatag `{!struct}` in your init file (or on top of each interview if you don’t have an init file).

What is the structure of a document? Often qualitative documents have a sort of natural syntax: interviews have speakers, field notes have time coded passages, etc. One reason I created TAMS was so that there was a way to have associated information: the name of the speaker, the time code of the field notes, included with the results of queries into the data. To take advantage of these natural sections, the first thing you will need to do, and TA offers few tools to help with this, is mark the ends of these natural sections with `{!end}` metatags (or `{!endsection}` metatags, see §IV.C.)

Manual structuring: You can manually structure your document by putting `{!end}` metatags after each natural break in your document: After a person speaks, at the end of a time coded passage, or at the end of a newspaper article, for instance. After an `{!end}` TA2 will forget who is talking and what time it is, etc., so you’ll need to have `!context` tags indicate all of that at the front of the next section. Alternatively, you can use `{!endsection}` which carries values of `!context` values forward (i.e., if you think of structuring a document as cutting each part of an interview—for instance—onto 4x5 cards, `{!end}` carries no information to the next card, `{!endsection}` carries everything to the next card.). You may also need to play with various program preferences regarding `!endsections` and `!ends`

in the program preference dialogue (look on the searching tab) to get the results you expect. The original use of an !endsection was to allow users to subdivide a section into smaller parts (e.g., coding letters to the editor. The whole “letters to the editor” article may terminate with an !end, but each letter would get an !endsection).

HINT:

You may want to check out TexEdit Plus for this, both of which have very fancy search and replace functions which can save a lot of time in marking up documents initially.

TAMS also has a very fancy though very technically complex search and replace mechanism known as “regular expressions.” You can use this to bulk code a document often. But the learning curve is steep. There is documentation included in the source folder regarding regular expressions (often called regex) as implemented in TAMS, which uses a programming library called PCRE (perl c regular expressions) to implement them. Also search the web for tutorials on regular expressions. There are a lot of them out there. Powerful stuff.

HINT:

TA will let you turn a passage of text into a tool bar button. After the first time you type {!end} (or pick it from the Metatags->Structure menu) select the tag with your mouse and pick “Turn selection into toolbar button” from the Coding menu. That adds a button to your toolbar that will insert that text when you click it. Then it’s a simple, single click to stick {!end}s where you need them. **Note: See Coding->Toolbar for keyboard shortcuts to your toolbar items.**

Automatic structuring: Rather than manually structure your document, there is some support for having TA structure your document for you. There are two metatags which you can use to save yourself a lot of time and have TA automatically decide where !ends fall. These are the !inner and !last metatags (use one or the other, not both). You use both of these with !context values, which can act as a sort of demarcation of sections in your document (see last section of this guide).

The syntax for !inner is {!inner contextCodeName} and for !last, similarly, is {!last contextCodeName}. These should appear in the init file or towards the top (i.e., before data codes) of your document files.

!inner indicates that each occurrence of contextCodeName should be treated as though it has an !endsection before it.

!last indicates that contextCodeName is the last repeat code being assigned to this section. If TA finds another repeat code it should act as though there is an !endsection before it.

These are easiest to use if only one context value is deployed through the document, or if you use the same context codes for each and every section in the same order. Then you just make the last one the !last.

Advanced note: The way that endsection and end work (or implied endsections with !last) is that no data is collected during a search until it finds that !end/!endsection metatag. Also, context values aren't paired with data until the section ends. This means that if in a given section the context code "time" changes value, only the last value it received will be reported. Consider this case:

```

1 {!context time, speaker}{!last speaker}
2
3 {time}100{/time}
4
5 {speaker}bob{/speaker}: {a}food's good{/a}
6 {time}110{/time}
7
8 {food>parsley}I like parsley.{/food>parsley}
9
10 {time}112{/time}
11
12 {speaker}bob{/speaker}: {food>parsley}it really cleans my
13 breath.{/food>parsley}
14
15 {time}115{/time}

```

Here speaker is our !last code. There are two sections, one starting at line 1 and going to line 6 (since that is the first time a repeat code is found after speaker—our designated !last code) and the second going from line 6 all the way to 15. Again, it's the speaker in line 12 that tells TA that the next repeat code should be treated as if it had a {!endsection} before it. Now consider the value of the time code for each of these: For the first section all data will be reported with time = 100. For the second ALL DATA will be reported with time = 112. The 110 never gets registered because it is changed (in line 10) before the data is written (in line 15). Even the food>parsley code in line 8 will say time=112. The answer to the problem is to put a speaker before line 8 so that when {time} is found in line 10 the data is written, or put an !endsection metatag on line 9:

```

1  {!context time, speaker}{!last speaker}
2
3  {time}100{/time}
4
5  {speaker}bob{/speaker}: {a}food's good{/a}
6  {time}110{/time}
7
8  {speaker}bob{/speaker} {food>parsley}I like parsley.{/food>parsley}
9
10 {time}112{/time}
11
12 {speaker}bob{/speaker}: {food>parsley}it really cleans my
13 breath.{/food>parsley}
14
15 {time}115{/time}

```

The moral: Automation is useful (in fact necessary) but dangerous. Make sure that every intended section has as its last !repeat value the one listed in !last, filling it in if necessary. You may need to hand enter {!endsection}s to cover those places that don't fit the pattern. Also, see !inner in the Metatag appendix, appendix 3.

C. !end vs. !endsection

To reiterate, these are used when manually structuring a document. When TAMS hits an {!end} tag it clears all the repeats that it has found. None of the values will carry forward from the previous part of the document. Using {!endsection} rather than {!end} is one answer to this. It keeps the last values, so that if only a few change in the next section of the document, the previous values will be retained. In the previous example with an interview where you are tracking who is talking and maybe only occasionally entering a time_code you will want to use {!endsection}, but be careful to mark all the speakers, or you will think the wrong people are saying the things you are finding!!! Also make sure that you put in an {!endsection} whenever the value of speaker changes, or you will be seriously misled as to who is speaking.

ADVANCED NOTE: An alternative to the {!endsection} metatag is the {!dirty} and {!clean} metatags which can be sprinkled throughout your document. They handle how {!end} metatags are handled. {!dirty} tells the TAMS processor to carry old values forward when it finds an {!end};

`{!clean}` tells TAMS to zero values when it finds an `{!end}`. By default TAMS assumes that `{!end}`s should be `{!clean}`.

VERY ADVANCED BUT USEFUL NOTE: You can decide whether the `!last` and `!first` metatags, stick virtual `!endsections` or `!ends` through the document. By default declaring `{!first X}` means that every occurrence of `X` is treated as having a `!endsection` in front of it; `{!last X}` means that the next context code that TAMS finds is treated as having an `!endsection`. Recall that the difference is that context codes are blanked out after `!ends`.

To set whether `!last` and `!first` create `!ends` or `!endsections` use the `{!virtualend}` and `{!virtualendsection}` tags in your init file. The program defaults to `{!virtualendsection}` if you do not indicate which you prefer (which was the former behavior of the program). This is particularly powerful when mixed with the new ability to create variables with specific “horizons”.

Super advanced information about how to really control the variables in TAMS.

In versions of TAMS before 3.30, the program kept separate “accountings” of context variables and universal variables. There were two sets of books: one for universals the other for contexts. That is why `!if` statements and `!map` statements had to connect similar variables (context to context; universal to universal); you were not allowed to map a context variable onto a universal variable or vice versa.

Since then, variables are all accounted for in the same log book, which now enables that sort of mixing. More important, the variables can actually switch their type. Basically what TAMS tracks are two properties of a variable with a given name (say `X`). First it tracks the value of the variable (e.g., whether a variable named “sex” is currently set to “m” or “f”) and the horizon of the variable. The horizon indicates what triggers a variable to clear its value. Previous versions of TAMS would only clear context variables at an `{!end}` tag, and a universal at the end of the file (eof). This version allows you to clear the value at an `endsection`, an `end`, an `eof` (end of file), or never. These points represent the horizon of the current variable, the point at which its value is blanked out (in technical terms, changed to an empty string). So in the new way of thinking, context variables are simply variables whose horizon is “end” and universal variables are ones

whose horizon is “eof”. Now there are two other possibilities. You can have variables that get their value wiped at !endsection or never get their value scrubbed at all!

To manage this, the new system introduces a new metatag: !var (for variable). This metatag lets you set a combination of the name, value and horizon of a variable, separated by commas:

```
{!var name="City", horizon="endsection", value="NYC"}
```

Note that name, horizon, and value can happen in any order. Also only name is required. `{!var name="City"}` is the same as `{!var name="City", value="", horizon="end"}`, if a variable “City” has not been created (through !var, !context, !universal, etc.) earlier.

If a variable “City” has already been created then only the listed properties (value and/or horizon) are changed. In other words if a variable City is already declared `{!var name="City", horizon="eof"}` only changes the horizon, not the value.

So !var combines the ability to declare variables, set their value, and set their horizon at the same time.

A single !var metatag can declare several variables by separating them by a semicolon: `{!var name="City"; name="Gender", horizon="never"}`

If this were placed at the top of the init file it would create two variables: “City” with an initial value of “” (nothing) and a horizon of “end” (that’s the default) and a second variable “Gender” which never is wiped out and also has a blank (“”) value.

So what happens to legacy calls like:

1. `{!context gender, city}`
2. `{!name "My doc"}`
3. `{!universal type="Interview"}`

???

TAMS 3.3 just saves them in the same accounting system, treating them as if they were created with the following:

1. `{!var name="gender", horizon="end"; name="city", horizon="end"}`
2. `{!var name="FileName", value="My doc", horizon="eof"}`

3. `{!var name="type", value="Interview", horizon="eof"}`

An Example:

Someone asked me to help with the following problem. Given an interview like:

```
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{time}00:01:25{/time} {name}Sally{/name} First time, you?
{time}00:01:30{/time} {name}Bob{/name} Every day!
```

They first wanted to structure it using `!last`, with time and name as context variables:

```
{!context time, name}{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{time}00:01:25{/time} {name}Sally{/name} First time, you?
{time}00:01:30{/time} {name}Bob{/name} Every day!
```

This basically stuck a virtual `!endsection` in front of every `{time}`, except the first one (since no variable named “name” was used before then). To TAMS the data now looks like this, because of the `!last` statement :

```
{!context time, name}{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{!endsection}{time}00:01:25{/time} {name}Sally{/name} First time, you?
{!endsection}{time}00:01:30{/time} {name}Bob{/name} Every day!
```

Now the researcher wanted to add some cultural information about Bob, so he declared a new variable “culture” and used `!if` to assign it to Bob. Note this is what is in the document window, not what TAMS sees (I’ve removed the virtual `!endsections`)

```
{!context time, name, culture}{!if name="Bob"=>culture="Canadian"}
{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{time}00:01:25{/time} {name}Sally{/name} First time, you?
{time}00:01:30{/time} {name}Bob{/name} Every day!
```

The problem he had was that when he searched, Sally turned into a Canadian as well, since `!endsection` does not clear the value of `!context` variables, and no other value had been assigned to Sally as her “culture”.

One solution would be to make “time” a universal code, though universals are designed to be constant over the scope of a whole document. Another solution might be to hardcode `!end`’s before each value of `{time}` (not too bad with regular expressions). However, it uglifies the data. He could also make `!last` create virtual `!ends` rather than `!endsections`, which would accomplish the same thing without the uglification.

So what our researcher would type would be:

```
{!context time, name, culture}
{!if name="Bob"=>culture="Canadian"}
{!virtualend}{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{time}00:01:25{/time} {name}Sally{/name} First time, you?
{time}00:01:30{/time} {name}Bob{/name} Every day!
```

!virtualend tells TAMS to create implied !end statements rather than !endsections with !last. And what TAMS would see would be this if the virtual !ends were to be made explicit.

So what our researcher would type would be:

```
{!context time, name, culture}
{!if name="Bob"=>culture="Canadian"}
{!virtualend}{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{!end}{time}00:01:25{/time} {name}Sally{/name} First time, you?
{!end}{time}00:01:30{/time} {name}Bob{/name} Every day!
```

In this case, this would work because every speaker also has a time code. In my interviews this wouldn't work because I don't enter a time code for each person. I may have several turns of conversation after a given time code. In other words, I want time codes to trickle down (which they won't because !end clears context codes, which includes "time"), but I want culture codes to be wiped. So the better approach might be to let !last be !endsection, as before, but to indicate that culture's horizon is an !endsection, and time's horizon should be the end of the document (or even !end, in this case, since there are none!):

```
{!context time, name, culture}
{!var name="time", horizon="eof"; name="culture", horizon="endsection"}
{!if name="Bob"=>culture="Canadian"}{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{time}00:01:25{/time} {name}Sally{/name} First time, you?
{time}00:01:30{/time} {name}Bob{/name} Every day!
```

Now "time"s will carry forward, speaker to speaker til the end of the document, while "culture" will be wiped clean for each speaker! Note that it is redundant to have "time" and "culture" declared as context variables here, since !var will create those variables if they don't exist. So I could have just done:

```
{!context name}
```

```

{!var name="time", horizon="eof"; name="culture", horizon="endsection"}
{!if name="Bob"=>culture="Canadian"}{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{time}00:01:25{/time} {name}Sally{/name} First time, you?
{time}00:01:30{/time} {name}Bob{/name} Every day!

```

I could also have rolled name right into the !var statement:

```

{!var name="name"; name="time", horizon="eof"; name="culture", horizon="endsection"}
{!if name="Bob"=>culture="Canadian"}{!last name}
{time}00:01:20{/time} {name}Bob{/name} Come here often?
{time}00:01:25{/time} {name}Sally{/name} First time, you?
{time}00:01:30{/time} {name}Bob{/name} Every day!

```

time	name	culture	_code	_data
00:01:20	Bob	Canadian	a	{a}h
00:01:25	Sally		c	{c}Fi
00:01:30	Bob	Canadian	b	{b}E
00:01:30	Bob	Canadian	a	{a}d

Fig 4.1. Results showing Sally is not a Canadian and all is right with the data

D. Linking Documents

In qualitative research it is often helpful to refer to other documents. In creating memos, for instance you may want to go to a specific passage to see concretely what you are talking about. TA uses the bookmark feature to implement a system for referring to other places in your database. The basic scheme involves putting a bookmark at the site of interest, and then putting a !goto metatag where you want to refer to that bookmark. A bookmark in this context I will also call a “reference.”

Bookmarks have a format like the following:

```
{!bookmark My first bookmark}
```

“My first bookmark” is the title or name of the bookmark. You would put a tag (with the appropriate name) somewhere in your document that you want to refer to. Make sure your bookmark names are unique, or TAMS will refer you only to the first bookmark with that name.

In your memo, you will want to add a link to that bookmark. The link is a metatag with the following structure

```
{!goto file="myfile.rtf" bookmark="My first bookmark"}
```

The file has to have the value of the full and unique name of the file containing the bookmark, the bookmark has to have the value of the unique name of the bookmark in that file.

To use the link, simply click anywhere in the !goto metatag, and pick “Coding->References->Go to reference”. TA will open the file and move you to the bookmark.

Rather than typing in the !goto yourself, it’s easier to have TA do it for you. Go to the bookmark you are interested in and click inside of it. Then pick “Coding->References->Remember reference”. TA will put together a !goto for you and put it on a special clipboard. Then go to where you want the link, and pick “Coding->References->Insert reference link” and an appropriate !goto will be inserted at the cursor position.

All three reference functions are available as toolbar buttons. The “goto reference” is particularly useful for jumping with a click to the !bookmark location. To permanently add these functions to your tool bar you will need to use the !button metatag followed by the following, in some order, `cmd::goToReference`, `cmd::rememberReference`, or `cmd::insertReference` in your init file, or at the top of document file.

You can create references to codes in image files and PDFs as well. Each row of the coded selection table acts as its own bookmark. Simply select the row of interest and pick “Coding->References->Remember reference”. Move to a memo file, or other file where you want to refer to the image file or PDF. Then pick “Coding->References->Insert reference link”. The !goto statement will look something like `{!goto file=”mypdf.pdf” bookmark=”13”}` where 13 is the unique row number of the coded selection you are interested in referencing. Note this row number is not the same as the row number in your table. See the location column to find the unique row number. The table row number will change when you sort your data in different ways; the unique row number will not.

Chapter 5: Searching: Getting information out of documents

A. Workbench vs. Document searches

After you have coded your documents you will want to extract information from them. This generally involves looking up different codes and sifting through the results. There are two ways to do this on TA: through the workbench and through the Search tab of each document window (if I port this to X11/Linux, only workbench searches are likely to be supported, in fact, workbench searches can do everything and more that document searches can do). If you want to search across multiple documents you need to use the workbench. I will only discuss workbench searches in this document.

1. A simple walk though

To search from the workbench, **first put together your search list.**

This is done in the Files tab of the workbench. Practically, this means moving files over by selecting them from the file list (the left side of the files tab) clicking the “Add” and “Remove” buttons to move them onto the right hand, search list.

Once you have your file list assembled, click the search tab and fill in the codes you are looking for (separated by commas) into the field marked “Search” and **hit the button labeled “search”**. When asked for a file name, just click the “Ok” button. That’s actually enough to get you going. This will create a temporary file with your “results.” Your results are used to analyze your data for patterns and to quickly locate specific codes.

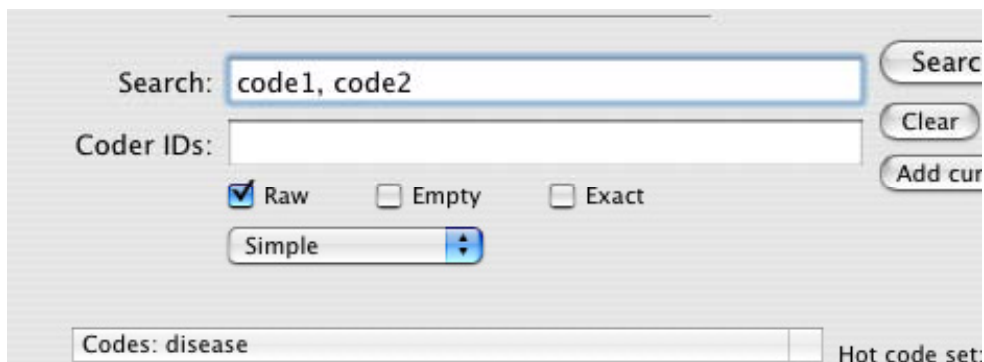


Figure 5.1. Searching for 2 codes

B. The unlimited search

This simply refers to searching without putting a code into the search field of the search tab; just leaving it blank. Hitting the “search” button will return a record for every coded passage in your document. That means the following will generate two results records:

```
{veggie} {food>parsley}I like eating things with parsley{/veggie} {/  
food>parsley}
```

One for veggie and one for food>parsley, even though the data will be almost identical. This is coded twice and it will provide two results in a “simple,” unlimited search (this assumes simple is selected as the search type).

(Note: the tags do not have to be properly nested, as this example shows, the end tags and start tags could be in either order).

C. Looking for particular codes

Of course, more often you will want to look for particular codes. From the workbench you just double click the code from the code list on the workbench and hit search.

You could also manually type them into the search field and hit the search button. By the way, searching for “food” will return the whole food family: food, food>parsley, etc. If you want to find only food, search for ‘food. That’s a single quote and the word food. This is called an exact search, and you could alternatively do it through turning on the exact flag (under the search) as well.

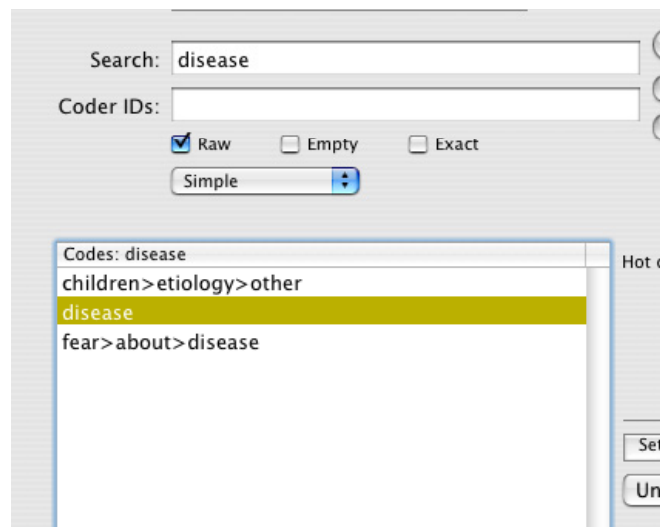


Figure 5.2. Double click from the code list to enter a code into the search

What if you want to find both food (and its family) OR things coded likes>food which is part of the “likes” family. You can search for food at any coding level by searching for “>food”. This indicates that it should look for food at all levels of the code.

What if you wanted to find food, moody, good, or neighborhood. You could search for “*ood” (asterisk followed by a pattern) this basically just searches the code name for a substring.

If you want to find text NOT coded with food, search for !food. You can combine this with > and * (put the ! first).

D. And & Or

You can combine search criteria with **AND**'s and **OR**'s. **AND**'s are indicated with the “+” sign and or's with a comma “,”. **AND**'s take higher precedence than **OR**'s, unless you group search terms with parentheses.

To search for either food>parsley or food>carrot you would enter “food>parsley, food>carrot” To find passages that are both food>carrot and loves>vegetarians you would search for “food>carrot+loves>vegetarians”

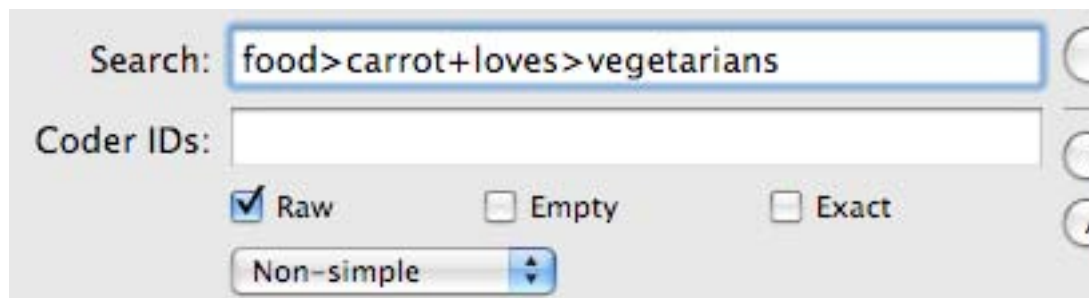


Figure 5.3. A search involving AND

Note: Use of AND usually means you want a non-simple search. There are a few cases when AND makes sense with a simple searches, usually involving the > and * modifiers. This is because each term of the AND will be applied to the single code in a simple search. So searching for two different codes will by necessity fail in a simple search. So, you will be prompted if you really mean AND when doing a simple search.

On the other hand, AND makes perfect sense in a non-simple search, since each row of your results represents a passage of text with however many codes are attached in non-simple results. In simple results each row represents a code with the text repeated in as many rows as it has codes. Again: Simple means a result row = one code; Non-simple means a result row = one passage of text.

E. Search Flags

In both the document search pane and on the workbench there are three flags or check boxes that control how TA does searches:

1. Raw searches

Raw simply means that the tags are shown in the results. TAMS will show you all the tags that are open for the start of the found passage, by the way. For actually putting things into papers you will want to turn off the raw flag when you search. By the way, returns are substituted with “\n” and tabs with “\t”. The original characters would be confusing to Excel or other databases.

2. Empty searches

Usually, when you search for data you only want to know what passages meet certain criteria. If you turn on the empty switch TA will

produce a record, data or not, at every !end (or !endsection if you turn on that feature in the preferences panel). This way you can find out how many times someone didn't mention X, Y or Z... Note this only works for structured documents.

3. Exact searches

Normally if you search for food, you will get the whole food family:

food

food>parsley

food>carrot

etc.

To look for only those things coded food, but not food>carrot, turn on the exact flag.

NOTE: you could also prefix a ' (single quote) in front of food.

F. Search types

TA can search a wide variety of ways giving very different result sets depending on need. These search types are chosen from a pull down menu under the check boxes on the workbench:

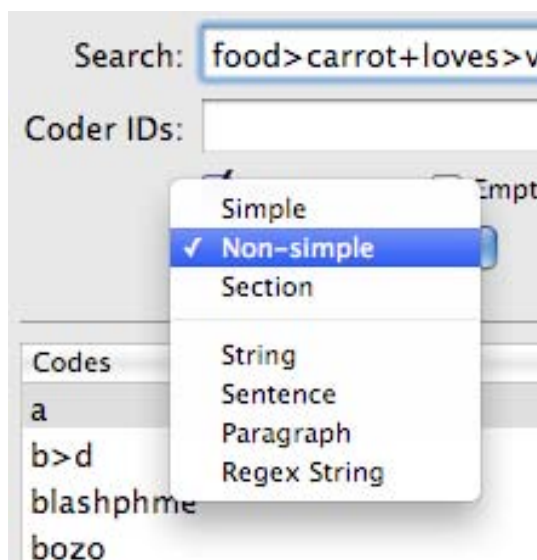


Figure 5.4. Search types

1. Code searches 1: Simple searches and non-simple searches

Code searches are the first three items on this menu. There are three types of searches that TA can do given a mix of codes. The first two, simple and non-simple are the primary ways researchers will search for data. In the first case (simple) TA searches for the tags that match the criteria that you've indicated and returns the passages of text surrounded by those codes. In the second case (non-simple), TA will check at each character whether this character is in a "zone" that meets the criteria you've set.

Consider searching for "**food, veggie**" for a document that contained this passage:

```
{veggie} {food>parsley}I like eating things with parsley{/veggie} {/food>parsley}
```

If you are doing a simple search, both `food>parsley` and `veggie` meet the criteria. So TA will generate a result for each code. You will have two rows (one for each code) in your results windows with almost identical data. On the other hand, if you select non-simple search, TA will ask if each character in "I like eating..." is in a zone that is either food or veggie and return it as meeting the criteria. There will only be one result record for this. TA indicates that this is not a simple search by prefixing the code in the results window with a "+" or "-". Generally you will want to pick simple searches.

NOTE: Searches with "**AND**" ("+") are always non-simple searches.

WARNING: Only simple search results can be recoded using your results!!! This is because passages returned from searches with "**AND**" may not be at tag boundaries.

2. Section searches

If you have used `{!end}` markers (or `{!inner}` or `{!last}`) to mark up your text so your text has sections, you can look for sections that meet certain criteria regarding codes. For example you could look for all sections that have either code "veggie" or code "animal" (your search would be "veggie, animal"). Or you could look for sections that have

both codes in them: “veggie+animal”. Notice that AND and OR mean different things in this context. If a section has both codes in it even if they do not apply to the same passage of text that section (i.e., an area ending with a `{!end}` metatag) will be shown in the results window. Similarly for OR, if either code appears in that section of text, that section will be copied into the results.

WARNING: Because `{!inner}` and `{!last}` implies that `{!endsection}` rather than `{!end}` as a break point, TA may not return the data you want. Normally TA looks only for `{!end}`'s. To make TA aware of `{!endsections}` Go to TAMS Analyzer->Preferences, and under the searching tab make sure “Report empty searches for each `{!endsection}`” is checked.

3. Character (or String), and Regex (Regular expression) searches

In addition to searching for codes, TA provides four mechanisms for searching for raw text in your documents. String, Sentence, and Paragraph searching look for the exact text you type in and return either the exact text itself, or sentences or paragraphs containing that text respectively (defaults to case insensitive—check the exact box to make it case sensitive). . In sentence and paragraph searching there is a basic mode which takes a single string (including punctuation and spaces) and tries to match them exactly in sentences or paragraphs. With the basic button unchecked you can use boolean operators (plus, comma and exclamation mark and parentheses). Terms in non-basic searches are expected to be individual words unless surrounded by quotes (either single or double but both ends must match). In string searches, the results browser will show not just the string but a little bit of context for the string (how much is determined by a slider on the top of browser option).

Regex string searches use the regular expression support provided by the PCRE library to search for text. For example with this you can search for all sentences, or occurrences at the beginning of a sentence, or words that start with b followed by three of any character and ending with x. It's powerful but really requires its own manual. Look online for tutorials on perl regular expressions (or just regular expressions).

G. Searching for coders

If you want results for simple searches only from a certain coder you could simply put in their code in the “Coder IDs” field of either the workbench or the document. You could also list coder’s separated by commas. If you want to include unsigned tags use * (no this doesn’t mean wildcard in the context of the “Coder IDs” field, it means no coder was provided). To look for unsigned and tags signed by mgw I’d fill in “*, mgw”.

You can do much more complex searches however by including coder information in the search field! If I want all cases of food coded by mgw, I could search for “food[mgw]”, or if I wanted unsigned results as well I could do “food[mgw; *]”. Notice that the coders are separated by semicolons rather than commas . If I wanted only unsigned results I’d look for “food[*]”. If there was another coder with bob as his initials I could look for “food[mgw; bob]” or if I want to know what both mgw and bob coded as food I could look for “food[mgw]+food[bob]”. This in-search coder designation can be used in both simple and non-simple searches.

ADVANCED FEATURE: You can set up pretty complex coder name systems and search for all sorts of subsets by using the ~ (tilde) in a search. Searching for [~m] will return all coder’s whose name begins with m. Or searching for “food[~m; *]” will return all passages coded as food by anyone whose name starts with m or was unsigned. Again this is not done in the “Coder IDs” field but in the Search field itself.

H. Saving Results for Excel and databases

When you have results, you may want to export them to a database. TA produces very nice tab delimited files readable pretty much straight away by Excel and other databases. Just pick Save To: and pick “Text” from the “File format” menu on the save dialogue. You can also use the Export dialogue (Under the File menu) to pick which parts of your results to export and how.

BEWARE:

If you're using a classic environment database you'll need to pick "Use old Mac new line character for results" from the preferences menu before you do a search.

BEWARE:

Panorama database stops reading things at the end of quotes. Put the following metatag in your init file: `{!noquote}`. This will turn quotes into `\Q` for double quotes and `\q` for single quotes.

Alternative: The easiest and most flexible way to get results data to other programs is through the File->Export data dialogue. Here you are given all sort of options for how to delimit fields and records. You can drag the field names around to change the order of the exported data and then select the fields you want (shift click or apple click to extend the selection). Data can be exported to the clipboard or to a file this way.

Chapter 6: From results to analysis

Once you have searched the list of files in the search list you will be looking at a “results window”, something like this:

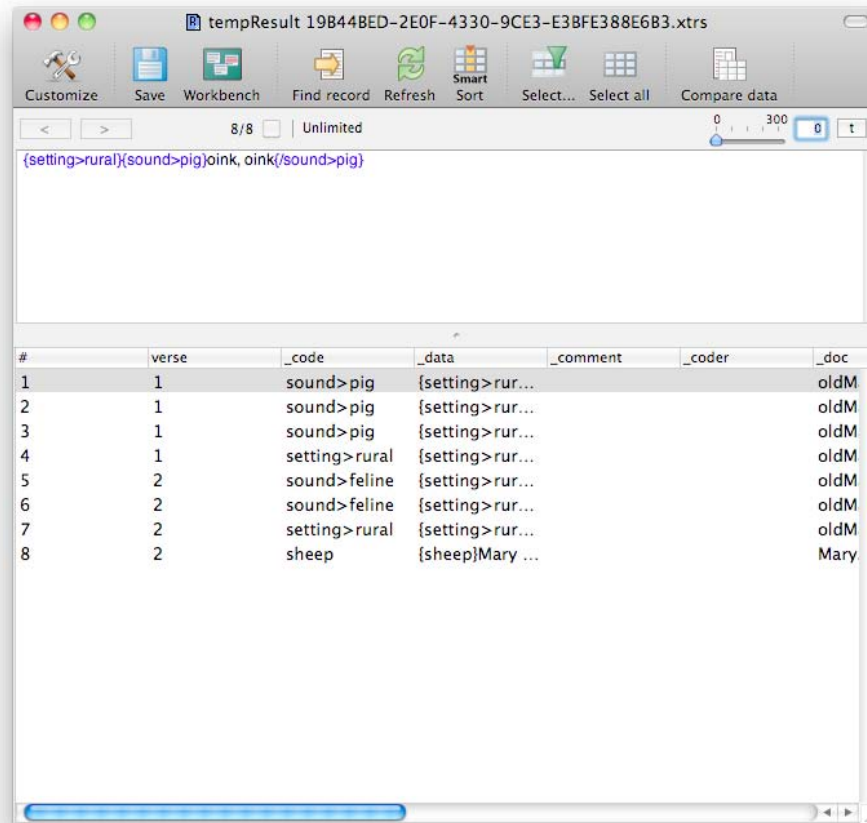


Figure 6.1. A results window

A results window is a browser for data. It allows you to move through your data, further search your data, and even modify your source documents. This section will talk about the first two of those ways of analyzing your data. First let us note the anatomy of the results window.

The lower pane. In the lower half of the results window you will see one row for each record returned by the search. This row is broken into columns: a # column which just keeps count of the records that are showing, a column for each universal code that applies to the data in that row, a column for each repeat code that applies to the data in that row, the data that met the search criteria, comments attached to that data, the coder for that passage, the name

of the document for that passage, and the beginning character offset in that document for that passage! These columns can be resized and dragged around (reordered) by dragging the column titles.

The browser pane. The upper half of the window is dominated by the browser pane that shows the value of the `_data` field/column (which is the part of the table that has the actual data in it) for whichever record is selected in the lower pane. If you want the browser to show a different column value change it using “Results->Set field to browse...” You can also adjust how much of the window should be browser and how much should be lower pane by dragging the dot on the bar between them.

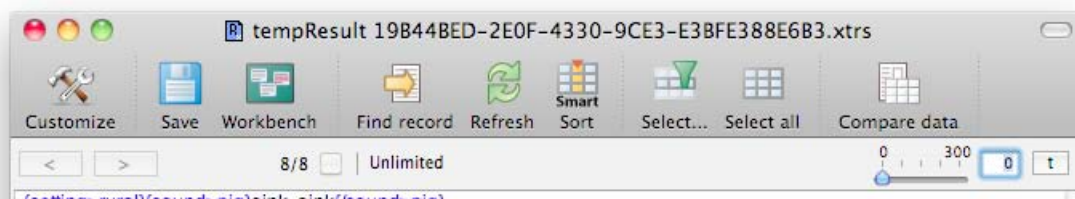


Figure 6.2. Results window button pane

Button pane. Above the browser pane is the information panel and toolbar. The two buttons labeled “<” and “>” are forward and backward buttons that allow you to flip through different “selections” of data (more on “selecting data” below). They act (and are modeled on) the forward and back buttons in browsers like Safari and Firefox. The “Workbench” (or project) button brings the workbench for this result window to the front; “sort” provides a quick and dirty ascending sort for the selected column (not the best way to sort in TA, however).

“Find record” will take you back to the document window and select the text connected with the currently selected row of the data table (all those rows at the bottom of the window). This is the best way to go back and see the context of that particular row’s data. However, you can also see a little bit of the text surrounding your data (i.e., its context) by moving the little slider on the right side of the information panel. In the above example, up to 100 characters before and after (200 total) can be added to the above display by moving the slider to the right. The exact number of characters will appear in the text box next to the slider (between the slider and the button marked “T”). You can also enter a number directly into that box. Hitting return will then show exactly that

number of characters before and after (as well as move the slider to the position appropriate for that number). If you want to set the maximum value for the slider just enter the maximum you would like as a negative number, this will tell TA that you are trying to reset the maximum on the slider. In any case, if you enter a number larger than the maximum, TA will adjust the slider so that it can handle the larger amount. So the negative number trick is mostly useful if you want to decrease the number of characters the slider can handle. Final note on the “padding” control slider: the default value of the slider is set in the program preferences on the search tab. Change the value for the “Number of characters added to string searches” to adjust the default maximum for the slider.

The “Refresh” button re-runs the search (in case the source document has changed). You’ll be tipped off that such changes may have occurred by a check mark next to the count of how many records.

The record count is displayed as a fraction with the numerator representing how many records are visible, and the denominator providing a total count of records in the search. As you select records from the search a string next to the checkbox will change to give you some indication of what characterizes records you are currently browsing (In Figure 6.2, we see the word Unlimited meaning that this is an unlimited search).

A. Selecting data

Often clicking the search button is only the first step in exploring the meaning of data. Having pulled up the records related to veggies, we now want to find out, what did “Bob” say about veggies? What (or how many) passages about veggies included the word carrot? Did any of those also include rutabaga? What records under veggie have neither rutabaga or carrot?

To answer these questions we need to do complex *searches* of our result window. The result window is really a fairly complete “flat file” database, i.e., a database program that works on one file at a time. Searches have to be done one column at a time BUT through doing multiple searches you can create complex AND and OR relationships. Furthermore, once you find a set of records that answer your question you can “name” this set of records so you can pull them up instantly. You can also create “autosets”: macros or little programs that can pull up similar records from any result

file!

I used the word search in the last sentence, but from here forward I will use the word select to mean “search in a result file window” and reserve the word “search” for workbench or file searches of your source documents.

Finding a value in a context, repeat or universal column. A context (as well as repeat or universal codes) code appears in the results window as a column. Typically such columns will have different values for different rows: in a gender column we’d expect to see some with “M” and some with “F” (forgive the dichotomous treatment of gender). In a speaker column you would expect to find different names: Bob, Sue, etc. If I wanted to see only the records (rows) that had “Bob” as a speaker (assuming I have coded Bob as a {speaker} in my source documents, and “speaker” is designated as a {!context} code) I would first, click the title of the “speaker” column so that the whole column was selected (highlighted). I would then I then pick Results->Select... from the Results menu (or just click the “Select...” button) and I would see this dialogue box:

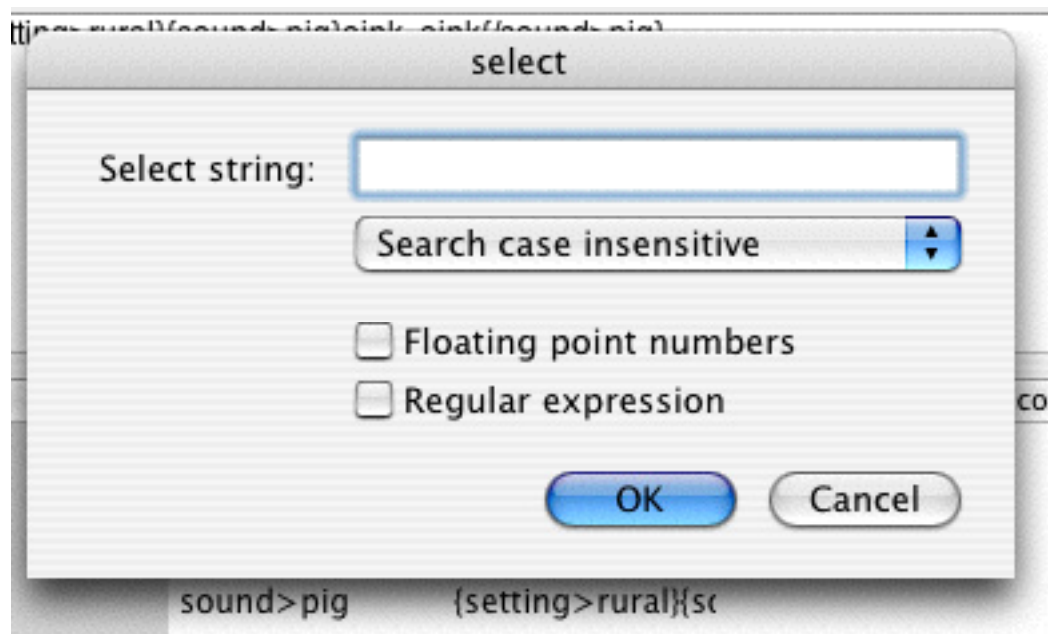


Figure 6.3. The select dialogue box

Enter Bob into the Select string field and press “OK.” Now only records (rows in the data table) with Bob or bob or boB (etc.) in the “speaker” column will be visible.

Now, fewer rows will be present in the data table, and the record count will have changed to indicate that in the information panel, as well. We can

see in our fictional example here that (for instance) 3 out of the total of 6 records in our search had speaker “Bob”:

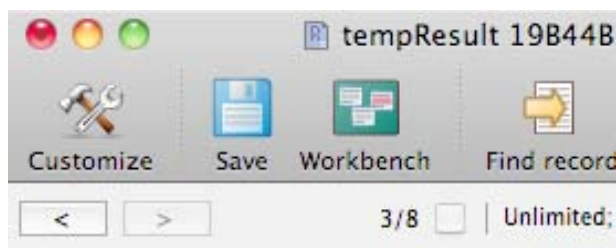


Figure 6.4. Changed record count

Seeing all of the records. Given that only one “select” ago we were looking at all the records, we could use our browser forward and back buttons to leap back one selection and see all of our records.

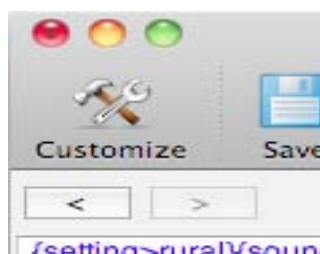


Figure 6.5. The browse back button

Alternatively we could leap to all records by picking “Results->Select all.” We could also click the “Select all” button on the tool bar.

Selecting values in your data. What if we want to find things that were coded veggie and have in their coded text the word “carrot?” This is done through two selects. In this case, the order of the selects does not matter, though there may be cases when it does (I’ll discuss one shortly). The first thing to do here is to select all those records coded as “veggie”. “veggie” is a code, and these are put in the column called “_code” (all of the columns that TA creates start with an underscore). Pick this column by clicking it’s title and making sure that the whole column is highlighted. Then select “veggie”: pick the “Results->Select...” menu option and type in “veggie” and click ok. Now only the veggie items are selected.

We now want to further whittle down the data to be only the ones where people used the word “carrot.” To do this we simply do another “Results->Select...” after clicking on the _data column, where the coded text resides. Each time we select data we are working only with the records showing, not with the entire count of records returned from the work bench

search. So in this case we will pick the “_data” column by clicking on the title (“_data”) and picking “Results->Select...” a second time and press OK. Now only records that met criteria in both selections will be visible: we have selected “veggie” in the _code column and “carrot” in the _data column.

Logical ORs in selecting records. What if we wanted records that were either coded “veggie” or “fruit”? If we select veggie the first time the way we’ve already described, we will have all the records coded “veggie”. To add to this the records coded fruit we select the _code column and pick “Results->Select additional...” This will give us the same dialogue box to type in “fruit” but instead of selecting from the shown records, TA goes back to the whole pool of records and adds the ones meeting this new condition to those shown.

Naming a selection. TA is very big on “sets”, meaning subsets of lists of things. It supports working with “code sets” (subsets of all your codes, so you don’t have to scroll through however many hundreds of codes you’ve created), file sets (so you can keep your memos separate from your interviews), and results sets (aka data sets) which name a portion of the records returned from a search.

In the case of having found all of those records that are coded “veggie” and spoken by “bob” and use the word “carrot”, it might be nice to “bookmark” these records so you can just leap back to them whenever you need.

With just those records showing (i.e., you’ve just done the selection of speaker=bob, _code=veggie and _data=carrot (by equal we means contains, in reality)) pick “Results->Result sets->Create named set”. You should be prompted to provide a name, type in something like “bob-veggie-carrot”, i.e., some meaningful identifier for this selection of data. Click “OK” and you have now created a named set. **A new item has been added to the “Results->Result sets” menu which is your named set!**

To show these records at any time you can just pick “Results->Result sets->bob-veggie-carrot”.

WARNING:

Named sets are saved with result files, **but are disposed of the minute you refresh your data.** As a result, if you need something persistent use auto sets!!! Named sets are best for on the fly data analysis since they are easier to use than autosets, but autosets have the advantage of permanence and

can be used across results (searches).

ADVANCED:

Named sets and autosets are not just bookmarks for your data (though they are primarily that). You can do complex set operations with data sets. Using the “Results->Result sets->Set operations...” menu option you can take the selection that is showing in your results window and do things like intersect it with a named set or find the union of it and a named set. These of course can then be named as well. This allows for very complex analysis and mining of data.

Autosets. In named sets you are just giving a name to a group of records. With autosets you actually teach TAMS how to find records meeting the criteria you used. The advantages of this are many: autosets will persist after you refresh and autosets can be applied to other searches. To create an autoset, choose the “Results->Result sets->Create autoset...” option. That will show the following dialogue:

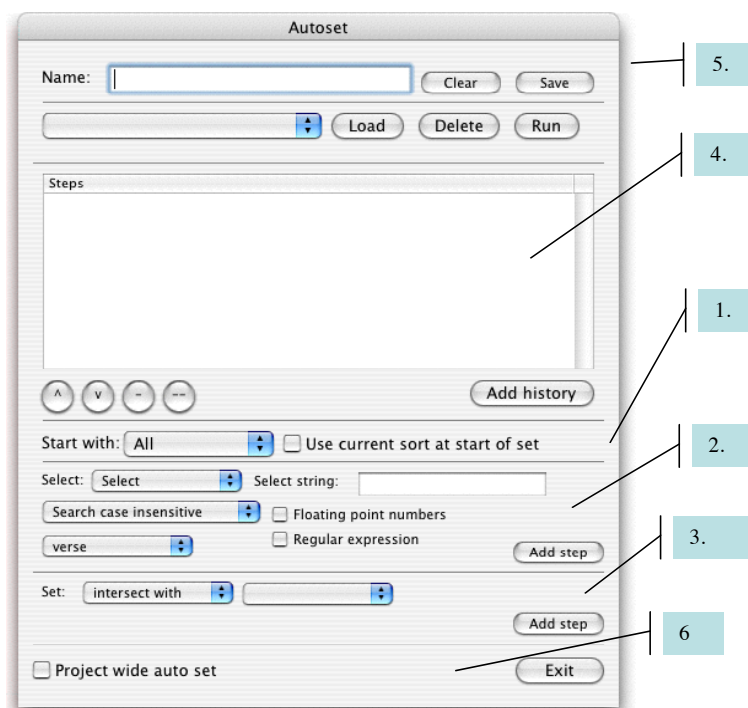


Figure 6.6. Autoset dialogue

With the autoset dialogue you teach TA the steps needed to make a results set. **If you have done a selection, and want the steps involved in that**

selection to be your autoset, press “Add history” (the sequence of steps you have just done) in area #3. You may need to modify what appears in area #4 using areas #1 #2, and # 3, and removing wrong steps using the “-” button in area #4.

To create an autoset from scratch, you start in area #1 (see Figure 6.6) by indicating whether the steps you indicate should start with all of the result records (no matter what is currently showing) or with a current selection. If you have used the Results->Sort up and Results->Sort down to organize your data TA can remember this sort order and automatically sort your data before applying the next actions to it.

To the base established in step 1, you can add two types of steps, first you can apply selections through the menus and fields in area #2. Alternatively you can do set operations using other autosets through area #3. In either case once you’ve filled in the information through the menus, fields, and check boxes, click add step to have this operation registered in the autoset. **If you don’t click “Add step” the act will not be registered.**

Area #4 is where these steps show up. In area #4 you can rearrange the order of steps (using the ^ and v arrows), you can delete one step (click on it and press the “-” button) or all steps (press the “- -” button).

Once the steps are in place, decide if you want this auto set available project wide. If yes make sure the box in area #6 is checked. In all cases you need to enter a name for this autoset in the box in area 5 and press save.

Under area #5 is a menu which allows you to recall and delete older autosets.

When done press the Exit button in area #6. To actually run your autoset pick the name you entered off of the “Results->Result sets” menu.

Sorting. There are two ways to sort data in TA. The first is through the sort button right in the button panel on the results window. This sorts up (A before Z) and is good for quickly arranging a single column. The preferred method of sorting however is through the Results->Sort up and Results->Sort down menus. These allow you to nest sorts so you can sort columns inside of columns. You can also control the type of data that is being sorted, i.e., you can specify whether the column contains dates (but set the date format first through Results->Date format), integer, string, real

number (floating point) or a code. Sorting codes require you to set a code level through the Results->code level menu item. If the code level is 0, then the code is simply treated as a string and is sorted alphabetically. If code level is 1 then a>b and a>c are considered the same! Only the first level is examined. If code level is 2 then a>b>c is the same a>b>d but, both are different from a>c or just “a”.

When you use the menus first you sort using one of the criteria that does not have the word “within” included. This tells TA to forget all previous sorts and start again. The menu items that include the word “within” add onto whatever sorts have been done. So after the first time you sort, use the within’s to sort additional s.

What’s important is that TA remembers the sorts done by the menu and can import the criteria for sorting into data summary reports and autosets. If you use the sort button none of those criteria are available to the TA program.

B. Temporary Columns

Sometimes the columns generated by context variables and TA (TA supplies things like `_code`, `_data`, `_begin_loc`, etc.) are not enough. You’ll want to add your own columns with either unique data or data transformed from other columns. TA provides a function called temporary columns that are columns you can add and modify directly.

Temporary columns don’t affect your source files (directly), but they allow you to group and select data and create analysis tables using emergent categories. The data in these columns can be added and edited through setting values directly (double clicking on a cell), setting values for the selected or marked records, and by cascading values so that blank cells take their contents from the ones above them.

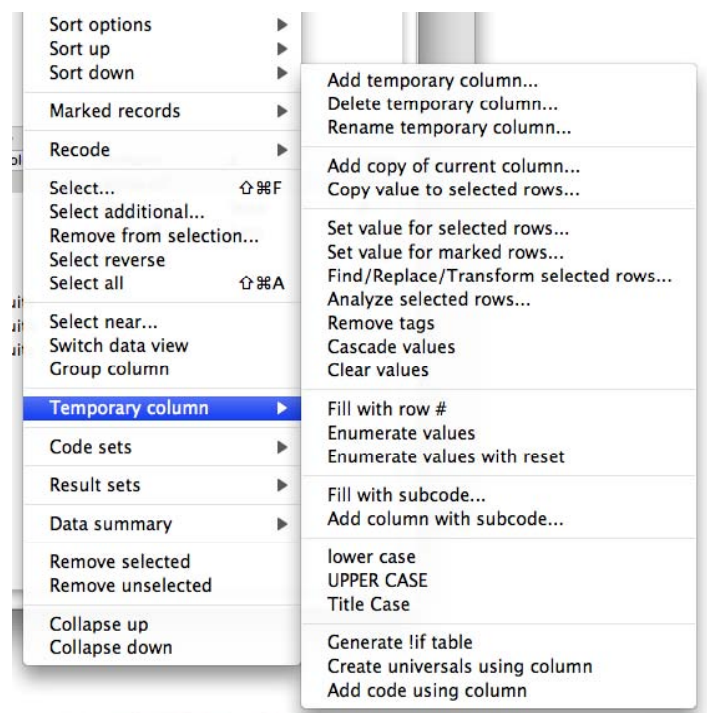


Figure 6.7. The “Temporary column” submenu of the Results menu

While temporary columns and their contents are saved when the result file is saved, they are destroyed if you “refresh” or do anything that refreshes your results windows (check your preferences!).

For most of the functions listed in the submenu, you must first select a column by clicking its header.

When you add a temporary column you fill in this dialogue:

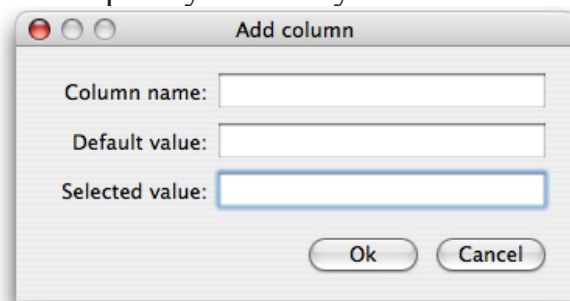


Figure 6.8. The add column dialogue

The column name is the only required field. The program fills in the default value for every record. Then it goes through the selected records (the visible ones) and fills in the value in the “Selected value” field. It does this even if both are blank.

Even more often than adding a column, I find, is copying one. Click on a column you want to copy, then pick “Results->Temporary column->Add copy of current column.” You’ll be prompted for a name for the new column.

You can also very easily append the contents of a column to a temporary column. Select the temporary column you want to append to; and pick “Results->Temporary->Copy values to selected rows.” In the dialogue that is shown, pick the column you want to append to the currently chosen column. The old data and the new data will be separated by a space unless you click the append box and fill in what you want between the existing data in the column and the data you are adding:

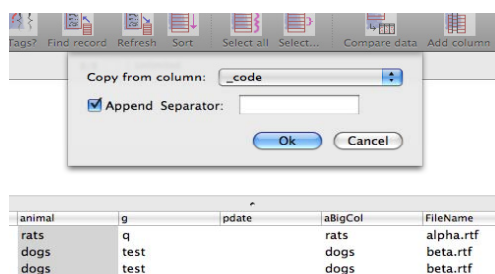


Figure 6.9. Appending `_code` to column animal

This operation only works for temporary columns.

There are a variety of ways that you can add content to or transform temporary columns. You can change the case, you can find and replace in each cell, you can fill the cells with numbers (enumeration). In all these cases the temporary column must already exist, you then select the column (click its title) and pick the function from the temporary column submenu providing any information TA asks for.

1. Analysis of temporary columns

Selecting a temporary column and picking “Results->Temporary column->Analyze selected rows...” provides several numerical analyses of the data in the selected temporary column.

Make sure you are working on a copy of the column as the contents will be replaced by the results of the analyses described below. To make a copy of the column of interest pick “Results->Temporary column->Add copy of the current column...” command, which asks for a new column

name and then generates a new temporary column with a copy of the currently selected column's data.

Three analyses are available to you

A. Length:

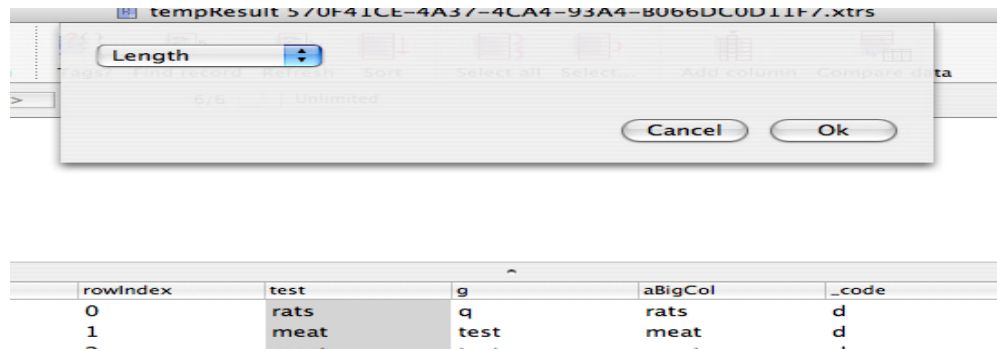


Figure 6.10. Replace string by its length

This simply replaces the current column with a measure of the length of the text in each row in the currently selected column.

B. Count of expression

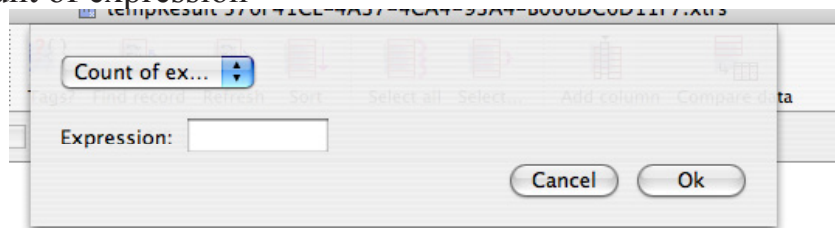


Figure 6.11. Replace string with count of matching expressions

Using regular expressions (filled into the Expression box), this choice will replace the current column with the number of matches of that expression in each row in the current column. This is useful for getting a word count (Expression = “[a-zA-Z]+”) for instance.

C. Average length of expression

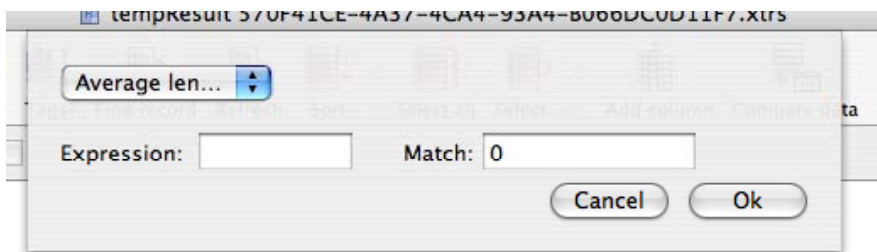


Figure 6.12. Analysis of the average length of expressions

Here the program replaces the column contents with a calculation of the average length of patterns. Users can indicate if they wish to examine the whole expression found (Match=0) or a substring of that expression (Match > 0). Refer to a regular expression tutorial to learn about substrings. To find the average word length we would fill in the “[A-Za-z]+” as the “Expression” and keep Match at 0.

2. Enumerating data

TA can replace the contents of a temporary column an enumeration of its values. Consider this example:

Original	No reset	Reset
x	1	1
x	2	2
y	1	1
y	2	2
y	3	3
x	3	1

Figure 6.13. Reset vs. No reset

Originally, this consisted of three identical columns. Notice that there is an extra “X” in the last row, that’s orphaned from the two X’s at the top. TAMS offers two ways to count this data. The first way keeps counting when it finds matching values, whether those values are contiguous with others or not. That’s the “No reset” column above. The final X is numbered “3” since it is the 3rd X that TAMS finds in the column.

The second way that TAMS enumerates restarts the count every time the value in the column shifts. In this case, the final X is counted as “1” since the immediately previous value is a “Y”. Contiguous values are enumerated starting with 1, each time a patch of contiguous values are encountered by TAMS as it works its way down the list.

To use these option select the temporary column with the data you wish to enumerate and pick Results->Temporary column->Enumerate values (or Enumerate values with reset).

3. *Temporary columns with subcodes*

Temporary columns can be used to generate a column of “subcodes.” Subcodes simply means a part of a code. Given code a>b>c subcodes might include a>b or b>c or a or b or c. While this could be done with tricky regex coding, TA makes it easier. Pick “Results->Temporary column->Add column with subcode...” That will give you this dialogue:

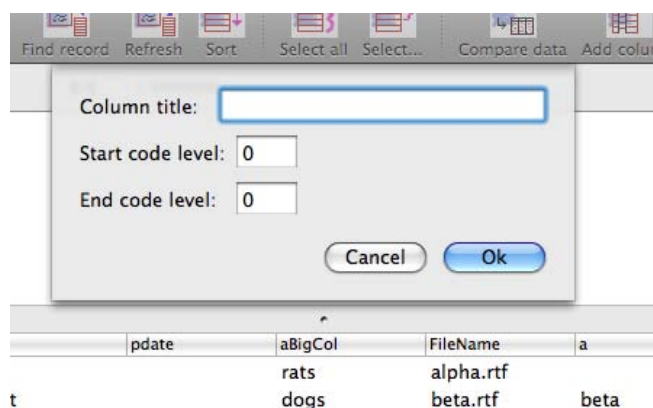


Figure 6.14. Subcode dialogue

Fill in a unique name for the column. Then put in the start and end levels you want: for a>b>c putting in a start tag level of 1 and end tag level of 2 would give you a>b. Putting in 1 and 1 respectively would give you a. Putting in 3 and 9999 (or some other large number) will give you c for this case but for a>b>c>d would give you c>d.

You can also use negative numbers to create subcodes based on the lowest levels in the end of codes. so putting in -1 -1 would give you the

lowest level of code for each row (i.e., for $a > b > c$ it would give you “c”). Filling -1 and -2 would give the last 2 for each row ($b > c$).

4. Using temporary columns to transform your source documents

Advanced topic



DANGER

The temporary columns can also be used to code at both the !universal and !context levels your source documents. Playing with these features can potentially scramble your data files, are likely to have unintended consequences, and if used more than once—will cancel earlier uses of these features in unintended ways. But if you want to play with fire here are some implementation notes.

You can set a repeat value in your source documents by picking “Add code using column.” This will insert a !setcontext metatag at the top of the source documents for the current selection. It will insert “{!setcontext X=“Y”}”, where X is the column title, and Y is the value in X for that row. If you have 3 or 4 rows of a result window with nested codes marking the same data, you can imagine the sort of mess this can leave in a file.

Somewhat, but only somewhat, less dangerous is the “Create universals using column” feature. This will first check that only one value in the selected rows applies to any file and will inform you if more than one applies (without doing anything). If it finds that one value maps to each file (though the same value can apply to many files), then it will go through each file represented in the selection and insert at each one’s start a {!universal X=“Y”} metatag where X is the column head and Y is the value for that file.

Both of these features also alter the init file of the project. “Add code using column” inserts a {!repeat X} where X is the column title at the start of the init file. “Create universals using column” inserts a {!universal X=“”} at

the start.

Use at your own risk!!! REMEMBER YOU CANNOT UNDO THIS!

C. Select Near (**Advanced topic warning**)

Select near provides a hierarchical view of the data that groups records related to other records in an outline structure. Select near allows users to answer questions such as “What occurrences of code X occur within 5 minutes of each occurrence of code Y?” “What occurrences of code set X are within 4 sections of the current selection?”

In both of these model questions there are two sets: a base set which is defined by the current record selection, and a comparison set. In the first question Y is the base set and X is the comparison set. A set image of this relationship is suggested by the following:

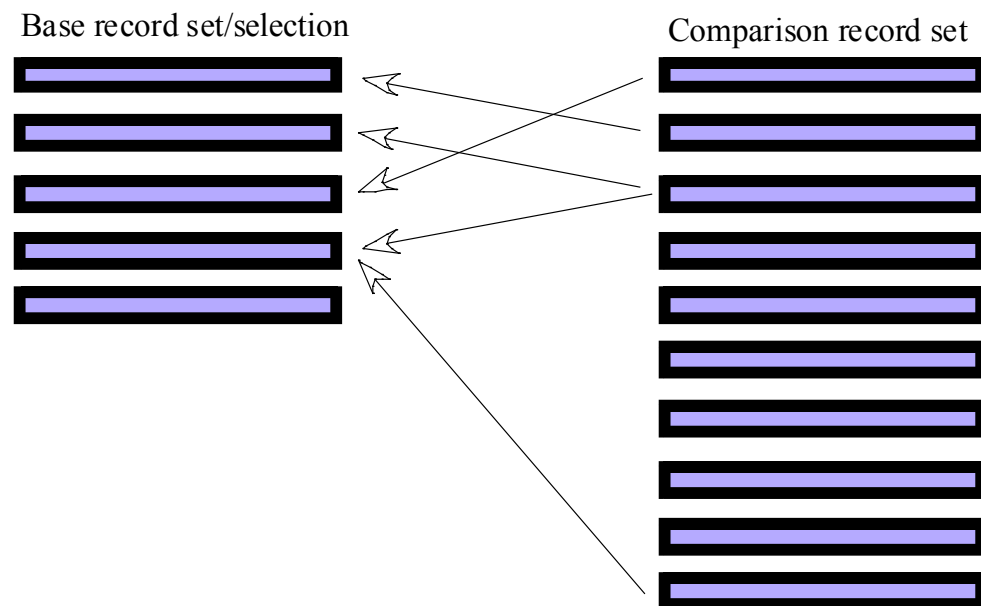


Figure 6.15. Select near matches many records to the current selection

It is clear here that each base can have many comparison records assigned to it, and that each comparison record may be assigned to multiple base records.

To understand this complex new function I want to work through the 1st

example above. Here it is diagrammed with its different parts:

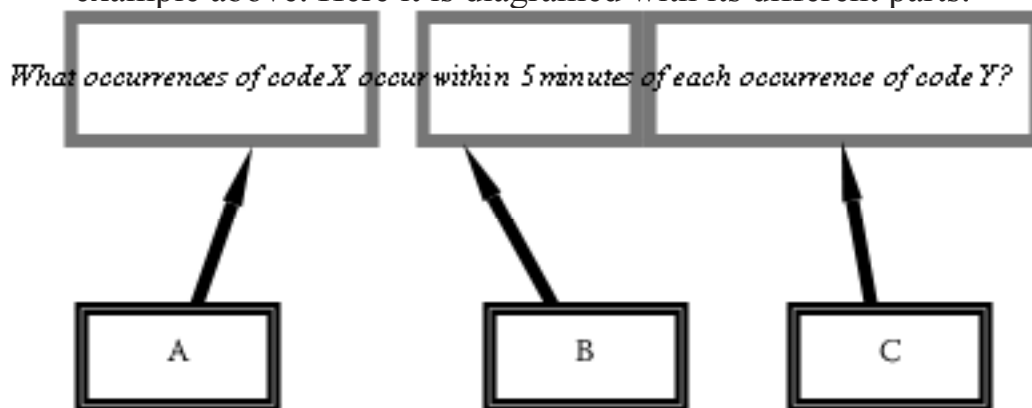


Figure 6.16. Sample Query

1. To turn this into a TAMS operation, first select the records that you want to take as the base of the *Select near* operation. This is determined by the part of the question I've marked as "C," the "each occurrence of code Y." To do this select the `_code` column and do a simple selection for code Y.

Note: select near will work with any selection; it doesn't have to be the `_code` column. It can be as complex a selection of records as you like.

2. Pick "Select near" from the results menu. It will provide you with this sheet

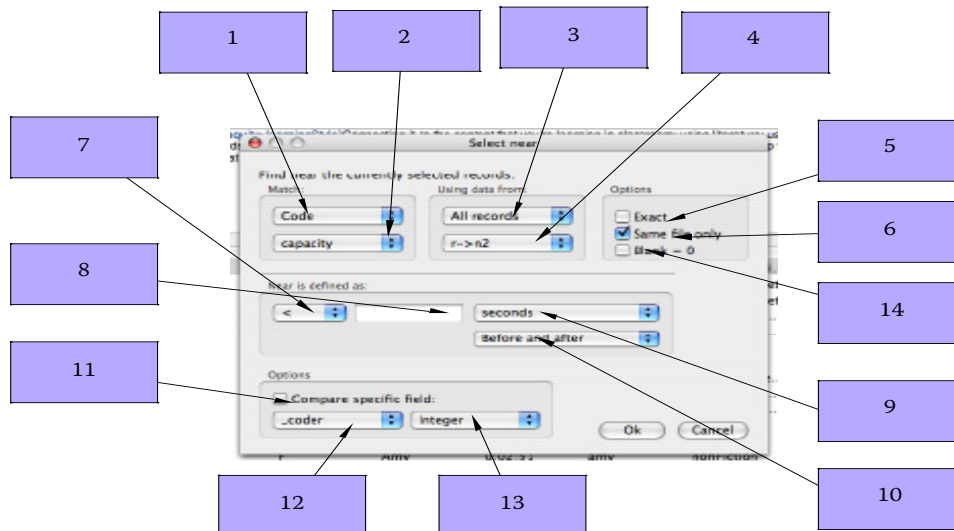


Figure 6.17. Index to the parts of *Select near* dialogue

- A. Since we are looking for code X pick “code” in the “Match” box menu 1 and then pick X from the lower menu (Menu 2). This matches the A part of the diagrammed query above..
- B. Presumably we want the comparison set to be with all of the records in the result file. If so, pick “All records” from menu 3. Alternatively you can use the current selection or a particular results set. In the latter case pick the results set from menu 4.
- C. If we want only to match X (and not its children) select check box 5; if we want anything in the X family leave it unchecked.
- D. If we want the matching records only to come from the same source/ data file as the base record select box 6. If box 6 is unchecked comparison records from any file will be matched to the base set.
- E. If we want blank records to count as a numerical 0, check box 14. Otherwise leave unchecked and empty records will be thrown out of the analysis.
- F. According to our Query we want comparison records that are less than 5 minutes from their matching base records (see part B of the query). So for G we’ll pick “<” (within is similar to less than) from menu 7. Fill in “5” in field 8. Pick minutes from menu 9. If we want those 5 minutes to be before or after the time in the base record pick “Before and after” from menu 10, otherwise select before (the base record time) or after as is appropriate.

Note that TA will pick the field to examine based on your choice in menu 9. For time units it will use the time field indicated in your program preferences, for characters it will use `_begin_loc`, for line numbers it will use `_line_number` (if you are scanning for line numbers and have used them in your source files. To over ride these decisions use items 11, 12, and 13. Check box 11, pick the field to search from menu 12, and indicate the type of data included in it from 13. You will still need to pick a comparison operator from menu 7 and fill in a value into field 8.

3. At this point pick OK and the results will appear in an outline format:

	FileName	time	speaker	_code
1	Amy	0:01:28	amy	narrative
▼2	Xander	0:01:53	xander	narrative
1	Xander	0:01:53	xander	capacity
2	Xander	0:03:30	xander	capacity
3	Xander	0:03:30	xander	capacity>

Figure 6.18. Results of Select Near

4. At this point you will want to get back to the “table view” of your results. TA3 will not allow you to do any thing other than view results and export them from the “outline view.” There are three different ways to toggle back and forth between table and outline views..

- A. Use the *Results->Switch data view* menu item
- B. Use the quick toggle button on the info bar. It will switch from O (outline) and T (table) to indicate which view you are currently in:

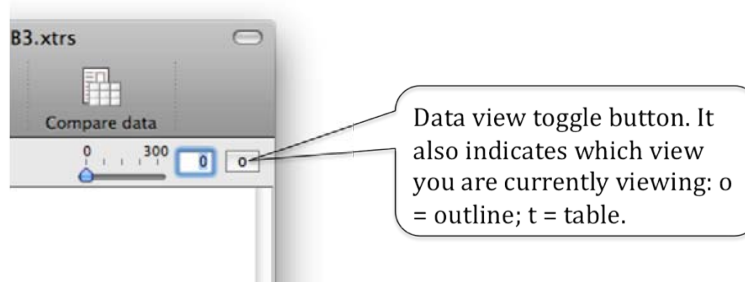


Figure 6.19. Data view toggle button

- C. You can add an optional button to your toolbar which will toggle the two views: Use “Windows->Customize tool bar”.



Figure 6.20. The Toggle Views Icon.

D. Exporting “Select near” records

Select near results are not saved when the result file is saved. The only way to save these results is to export them with the File->Export data menu item.

First make sure that your results window is showing the results of your select near search. In other words, the result window must be in outline view. Pick File->Export data and you will get an export data pane with some additional fields and altered text from the one for the table view:

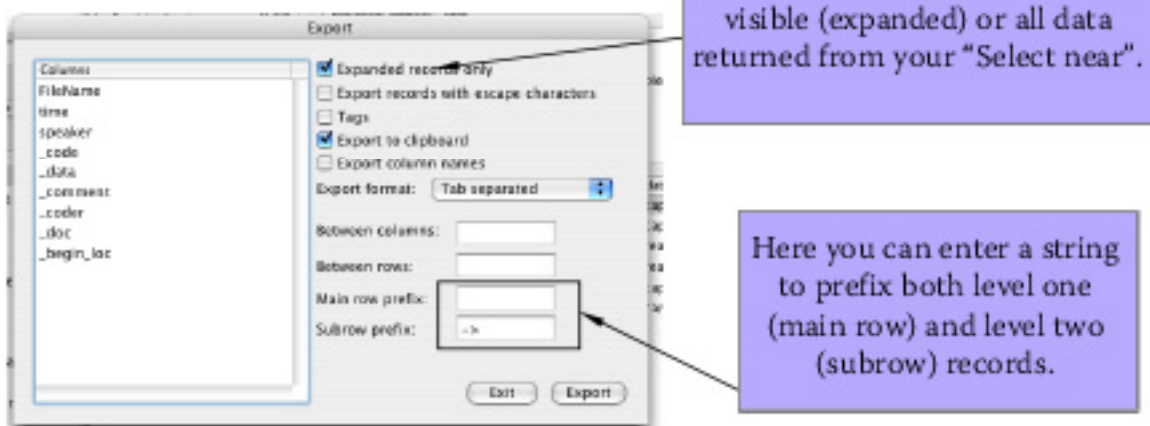


Figure 6.21. Export data for Near Results

E. Working with PDF and Image Files in Results.

PDF and Image files return result rows just like text files. Simple searches work as expected. Unlimited and section searches, however, just do what

simple searches do: they return one line per row on the coded selection table. What is included in that line depends on the options set in the PDF/Image file (see figure 6.22)

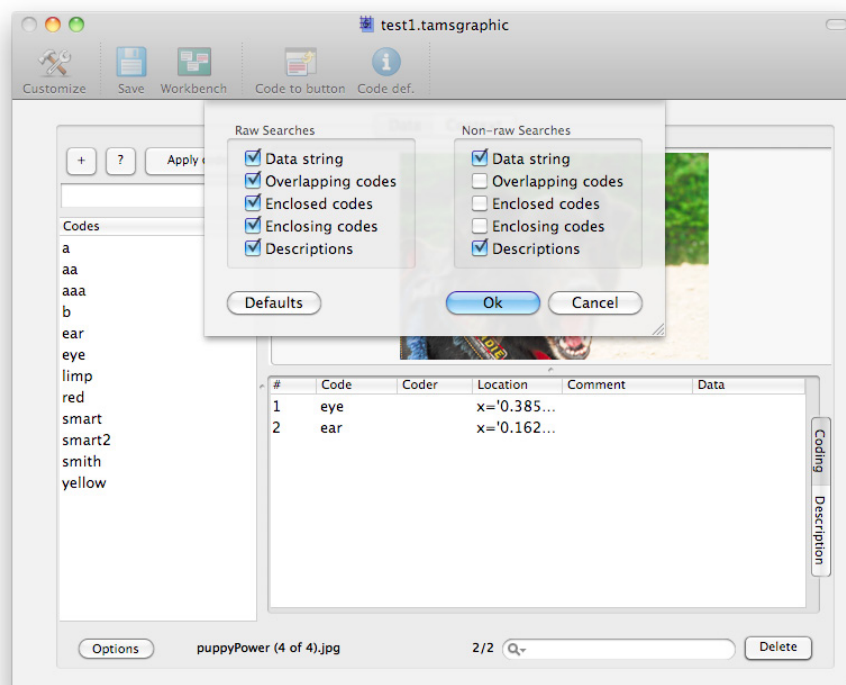


Figure 6.22. Search options for an image file. Note **options** button in lower left corner of the window

These options include not just the data you have filled in under the Data column, but the intersecting and enclosing codes—which can help in some types of analysis. Furthermore, you can set different options for raw searches vs. non-raw searches. The results of such a search are shown in figure 6.23.

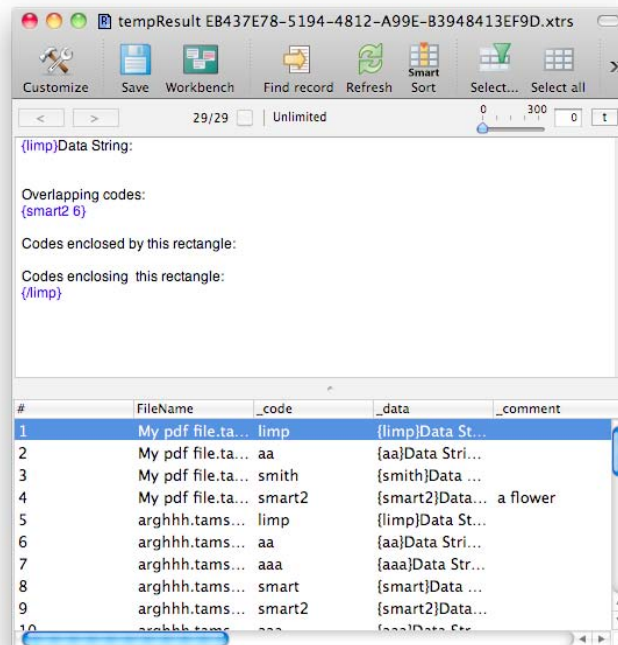


Figure 6.23. Results of an image file

In figure 6.23 you can see that, in addition to the data string (here blank), the descriptions are included as a list, and that one code is enclosed within the rectangle of the selected one (record 1, which is coded as “eager”). The enclosed code is “noodle” in this example. It is followed by an integer (#9, in this case). This is a reference to the unique row number of that code in the coded selection table in this row’s source window. Using that and the location field you can easily find that record. (You can also find that record by double clicking the row of this results window, and when the source window opens and selects this row, clicking the ? mark button to reveal all intersecting records).

This sort of textual information provides important information and allows you to do selections for overlapping codes. However, you probably want to view your image (or pdf) as you scroll through the results. Pick Results->Play media... to open the media player which will reveal each coded selection in a separate window (see figure 6.24).

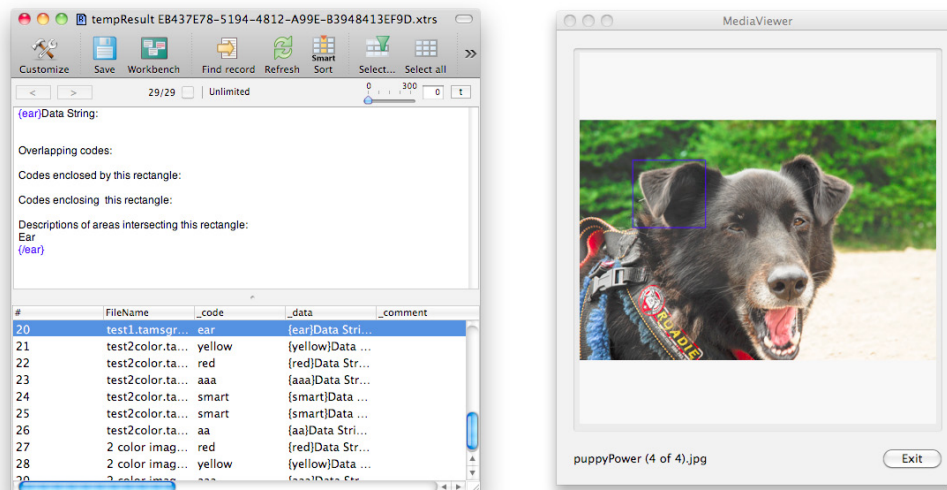


Figure 6.24. Results with media player (viewer) showing an image file and selection

While difficult to see in the screen shot image, the dog's eye is highlighted with a blue rectangle, indicating the coded portion of that image. As you scroll through the records the relevant part of the image (or pdf, or video) will be highlighted.

You can peek at the location information which includes the PDF page number, unique (to its file) record number, the selection type (image or text) and other information by creating a temporary data column in your results window and then copying to it the field called “._location_string”.

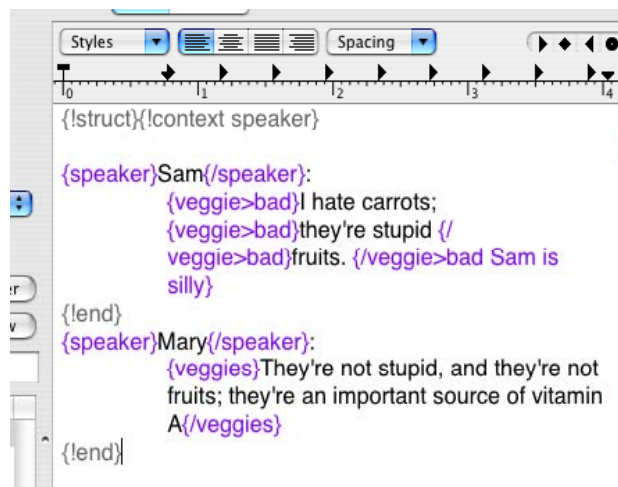
Interactively reworking your coding (Recoding)

Based on what you find in your searches you will want to go back and “recode” your document; which usually means adding another layer of subtlety to your codes. First time through you may have just wanted to catch any mention of veggies. So you coded anything that seemed slightly relevant “veggies”. Then you want to see what people are saying about vegetables, so after searching for veggies you’ll want to change those codes to `veggies>good`, `veggies>bad`, `veggies>whatever`. I refer to this process as *reanalysis*, and it involves recoding your data (which fits the example I just gave) and adding codes to your data.

A. Finding the results in the text

The first way you could do revise your initial codes is “manually”. To go back to the original place in the text from a result window, click on the record (row) you want to look at in the original context, and then just click the “Find record” button and the coded text in the original document will pop up!

Consider this section of a mock interview



```

{!struct}{!context speaker}

{speaker}Sam{/speaker}:
  {veggie>bad}I hate carrots;
  {veggie>bad}they're stupid {/
veggie>bad}fruits. {/veggie>bad Sam is
  silly}
{!end}
{speaker}Mary{/speaker}:
  {veggies}They're not stupid, and they're not
  fruits; they're an important source of vitamin
  A{/veggies}
{!end}
  
```

Figure 6.22. A mock interview

If you do an unlimited search you should get a results window like the following:

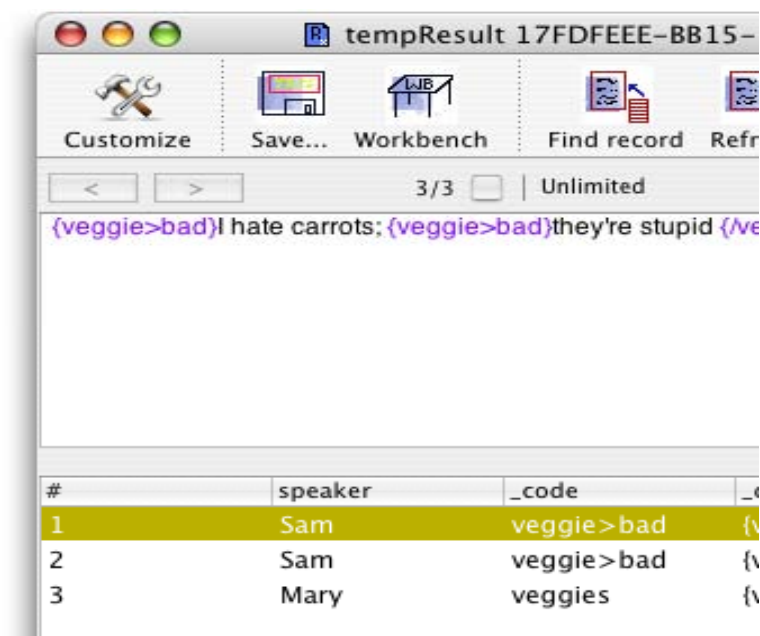


Figure 6.23. Unlimited results from the mock interview

Here, the first row is selected (you can see the text for that record in the browser above). Now click on “Find record” and you’ll be taken back to that first record, with it selected:

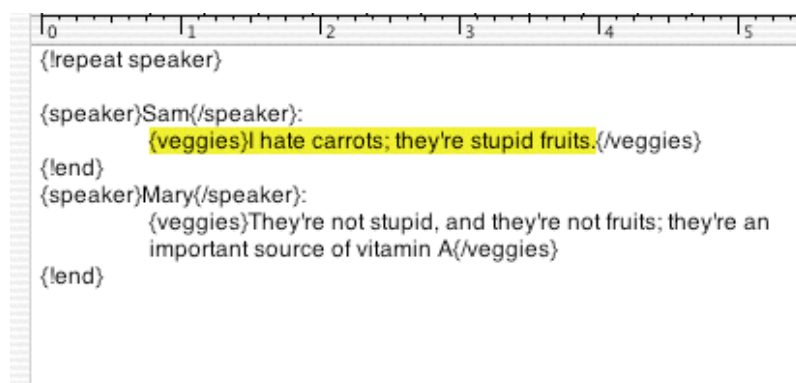


Figure 6.24. Find record takes us back to the original text

NOTE: This is an important tool for examining context!!! This takes you back to your source document and scrolls to the original text.

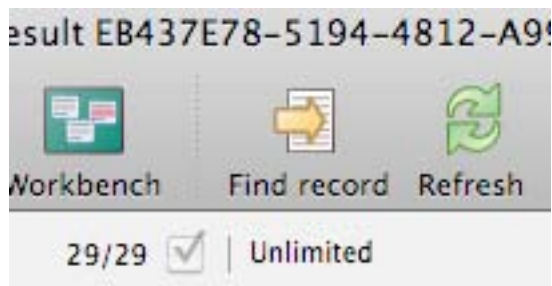


Figure 6.25. The result refresh button and a check indicating a changed document file

You can use the “Refresh” to re-synch your results and your original documents.

B. Marking results

An alternative to manual recoding is to have the program go through and change or add codes to for you. This involves (1) “marking” the records you want to change and then (2) picking “Results->Recode->Add code” or “Results->Recode->Recode”. **You must mark the records first!!!** TA works very hard to keep the source document and result document “in step” at least as far as the location of the listed result passages are. If you manually type in the source window all bets are off. TA will lose track of where the coded passages are. The best thing to do is to lock out those functions that will put TA out of synch between source/document windows and results windows. **To do that pick “Project->Reanalysis mode.”** Pick it again to turn it off.

To mark records (rows) for adding codes or recoding, select a row and pick “Results->Marked records->Mark.” That will add a “+” sign after the record number (your signal that this record is marked). You can unmark records by picking “Results->Recode->Unmark”.

#	speaker
1+	Sam
2	Sam
3	Mary

Figure 6.26. A marked record (notice the + sign by the #1)

NOTE:

In reanalysis mode the only things you are allowed to do in your document windows are double-click code, code using the code button (which you can set up to prompt for comments), and delete code pairs. These steps will keep open results windows in synch with the document windows.

C. Adding codes

Adding codes simply surrounds passage associated with the marked records with an additional code; the original code is not affected. After marking records (See §VI.B) you want to add a code to, pick “Results->Recode->Add codes”. You will get a dialog like this:

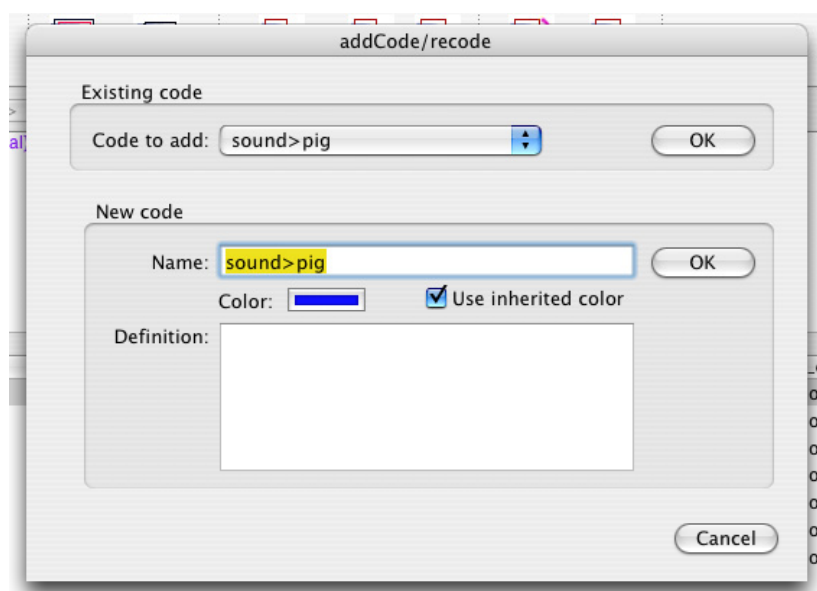


Figure 6.27. The Add code dialog box

You have two ways to go here: you can pick a code that exists from the menu and press the top OK button, or you can type in a new code into the Name: field and its definition and press the lower OK.

This may take some time, it's doing complex cutting and pasting and then refreshing of the window.

WARNING: See problems section for bad things that can happen when you add codes.

D. Recoding

Recoding is trickier for TA than adding codes. This goes through and actually replaces the codes and includes comments that the original codes had. Note that you cannot recode a search that involves an "and" (i.e. +) or is not simple. So only simple searches can be recoded. As with adding codes you are presented with a dialog that either allows you to pick from existing codes or substitute a new one to the code file:

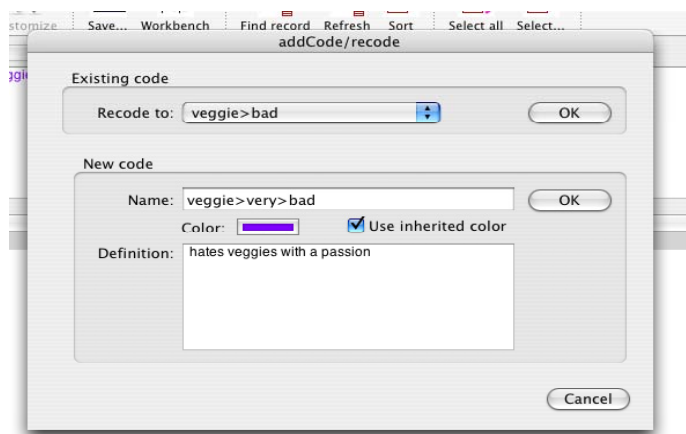


Figure 6.28. A filled in Recode dialog box

Here I am recoding the first speaker's comment giving it a more negative spin by defining a new code: veggie>very>bad. I'll click the lower "OK" since I'm filling in the lower information, and voila, the code will be changed. **WARNING: You could possibly see codes disappear from your results window since they may no longer meet the search criteria. Also see the next section for warnings about bad side effects from adding and recoding data.**

E. Problems with Adding and Recoding

Adding and recoding can make a mess of codes. The general problem is that you can land up with a nested situation which doesn't make sense to TAMS (or anyone else). If the original was

{a} This is {b} some text that {/a} will be recoded {/b}

and we recode b to a, we'll have

{a} This is {a} some text that {/a} will be recoded {/a}

Basically TA will have no idea that the second {/a} is there or which {a} it goes with, it will stop looking. And that second {/a} will give all sorts of problems in any case.

You may not get an error, but you'll get unexpected results. The answer: check the syntax by going to your workbench and running "Check for nested" from the Coding menu. Check often.

F. Updating your results

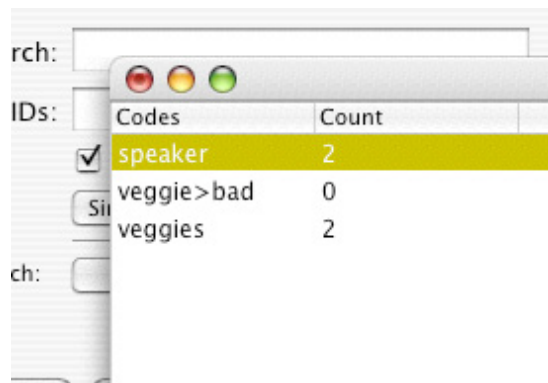
This is redundant with what has been said elsewhere, but if you see a results window with a check by the "Refresh" button, it means one of the documents that feed this results window has changed. Clicking the "Refresh" button should update your results.

Chapter 7: Reports

Getting results is one step of transforming an interview into data. An additional step is done through generating reports from your data.

A. Frequency counts

The simplest report to generate is a count of the codes in the files in the search list. This is done by having your project's workbench as your front window and picking Reports->Count from the menu. This will list the number of times each code was used in the files listed on the search list (but not broken down by file... But wait, there is a way! See data summaries below).



Codes	Count
<input checked="" type="checkbox"/> speaker	2
<input type="checkbox"/> veggie>bad	0
<input type="checkbox"/> veggies	2

Figure 7.1. Code count report

Note that this report only uses the codes in the current code set, so you can control the codes reported upon.

A. Co-frequency counts

A second easy report to generate is the co-coding frequency report. This report shows how many times a passage of text in the search list has been coded with each pair of codes in the current code set. This is a good report for analysis, to see what codes seem related to each other across the project (or portions thereof).

B. Data summaries

Data summaries are one of most complex and powerful reports in TAMS. They operate from the current selection in a results window. A data summary report allows you to group data in a column and then count matching criteria. These counts assume that data is arranged to be counted.

In other words, the data has to be sorted so that what you are grouping is together. To keep the meaningful data together you need to use the “Results->Sort up” and “Results->Sort down” menus NOT THE GENERIC sort button on the default button panel. Data summaries are described in separate documentation: *Data Summary How To*. This report can generate a frequency count by document (or speaker, or just about any other selectable/sortable criteria). For simple counts the Data Comparison Report may be simpler. Data summaries are best for getting counts of multiple nested criteria.

C. “Graph data” Output

Graphs, or dot graphs which refers to the file name extension of the files generated by this report (“.dot”) provide a graphical view of data. The actual program “Graphviz” (available at www.pixelglow.com) makes images like concept maps: nodes connecting to each other. This is perhaps the most dramatic report that TA produces. Most reports will either be available either from the workbench (i.e., with the workbench as the front window) or from a results, but not both. One simple report to run from the workbench is Reports->Graph code families.” It produces a “graph” of the tree structure of the hot code set:

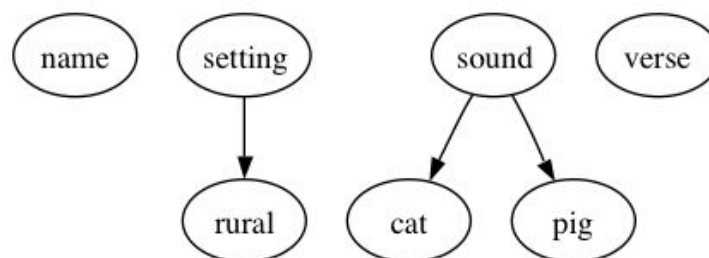


Figure 7.2. A “graph” of the codes in a project

Here sound>cat and sound>pig have been converted into a tree picture representing the coding system in this project.

Graphs of the relationships of codes to code sets as well as code sets with each other can be generated using the “Reports->Graph code sets” menu item.

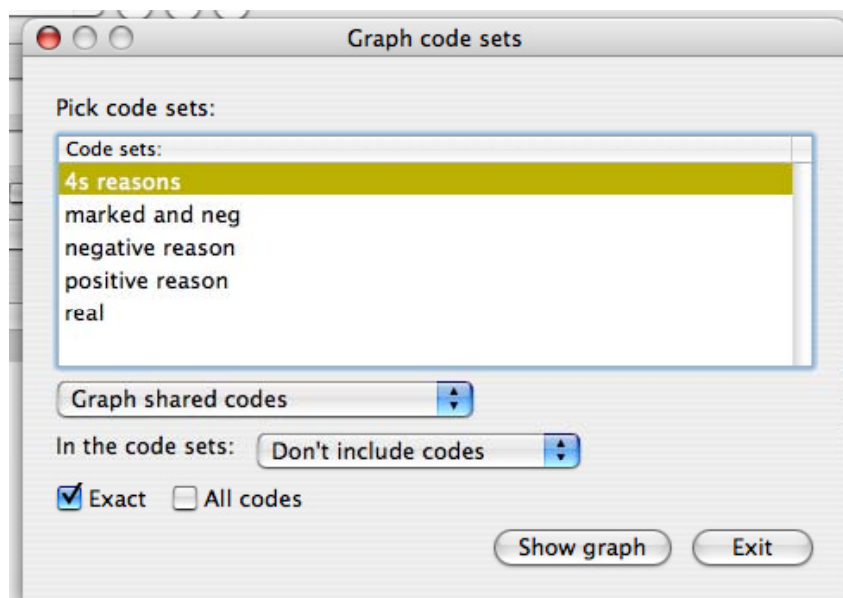


Figure 7.3. Graphing code sets (Project window version)

To generate a graph of the relationship of currently selected codes to code sets, select the code sets you are interested in examining and pick “Graph codes to code sets” from the pull down menu. That will produce a graph like this:

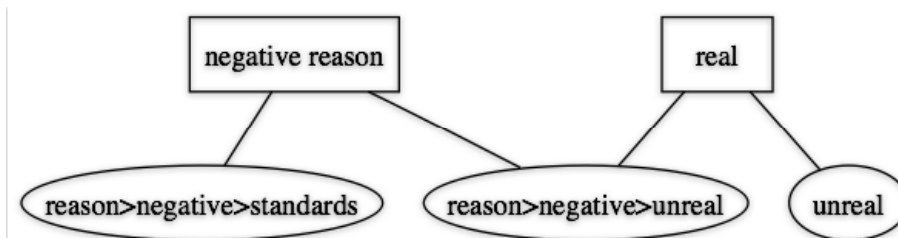


Figure 7.4. Code to code set graph

The ovals in Figure 7.4 represent codes from the current code set, the boxes represent the selected codes.

On the other hand, to generate a graph between code sets pick from the first menu in figure 7.3, either “Do not graph shared codes” or “Graph shared codes”. In addition, you have options on what is included in the box representing the code set. In this example I have included no codes in the boxes and am choosing to graph the shared codes:

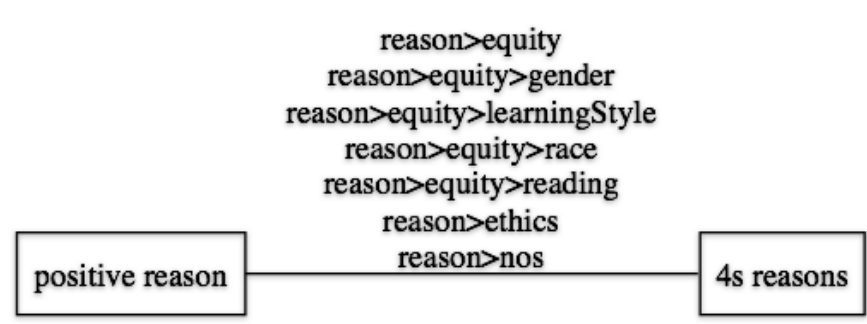


Figure 7.5. Code set to code set graph

In figure 7.5, the codes in common between the two code sets appear above the line connecting the code sets.

More complex images can be generated from result files. In result files, “Graph data” reports represent the connection between different columns (think of the columns as variables). The lines connecting the values in the columns show the frequency count.

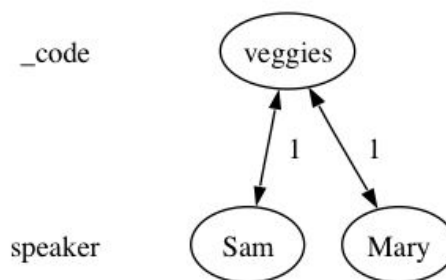


Figure 7.6. A “graph” of the `_code` and `speaker` columns of a result file

The shape and arrows of both whole levels and individual values at each level can be individually specified. See the “Dot Graph Output How To” for more instructions on producing these type of graphical reports.

D. Data Comparison Table

This is TAMS Analyzer’s primary report generator: a table that can compare two or three data columns (or code, file or value sets). The table is an HTML table generated from a result window in such a way that the elements of the table are linked back to the data in the results table: click on a cell in the report and you’ll see the relevant data in the results window. It presents either counts, like the data summary, or the actual text of various columns of the result window in a table structure. The columns and rows are either

the code sets defined in the project, your codes (either all or those in the currently selected code set), or the values of any column you choose. What is filled in as the contents of the table are the values from some other fields selected from table labeled #13 in figure 7.7, as well as (or instead with) a count of the relevant data. The idea here is that you can directly compare what different people said that you coded X. Unlike the results window which shows you one piece of data at a time, the data comparison table puts comparable data next to each other for your perusal. Note: it is not doing the comparison, you are; it's just giving you a format that allows you to do so. Also, like dot graphs, an ethos of less is more (revealing) holds here as well. Generating a 200 by 40 table of all your codes matched against all your interviews will be a mess to navigate, at best. So here's how it works:

The data, as always in TA, comes from the current selection: When you pick Reports->Data comparison table you get this dialogue, which provides slightly different options depending on whether you have selected “Codes”, “Code sets”, “Other column” from menu #2 (numbers in the following description all refer to figure 7.7):

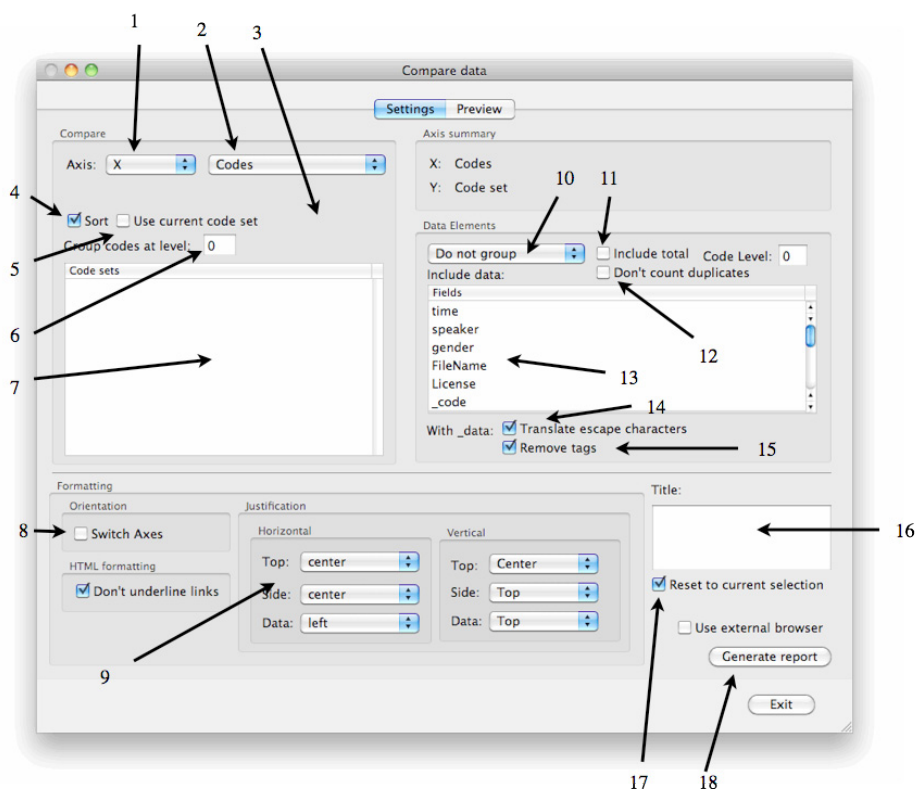


Figure 7.7. Data Comparison Table Dialogue

First you need to set the values that you are charting on the X and Y axes of the chart; X being the values moving horizontally across the chart, and Y being the values moving vertically down the chart. You will toggle whether you are setting X or Y by picking each in turn from menu #1. To set the type of data you are charting make choice from menu #2. If you pick “Codes” from menu #2, items 4 and 6 will show, but the code set list (# 7) will be empty; if you pick “Code sets”, items 4 and 6 will hide, and all of your code sets will appear in table 7 (same for value sets and file sets), and a checkbox called “Exact” will appear at spot #3. If you pick “Other column” a menu of columns you can choose will appear in spot #3.

The default table will have your codes across the top, and a single row called “All” that contains the data for those codes. This will make more sense if you click #8 (Switch axes); then your codes will appear in a list, and their associated data will appear in a single column vertically. In general, if you want to reverse the columns and rows, click on switch #8 “Switch axes”, which will put your choice for X (menu #1) from menu #2 down the table, and the values from menu #2 when menu #1 is set to “Y” across the table.

If you choose “Codes” for #2, the table will use all the codes from the currently selected rows. If you check box 4, the current code list (the “hot” code list from your work bench) is used as the basis for selecting which rows to show. Only rows in the current selection that have a code included in the current code list will appear in the table. Use the exact switch (item 3, hidden in the above picture) to determine whether the codes of the current selection have to exactly match the current code set (a>b will not match a) or not (a>b does match a, it’s in its family). **Note: the exact switch is only visible if menu 2 is “Code sets”!**

For a code data comparison table you can group codes together by setting a code level. This is similar to autosets and other tables. If you put “1” into item 6 a>b and a will be treated as the same. If “0” is entered all levels are matched—codes will not be grouped together.

For all types of tables (codes, code sets, other column, etc.) you can use switch 11 to include a count with your data. You can also group and provide subcounts of the data that you display in the core of the table, which is

controlled by selecting fields from 13 and by your selection from menu 10. **(Warning: These counts may not be meaningful if you are using sets in your table, simply because single items may appear in multiple sets.)** Using menu 10, you can either show all the content of the fields each row listed separately even if they are highly redundant (Do not group), group matching instances from the field (Group), group and show a subcount of repeated entries, or just show a count. The count can represent the number of records or the number of unique values, depending on whether 12 is checked. You can select multiple fields from table 9 (hold down the apple and/or shift key to select fields; you can also drag fields to change their order. If multiple values are selected in table 13, a final report with sub counts will not give separate counts for each field, but treats the data from the fields in table 13 treated as a single entity.

If you are including your “_data”, which typically you will, you can have the program remove or include tags as well as convert escape characters (\n for a new line, \t for a tab) using items 14 and 15.

For code set or value set or file set data comparison tables (menu #2) you will be provided a list of project sets in item 7. You must select the code sets you want included in your table from this list. What these are compared to depends on what you pick when you toggle menu #1 to the other axis. You can compare codes (from column _code) from the records of the current selection of your results window or other sets or column values. These are compared to the selected sets either exactly or not depending on the state of the exact switch (here hidden in area 5). The one novel option on when Y is chosen as the axis is the “Single column” option on menu 2 (it will be invisible when menu 1 is set to X). Then use the menus, tables, and check boxes 10-18 to determine how the information will be formatted.

Example #1. I want a simple table of what data has been coded “reason>ethics” from my data. I start by selecting in my results window all of the cases of “reason>ethics”: I simply select the _code column and pick *Results->Select...* and fill in “reason>ethics. Now I pick *Reports->Data comparison table*. Starting with menu #1 set to “X,” I would pick codes for menu #2; Then I would set menu #1 to “Y” and would pick “Other column” from menu #2. A new menu will become visible in area #5. From it I would pick “FileName.” From menu #10, I would pick “Do not group” (for this

report it wouldn't have made a difference, since none of my “_data” is identical with each other). I've picked _data from table 13, since I want to see what they said and this is in the _data column. Clicking Generate report (#18) provides the following table and allows you to see all the values of your _data in the current results table:

"FileName" compared against "Codes"

FileName	reason>ethics
Erika	_data: I know with the 6th day it was part; I took just a clip out of it; and it was the doctor who created cloning; they were opening a big lab scene to everyone so it's a big opening yet there are all these protesters; like in the future; like they were all yelling; And he had to defend his like why have you a human cloning and y'know because the news people were asking him questions and so it was kind of the just that part. kind of the ethics of it and seeing what might happen
Cyndy	_data: Like was it even ok for them to do this experiment with this chimpanzee? They got cool results but you can start talking about that kind of stuff too.

Figure 7.8. Screen shot of a data comparison table

Example #2. Imagine I want to know what codes relevant to a specific code set my informants used. Again, I would select data that I'm interested in (e.g., the informants of interest) and pick “Reports->Data comparison table.” This time I will pick “Code sets” from menu #2 when menu #1 is set to “X.” The available code sets in the project will be listed in table #7, and I would select the ones I am interested in. Changing menu #1 to “Y”, I would pick “Other column.” From the new menu that appears in area #5, I would pick “Filename” (or if I had an informant context variable I'd pick that). From table 13 I'm picking _code. When I hit generate report (button #18) I get this (this is just a partial picture):

"FileName" compared against "Code sets"

FileName	negative reason	positive reason
Cyndy	reason>negative>image	reason>content
	reason>negative>read	reason>discussion
	reason>negative>read	reason>discussion
	reason>negative>subjective	reason>equity>gender
		reason>ethics
		reason>ethics
		reason>hot
		reason>hot
		reason>misconception
		reason>misconception

Figure 7.9. Screen shot of a report of `_code`

I picked two code sets from table 12 (“negative reason” and “positive reason” and I can see what codes Cyndy had here side by side. But there’s a problem. There’s a lot of redundancy here. This can be cleaned up by changing the value in menu #10. Click the settings tab, and pick *Group and subcount* from menu #10. Click generate report (button #18) and the data is significantly summarized:

"FileName" compared against "Code sets"

FileName	negative reason	positive reason
Cyndy	reason>negative>image (1)	reason>content (1)
	reason>negative>read (2)	reason>discussion (2)
	reason>negative>subjective (1)	reason>equity>gender (1)
		reason>ethics (2)
		reason>hot (2)
		reason>misconception (3)
Erika	reason>negative>politics (1)	reason>ethics (2)
	reason>negative>sensitive (1)	reason>relevance (2)
	reason>negative>subjective (1)	
	reason>negative>unreal (1)	
Total	8	21

Figure 7.10. Table after grouping and subcounting

By the way, if I want to look at the two instances of `reason>hot` that are listed for Cyndy in figure 7.11, I can click that table item and the result window will show just those two records. While useful, this changes the current selection, so if you want to further adjust your data comparison table use the back button to restore the selection of interest before making further adjustments.

If I wasn't interested in counting each instance of a code used, but just wanted to know the total number of different codes each informant used, I can obtain this by going back to the settings tab, picking "Group data" from menu 10, and making sure items 11 and 12 are checked. Now click generate report (#18):

"FileName" compared against "Code sets"			
FileName	negative reason	positive reason	Total
Cyndy	reason>negative>image	reason>content	10
	reason>negative>read	reason>discussion	
	reason>negative>subjective	reason>equity>gender	
		reason>ethics	
		reason>hot	
		reason>misconception	
		reason>motivation	
Erika	reason>negative>politics	reason>ethics	6
	reason>negative>sensitive	reason>relevance	
	reason>negative>subjective		
	reason>negative>unreal		
Total	7	9	16

Figure 7.11. Counting without duplicates

There's actually one problem with this report, which is that total count includes both columns, and it would be better to actually generate this report for each column separately. No problem, click the settings tab; only select the *negative reason* code set and click generate report:

"FileName" compared against "Code sets"		
FileName	negative reason	Total
Cyndy	reason>negative>image	3
	reason>negative>read	
	reason>negative>subjective	
Erika	reason>negative>politics	4
	reason>negative>sensitive	
	reason>negative>subjective	
	reason>negative>unreal	
Total	7	7

Figure 7.12. Negative reason count

I'll want to go back and see positive reasons, so I'll need to save this for later; I'll need to spin it off, as it were. So click the *Open in window* button at the bottom of the window. Now I can go back, click the settings tab, and select the code set called "positive result" and click *Generate report* again.

This example illustrates the typical trial and error that happens in generating comparison tables. The metaphor for this dialogue box is a sandbox. Try different settings, click "generate report" and see if you're getting the results you want. By picking "Counts only" or using *Include Total* (#11) and *Group and Subcount* (#10) you can replicate a lot

of the data summary table! It's the easiest way to get simple counts and to see your data in side-by-side glory.

The data comparison table is even more powerful than just a report generator, because it is interactive and can help in your analysis. The content of the table can be clicked on and that data will display in your results table (be careful as this can throw you off if you want to modify your comparison table since the selection of interest will have changed. Use the "back" button (on the status bar) to restore your selection. If a given entry refers to multiple, such as if you group data (to remove redundancies) clicking that entry will pull up the rows in that group as a selection in the results table! See *tamsZine #2* for more examples of using the Data Comparison Table report generator.

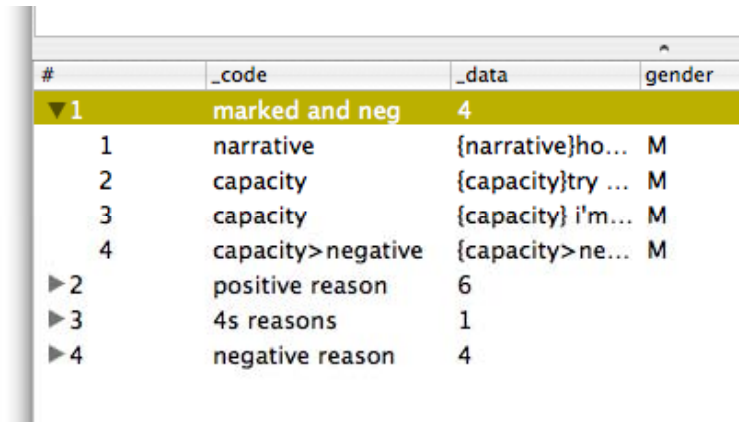
Chapter 8: Analysis of Code Sets

A. Using code sets as an analytic tool

At late stages of analysis, often codes are not good enough to serve alone as defining the categories you are interested in tracking. The real dependent variables you are tracking will probably exist in a variety of code families rather than in one genetic tree. Code sets were designed to provide cross-code-family analysis. Until TA3.1 there was no easy way to get counts of the relationship between code sets and other data categories. Here I will briefly discuss the several key means of mining code set relation information from your data: grouping by code set and graphing code sets. While this section should really have followed section VI, the fact is that it uses a lot of the ideas introduced in VII, so I am concluding this manual with a discussion of using code sets.

B. Grouping your data in code sets and getting a count

The simplest analysis you can do is to simply see how a selection of results records map onto the code sets you've defined. This is done by picking "Results->Code Sets->Group by code set". This will clear away any "Select near" analysis you have done, and show you an outline with each code set represented at level 1, and matched records underneath at level 2.



#	_code	_data	gender
▼ 1	marked and neg	4	
1	narrative	{narrative}ho...	M
2	capacity	{capacity}try ...	M
3	capacity	{capacity} i'm...	M
4	capacity>negative	{capacity>ne...	M
▶ 2	positive reason	6	
▶ 3	4s reasons	1	
▶ 4	negative reason	4	

Figure 8.1: The results of Group by code set

The name of the code set is placed in the “_code” column, and a count of how many records are matched with that code set is placed in “_data”. In figure 41, I've moved (dragged) the columns so that _code and _data are first for instructional purposes. You may find you have to scroll over to find your _code and _data columns.

Note that these results can all be moved to other programs or saved through the “Files->Export data” operation. Just make sure that the outline view is showing.

By default codes are matched with code sets “non-exactly” meaning “a>b” is matched with a code set if it has code “a”. You can force an exact match by selecting “Results->Sort options->Case sensitive”.

C. Counting the relations between variables and code sets

More often you will want to see how the values of a repeat code are matched with your code sets. Starting with TA3.1 there is a simple way to get information like this in two simple forms: a table and a graph. Once you have the selection of data you want to analyze this way simply pick Reports->Graph code sets. Note that this will first do a “group by code set” which will obliterate any “Select near” data you’ve gathered.

The following dialogue will appear:

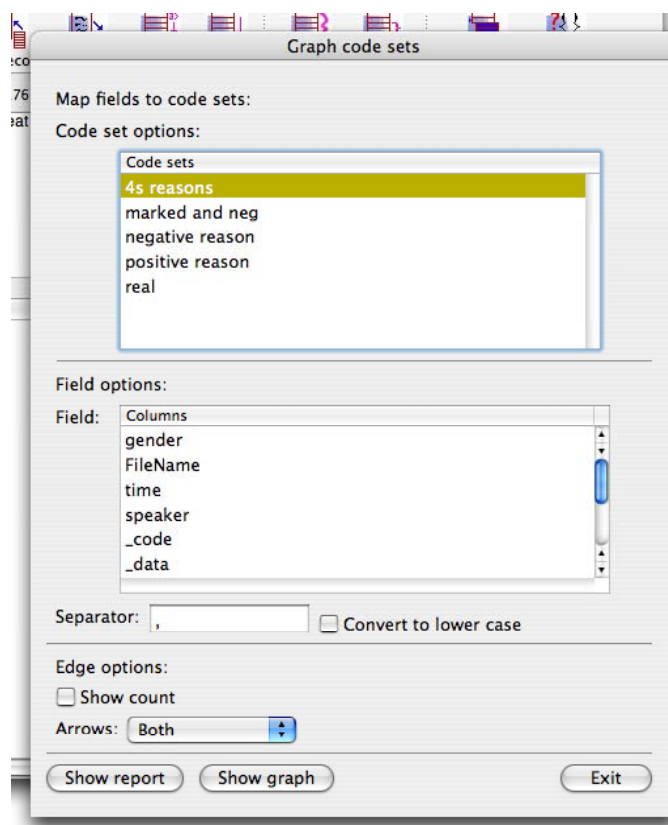


Figure 8.2: Graph code sets (Result window version)

At the top you can pick the code sets you want to study. Below you pick the fields (columns) you want matched with those code sets. Note that you can pick more than one field, in which case the values of those fields are combined using the separator and all values will have to match to be counted. This is useful if you want data separated by file for instance if a value crosses different files. Note that the separator can use “escape characters.” Use `\t` for tab and `\n` for return.

The third part of this dialogue is for the “graph” output. You can control how arrow heads appear and whether a count is shown over the arrow. Note that the report (which opens in your browser) ignores these options and provides a count. In this example I’ve picked both “positive reasons” and “negative reasons” from the top list, and gender from the bottom list, clicked show count and then clicked show graph to get this “graph” of the gender relationship between code sets positive and negative reasons:

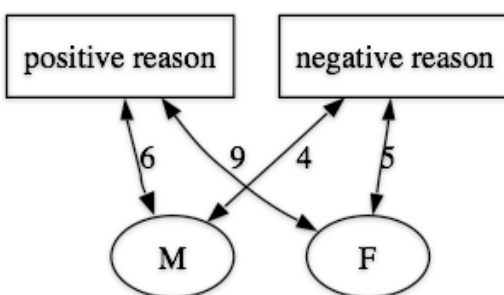


Figure 8.3: Graph output from Graph code sets (Result window version)

If I click on “Show report” I get the table shown in figure 8.4 in my browser:

"gender" mapped to code sets

gender	positive reason	negative reason
M	6	4
F	9	5

Figure 8.4: Report output from Graph code set (Result window)

Appendix 1: Converting from earlier versions of TA

Converting from TAMS 3 to TAMS 4

There is nothing to do. TA3 files are completely compatible with TA4. The only issue is that there is no backward compatibility with tamspdf and tamsgraphic files and any result files that reference them.

Converting from TAMS 2 to TAMS 3

To facilitate moving to the new directory structure TAMS has a new conversion menu item to help create the directory structure and move the files into the proper folders. To convert from earlier (2.x) versions of TAMS do the following

1. Save the project in a new empty folder as an XML TAMS Project file. The program will say something about saving as absolute paths, just click “Ok”
2. Pick *Project->Convert->Convert to ver 3 directory structure*
3. You will then be walked through a number of steps. First creates the directories, then it tries copying the files into the directories (assuming files are not already there)

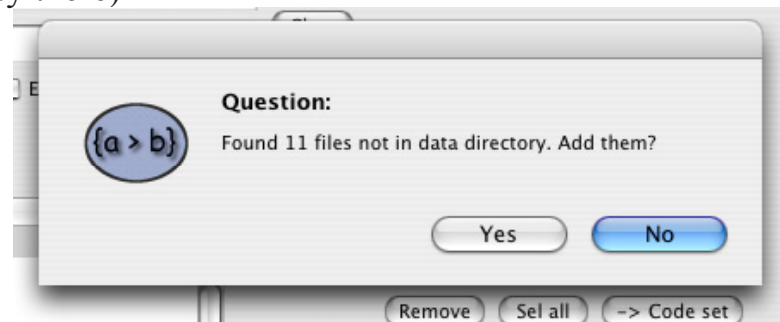


Figure A.1

Answering Yes here will copy (not move) the missing files to the ./data directory

- a. You may be prompted if it can't find a file whether you want to remove the “dead record” from the project file. If you say NO you can locate the file and drop it in the /data directory. If you say yes you can always find the file and import it later.

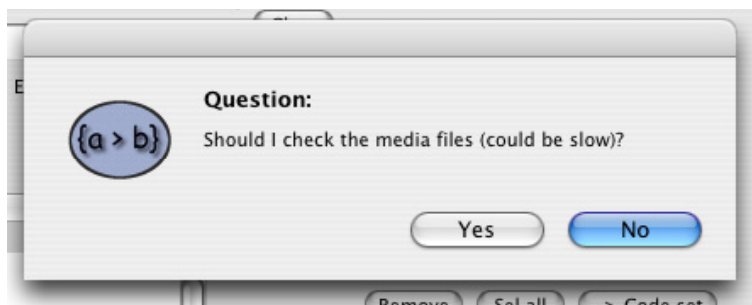


Figure A1.2

Answering yes will go through each data file looking for the !mediafile metatag. If that file exists in that spot it will copy it into the ./media directory of your project. If you have moved your media files, it will not find them, however. You can still use your media file by dragging them to the ./media directory yourself. The program always checks there for your media first (even before the directory specified in the !mediafile tag!). If you answer no, no attempt will be made to suss out the media files locations. If you have mp3's or other media attached to your documents you probably will want to move them to the media directory.

- b. The program then report any files it could not find on the hard disk. There's nothing to do here but click "ok."
- c. The process then repeats itself with result files. It starts by asking if you want to move result files to the ./results folder:

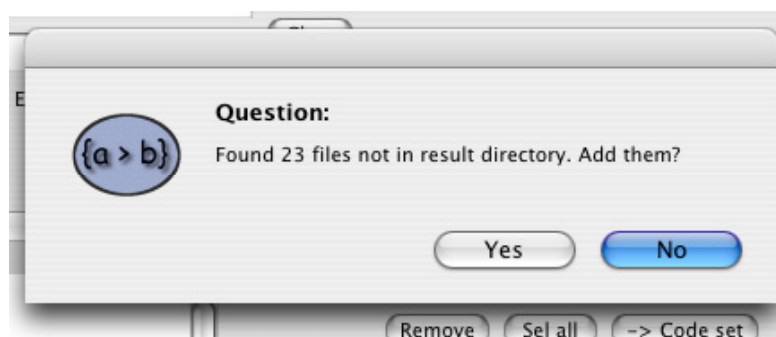


Figure A1.3

Click yes to have these results files copied to the ./results folder.

- d. The program then asks if you want to remove lingering, crufty, useless data that it is holding onto about various files that it still has data

structures for. This is a good time to clean up the project file by saying yes to questions like:

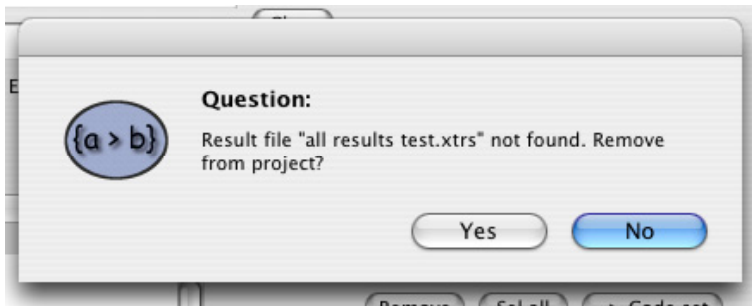


Figure A1.4

Voilà, you are done. You now have a TAMS Analyzer 3 project. One thing you might want to do is to add this new project to your work menu and get rid of any old TA 2.XX projects from the work menu you have converted.

Appendix 2: Types of codes

TAMS has traditionally works with three types of codes:

1. universal codes which are generated for every results window record and hold their value through the whole document. The syntax is as follows: `{!universal codename="mydata"}`. I use “{“ since most documents don’t use this. At some point I will add an escape character so you can use “{“ without triggering a response by the processor. A typical example would be `{!universal type="interview"}`
2. repeat codes which change their value throughout the document and generate a separate value in each record produced. These mark distinctive attributes for a section of a document (marked by `{!end}` or `{!endsection}`): typical repeat codes include speaker, time, question all of which you would want to be attached to a passage of text you have marked (coded) in some way. To indicate that the code “author” is a repeat code place `{!repeat author}` toward the top of the document or in the Init file. Then mark each occurrence of author by surrounding it with a start and end tag: `{author}Matthew Weinstein{/author}` (notice that unlike the `!repeat` command there is no `!` mark here, this is just a data code). In an interview the speaker might be marked as a repeat code. In field notes the time of an observation might be a repeat code. In our project which involves numerous newspaper articles per document, author, title and date of each article were repeat codes.
3. data codes. These are the passages that you are really interested in. These are marked with `{code}interesting passage{/code}`. You don’t need to declare anything, just fill in the word code with whatever you see fit: `{sample}Just some text{/sample}` is now coded as sample! Note codes consist of numbers, spaces and underscore (“_”) characters. **No spaces permitted**. Passages of text can have multiple codes; codes can be nested and can overlap.

Types 2 and 3 are now (since TA 3.30) combined into a more general idea of a context variable. Users can declare the name, value and “horizon” of a variable. The horizon is the point at which the variable loses its value (set to an empty string: “”). The horizon can be sent to “end” “endsection” “eof” (end of file), and “never.” This expands the previous types of variables which were limited to the first 2 horizons. Using the `!var` metatag, the user can specify these parameters as follows:

108

```
{!var name="x", value="my value", horizon="Pick one:  
endsection|end|eof|never"}
```

This is setting !context codes (what used to be repeat and universal codes), not data codes.

Appendix 3: Working with large projects

TA has a large variety of tools to help researchers manage large numbers of files and data. I hope in this section to just point them out and give some tips; I will not be doing much in the way of hand holding.

Named searches. The first tool for large projects is the possibility of saving and restoring different search lists. This way you can just by picking an item off of a menu pull up a search list that just has memos, for instance. The search list menu is just above the search list. To save the current search list (and init file) just press the + button above the search list on the work bench. You will be prompted for a name. To restore a saved search list, just pick it from the menu above the search list. To delete a searchlist, first select it from the menu and then press the – button. You can clear the search list of all search lists by pressing the - - button.

The init file. Perhaps the most important element for multiple file searches. Conceptually the idea is simple. The init file is simple a file with commands (metatags) that you want designated as the start of a search. It is just the first file searched. To indicate which is your init file just select the file from the file list on the work bench and press the button marked “Init File”. The init file is subsequently listed at the bottom of the work bench.

There are five tags that I usually will put into an init file: a !button command which provides project wide code buttons for the document button bar, an empty !universal command which simply lists the universals in the order I want them to appear in results windows: {!universal name=“”, city=“”, type=“”}. At the top of each document I will have a !universal with those values filled in (in whatever order). Third, I will declare my repeat codes in the header: {!repeat speaker, time, gender}. Fourth, if I am using an !inner or !last repeat I will declare that value in the init file as an indicator of the structure of my files. Finally, I will use the !if statement to attach any additional information to my result files: {!if speaker=“Amy”=>gender=“F”}. If I have declared a repeat value gender it will automatically be assigned a value “F” everytime speaker becomes “Amy”.

File sets. TAMS allows you to group your files together so that the file list can display just a subset of files rather than the whole body of data in your project. You can create a file set for just memos or just interviews. This is selected by the little set menu that is to the left of the file list. Every project

always has one set already established which is the results set, which contains every saved (and open) results window/file. In fact the menu item “View all files” should really be view all files except the result files. See the code sets section on how to operate the dialogue box that comes up with manage file sets. They are identical.

Inter-rater reliability. TAMS can calculate the interrater reliability as both a percent and as kappa. This requires the construction of special “test” files that two coders will attempt. See the Inter-rater reliability information folder in the How To folder in the docs folder.

Line number documents. TA can wrap and line number your interviews and field notes to help discussions of particular lines. Make sure you turn on the “Scan for line numbers” option in the results so that line number information is included in results.

Shareable project format. By default TA is *not* shareable between machines. If after initially saving your project you change the project type to Same folder or relative, keeping all of your files either in a single folder or in the same file tree respectively, the project will be shareable. If a colleague gives you a project folder that can’t seem to find its files then move all files to a single folder and change the project type to “Same folder”, close and reopen the project. These options are in the Project->Preferences menu option (not the usual preferences).

Multiple coders. There is support for codes being “signed” by different coders; searches can be coder specific. The signature is specified in the preference panel. Searches can be done for specific coders. One can even do searches that amount show me passages of veggie coded by MGW and passages of fruit coded by LJS (where MGW and LJS represent 2 different coders).

Polymorphous data. The search list can contain multiple types of data. To facilitate mixing data types, the !map metatag allows researchers to have different codes in the same repeat data column.

Appendix 4: Metatags

- `{!appendcomment X}` = appends X to the comment established with previous `!setcomments` and `!appendcomments`.
- `{!backloadrepeat}` = for a coded passage that crosses an `!end` value, the record will have its repeat values assigned from the values at the end of the passage.
- `{!block X}` = if repeat code X has already been declared then no values for X will be returned in results (X will be ignored)
- `{!bookmark X}` = designates a bookmark in the document file (accessible through the Coding->Bookmarks menu)
- `{!button X}` = creates a button bar with button X.
- `{!clean}` = designates that subsequent repeat values should have their values cleared at `{!end}` metatags
- `{!clearcomments}` = clears all existing comments set by `!setcomment` and `!appendcomment`.
- `{!codedobject X="Y"}` = special code generated by TAMS to represent graphical objects like selections of an image or PDF. This is converted more or less straight into a row in a results window.
- `{!comment X}` = See `!dummy`.
- `{!context X}` = See `!repeat`.
- `{!contextcode X}` = See `!repeat`.
- `{!date X="Y"}` = See `!dateformat`
- `{!dateformat X="Y"}` = sets the print format for dates and times. Used in sorting and grouping date/time data. X is the name of the context variable containing date/time information. Y is the Apple format string as described at <http://developer.apple.com/documentation/Cocoa/Conceptual/DataFormatting/Articles/dfDateFormatterSyntax.html>
- `{!dirty}` = designates that subsequent repeat values do not reset their value at an `{!end}`
- `{!dummy X}` = `{!comment X}` = a do nothing code which allows you to insert parenthetical comments into your code. Synonyms:

`!comment.`

`{!emptysection}` = specifies that the program should report on empty `{!endsection}`s when doing empty searches

`{!end}` = marks the end of a section; by default `!repeat` values are cleared. Data found is stored.

`{!endcomment X}` = clears the comment set by previous `!setcomment` and `!appendcomment` of X. X has to match verbatim a comment added using `!setcomment` or `!appendcomment`. `!endcomment` with no argument (i.e., `{!endcomment}`) clears all comments.

`{!endlastcomment}` = removes the last comment set by `!setcomment` or `!appendcomment`. To remove a specific comment use `!endcomment`.

`{!endsection}` = like `!end`, but `!repeat` codes keep their values, they are not cleared

`{!eofisend}` = end of file is treated as `{!end}`

`{!eofisnotend}` = end of file is not treated as end

`{!escapeoff}` = treat “\” as a regular character

`{!escapeon}` = attend to “\” as an escape character, i.e., it is a flag to TAMS to not treat the next character as special. Useful for texts that have braces (`{` and `}`) in them.

`{!first X}` = See `!inner`.

`{!goto file="X" bookmark="Y"}` = acts as a link between the metatag and a location in another (or same) file called X marked by a `!bookmark` called Y. Either X or Y can be left out. If X is missing it will look in the same file as the `!goto`; if Y is missing, it will open X and just scroll to the top.

`{!frontloadrepeat}` = for a coded passage that crosses an `!end` value, the record will have its repeat values assigned from the start of the passage.

`{!if X="something" => Y = "a value"}` = For already declared universal or repeat code values X and Y, the program will automatically assign Y to “a value” every time that X is assigned to “something”. X and Y must both be repeat values or they must both be universal

values; you cannot mix and match.

{!inner X} = short for innerrepeat; designates a code already declared as a repeat code should be treated as if it had a {!endsection} before it. Warning: !inner cancels !last and vice versa. Synonyms: !first, !innerrepeat.

{!innerrepeat X} = See !inner.

{!last X} = short for lastrepeat; designates that existing repeat code X will always be the last repeat for a section and that the next occurrence of any repeat code after X should be treated as having {!endsection} before it. Warning: !inner cancels !last and vice versa. Synonym: !lastrepeat.

{!lastrepeat X} = See !last.

{!map X->Y, A->B, ...} = specifies that instances of repeat value X should be put in the column called Y (mapped into Y), values of repeat value A should be put in the column called B. X, Y, A, and B must all be designated as repeat values. Y and B should be repeats at the beginning of your search list so that columns are created for them.

{!videowindow} Tells TAMS to open a document in the large video playback layout.

{!name X} = creates or assigns a universal code called FileName to value X.

{!noemptysection} = specifies that empty searches should only be returned for !end's not !endsections.

{!noheader} = specifies that result files should not produce a header row when saved as text

{!noquote} = specifies that quotes should be converted to escape characters.

{!noskipinneratend} = overrides the default behavior in which after an !end or !endsection metatag the first encountered occurrence of the !inner repeat value is not treated as having an !endsection before it. Using this metatag treats the first occurrence after an !end as having an !endsection in front of it.

{!noskipinnertopofdoc} = overrides the default behavior in which the

first time the !inner repeat code is encountered at the start of the document it is not treated as having an {!endsection} in front of it.

{!nozapuniversal} = specifies that universals should not be cleared at end of file

{!repeat X} = used to indicate that code X is designated a repeat code.
Synonyms: !context, !contextcode.

{!setcode X, Y, Z} = for structured documents, codes the entire section from the metatag forward with codes X, Y, Z as data codes. Takes its horizon as either endsection or end, depending on the value set by !virtualend or !virtualendsection (which is the default)

{!setcodeinfo codes="X, Y, Z" coder="coderName" horizon="end|endsection"} = for structured documents, codes the entire section from the metatag forward with codes X, Y, Z as data codes. Sets the coder for each of those codes to coderName, and takes its horizon as either endsection or end. Either the coder or the horizon part can be omitted. If the coder is omitted, then blank is the coder; see !setcode for a discussion of how the horizon is determined.

{!setcomment X} = appends comment X to the comments provided in all subsequent end tags. Using this metatag with no value, i.e., {!setcomment} clears the comment. This also replaces any previously established comment created with !setcomment or !appendcomment with X

{!setcontext X="Y"} = same as !setrepeat

{!setrepeat X="Y"} = sets existing repeat code X to value Y—if X does not exist it creates it.

{!skipinneratend} = reasserts the default behavior in which after encountering a !end or !endsection, the next occurrence of the !inner repeat code is not treated as having an !endsection before it. All subsequent occurrences in the document are treated as having an !endsection before it.

{!skipinnertopofdoc} = reasserts default behavior in which the first occurrence of the !inner repeat code is not treated as having an {!endsection} before it. All subsequent occurrences in the

document are treated as having an !endsection before it.

{!struct} = Indicates that a document or document part is structured, i.e., is broken up using !end, !endsection, !last or !inner. Important for conducting section searches.

{!universal X="Y"} = creates or assigns an existing universal code X and assigns it value Y

{!unstruct} = indicates that there is no structuring elements. Program treats each close tag as though it had an !endsection immediately after it. Section searches are pretty much meaningless. This is the default state of the tams interpreter.

{!var name="X", value="Y", horizon="endsection|end|eof|never"} = Declares a variable X, sets its value to Y, and sets the point at which it is automatically blanked out (assigned to the string ""). The "horizon," the point where it is zeroed can be an endsection, an end, the end of the file (eof) or never. Horizon and Value are optional components; multiple declarations can be made in one !var statement by separating the declarations with semicolons.

{!virtualend} = indicates that subsequent !first and !last metatags should insert !end}s into the document. For contrast see !virtualendsection}

{!virtualendsection} = indicates that subsequent !first and !last metatags should insert !endsection}s into the document. For contrast see !virtualend}. This is the default behavior.

{!zapmap} = Clears the map of all entries.

{!zapuniversal} = specifies that universals should be cleared at the end of each file

Appendix 5: Preferences

Coding preferences

Prompt for new code definition: Should TA ask you for a code definition when you enter a new code (i.e., press the new button on the document window)?

Use time-date stamp in new code definition: Should TA prefix a new code definition with a cryptic time/date stamp

Take the code by double clicking list: Is double clicking the code list your preferred way to code data? Turn off if you want it to take the code from the little box under the buttons on the document window (fill in box and click code)

Code list reveals codes as tool tips: a legacy from before TA used split views to accommodate long codes. Codes appear as tool tips. Slows down the system significantly. Not recommended.

Use coder id: insert a coder id (you provide it in the box) as you code.

Scan init file... : Should TA scan the init file every time you open a document to see if there are !button metacodes?

Color tags: What is the default color for meta and code tags?

Automatically refresh tag colors: Should TA automatically refresh the colors when a file is opened or other changes occur? Could slow down your system. You can manually uncolor and recolor tags from the coding menu.

Display comment dialogue for code button: rather than manually typing comments into the end tag you can set this up so that when you single click a code and then use the code button you are given a dialogue box into which to put your comments.

Use HH:MM:SS format: If checked insert time code and the display of media time will be in HH:MM:SS format. If unchecked the program will use raw seconds for both insert time code and the display of media time in both the document and results windows. (Note, this does not change your data from seconds to HH:MM:SS format. Use the convert menu options on the Coding->Audio/Visual menu to transform your data to one format or the other.)

Max graph edge width: Sets the amount that TAMS will use as the maximum width when scaling graph edges for Graphviz.app.

Vary pitch when rate changes: Normally varying the playback rate of media files will change the pitch. Note that **NOT** varying the pitch does seem to add distortion.

Searching Preferences

Detached result sheets: Should dialogue boxes float above all windows or be attached to them (affects some not all dialogues)? Recommended.

Autoreload after... : Should the results window refresh after recoding or add coding?

Report empty...: Should an empty search report at `{!end}` or `{!endsection}`?

Show coder...: Should the coder be reported when examining a raw search

EOF is same as `{!end}`: Should the end of a file be treated as if it had an implied `{!end}` there

Evaluate repeat variables...: If checked, coded passages that cross an `{!end}` boundary are evaluated in the last section, other wise they are evaluated at the first section that the coded passage includes.

Universal variables zapped...: Should universal variables carry their value from document to document or be cleared?

Include repeat variables...: If checked repeats are treated just as normal data codes when doing a non-simple search.

Use old Mac new line...: When doing a save to of result files should the program use OSX file new lines or older style new line characters (if you are exporting to programs that are run in classic you probably need this checked).

Export result file format: This sets the character set of data being exported when a result file is saved with “save to” type text.

Number of characters...: This sets how much (maximum) context should be added when doing a search. As of 3.5, this affects all types of searches and controls the default maximum setting of the context or padding controlled by the slider at the top of results windows.

Recognize ‘\’...: Should ‘\’ be taken as itself or as an escape character?

Scan for line numbers: If you use the line number and wrap feature you should have TA scan for line numbers: adds a line number to the first line of

results.

Unmark...: Should marked records be unmarked after doing a recode or addcode?

Update results...: Should TA attempt to revise your results based on addcodes and recodes. This is an art (and I'm not an artist). If you really need to see accurate results uncheck this and refresh your windows.

Save Graphviz files: If you want the default response of Report->dotGraph Output to be to save the report (rather than open it directly in graphviz) check this box. Useful if you want to use X11 graphviz software rather than the aqua version.

Enable the back button...: Should the back (and forward buttons) of the result window button panel be enabled? Why disable them? They use a lot of memory!!! If you're close to the limit you might want to turn off this feature.

Open document...: Should TA open affected files as it recodes, add codes etc.? This can get quite messy, with possibly dozens of files being opened, but these can then be "undone" file by file. Otherwise (unchecked) add code and recode etc. work in the background and are not undo-able.

Documents treated as structured...: When doing a search should TA treat the document as structured, which means that repeat values aren't matched up with data until an explicit or implicit (set through !last or !inner) {!end}/ {!endsection} is found? Otherwise the program assumes documents are unstructured which means that every end tag is treated as if it has an !endsection following it.

Appendix 6: TAMS Document Toolbars

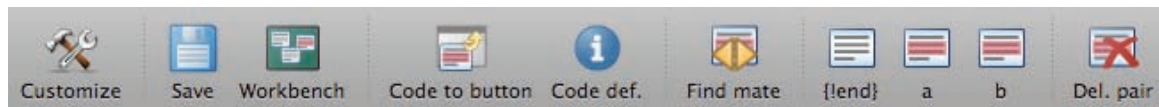


Figure A6-1

TAMS Analyzer’s document and result windows both sport toolbars. Result window toolbars are fairly traditional OSX style toolbars. You use the “customize” button and you can pick the features you want on the toolbar and this will change the toolbars on all result windows.

Document window toolbars are very different, however. The main purpose of them has been to keep handy codes and text (including metacodes) that are used frequently or that you want to have handy (visible and single-clickable). These toolbars can be customized for different projects and each individual document, given that the codes and text you want to have access to are likely to vary a lot! You can still use the customize button, but when the window closes your changes will not be saved.

To put a code on the toolbar temporarily (available until you close the window) click on the code in the code list on the left side of the document window and click the button marked “Code to toolbar” The code will be cropped at 10 characters, but you can see the whole code if you just float your mouse over the toolbar button. You can code text with this button (i.e., with this code) just by selecting the text and clicking this button once. Coding buttons are indicated by a question mark in braces: {?} (In the picture above this is shown for codes a and b above).

To put other text in the toolbar just select the text and pick “Coding->Toolbar->Add selection to toolbar” from the TAMS menu. Again, the text under the button will be cropped at 10 characters, but the whole text will appear if you float the mouse over the button. Clicking this button will replace whatever text is selected or if there is no selection, it will insert the text where the cursor is. This contrasts with the coding buttons which surround the selected text with the code in braces. Text buttons are indicated with an exclamation mark in braces: {!} (see {!end} in the example above).

To put a metacode in a toolbar button just type the text in (e.g. {!end}) and select it and pick “Turn selection into toolbar button” from the TAMS menu.

For a variety of practical reasons, TA does not remember your button bar from time to time or window to window. You can have the program automatically assemble a button bar for you however with the `{!button ...}` metatag, which should appear at the top of the document. Four types of items can follow the word `button` in the tag, commands from the Coding menu, the name of a code (without quotes), a passage of text in quotes, and the vertical bar to indicate a separator. The 15 commands that TAMS 3.53 supports are shown in table A6-1.

To indicate that you want one of the commands listed in A6-1 on your toolbar simply use the Toolbar equivalent from the A6-1 in a `!button` statement placed ideally in your init file or at the top of your document.

You do not need to memorize these, they can all be selected from the Coding->Toolbar->Commands submenu.

So to create the toolbar shown in figure A6-1 at the start of this appendix use this metatag:

```
{!button cmd::findMate, cmd::decolorTags, cmd::colorTags,|, "{!end}" ,a,b, |,cmd::delTagPair}
```

The vertical bars are added to the tool bar with the `|` character (the shift of the `\` key).

To create project wide button bars put the `!button` metatag in the init file; you can then add additional buttons in a `!button` metatag placed in the specific document files.

Finally, for your `!button` to take effect, save and close your document and reopen it (double clicking it from the workbench) and voila, your tool bar.

Menu option	Tool bar equivalent
Coding->Toolbar->Add selection to toolbar	cmd::selectedTextToButton
Coding->Toolbar->Add code to toolbar	cmd::codeToButton
Coding->Code definition of selected text	cmd::selectedTextCodeDefinition
Coding->Code definition	cmd::codeDef
Coding->Delete code pair	cmd::delTagPair
Coding->Find next code	cmd::findNextTag
Coding->Find current code	cmd::findPrevTag
Coding->Find paired code	cmd::findMate
Coding->Recolor tags	cmd::recolorTags
Coding->Color tags->Color tags	cmd::colorTags
Coding->Color tags->Text and tags to black	cmd::decolorTags
Project->Code browser	cmd::codeBrowser
Coding->References->Remember reference	cmd::rememberReference
Coding->References->Insert reference link	cmd::insertReferenceLink
Coding->References->Go to reference	cmd::goToReference
Coding->References->Find citations	cmd::cmdCitations
Coding->Hot code sets->Select more	cmd::selectMore
Results->Sort up->Smart sort	cmd::sort*

* Image and PDF files only

Table A6-1

Appendix 7: Working with Audio/Visual Files

TAMS supports for working Quicktime friendly audio and visual files including mp3's, aiff's and other media formats. Basically, TA just acts as a transcription machine. It allows you to attach a single media file to a transcript. Play the media file, insert time codes, control the speed of playback, insert time codes into your transcript and jump to specific time codes in your media file. Time codes are all represented as seconds, which allows for rapid conversion between TA's text and Quicktime's internal time representations.

By default the main controls for the media files are placed above the document code list. The media player is separated by a splitview divider from the code list, so that if you are not working with media, you can basically make the media player go away:

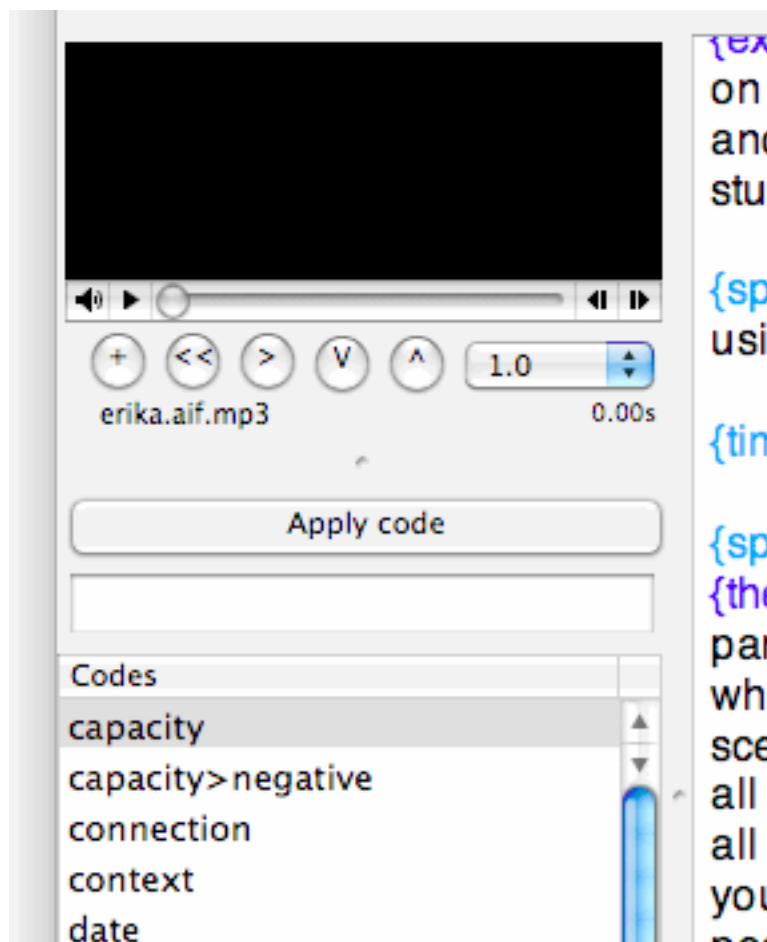


Figure 1. TA media player (default view)

To attach a media file to your transcript use the + button in the media player.

This loads a media file and also inserts the !mediafile tag to the start of the transcript which tells TA where to find the media file next time you open this transcript in TA. Note that TA copies the audio or visual file into the media folder of your tams project. If TA can't find the media file in the project media folder you will be warned.

If you are working with video, you may need a to use the alternative layout which provides a large media player over the transcript pane, rather than over the code list. Put `{!videowindow}` at the start of your file. Save, close, and reopen. The file should open in the alternative layout.

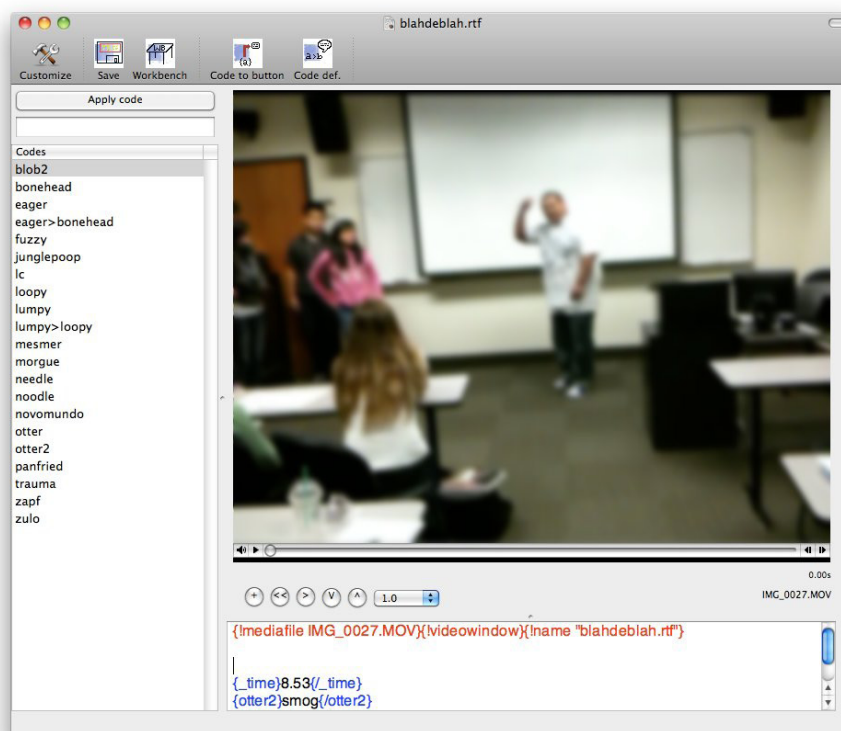


Figure 2. TA media player with the `{!videowindow}` metatag

To play the media file use the `>` button or use the key equivalent apple-K. This is a toggle that becomes the stop button when media are playing.

The media player is basically a transcription machine. **To backspace so you can rehear a portion of your media file use the << key or the key equivalent apple-shift-B.** You can slow down the playback speed by using the small menu to pick a different rate (pick 1.0 to return to normal speed). This can be fine tuned with apple-] and apple-[-.

If you want to insert a time code into your document use the v button. This can either insert the code raw or the time already surrounded by tags depending on user preferences:

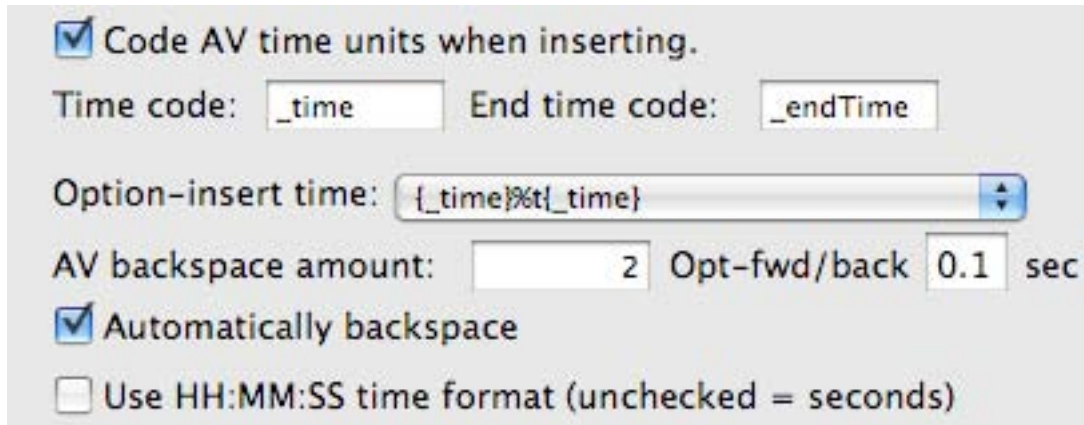


Figure 3. TA User preference options involving audiovisual aspects of TA

The first item determines if and how it is coded. You can also control the amount of back and forward space (again in seconds). And you also link the backspace to the play button so that every time you press play it automatically backspaces.

You can also return to any location in your media file by selecting a time in seconds and using the jump button (the ^ button) or the key equivalent apple-shift-J. The reverse is available as well; move the scrub bar of the media control panel, and pick Coding->Audio/visual->Find nearest time code, and the document will move you to the nearest time.

The Coding->Audio/visual menu provides additional options as well as an easy reference for the key equivalents of the buttons:

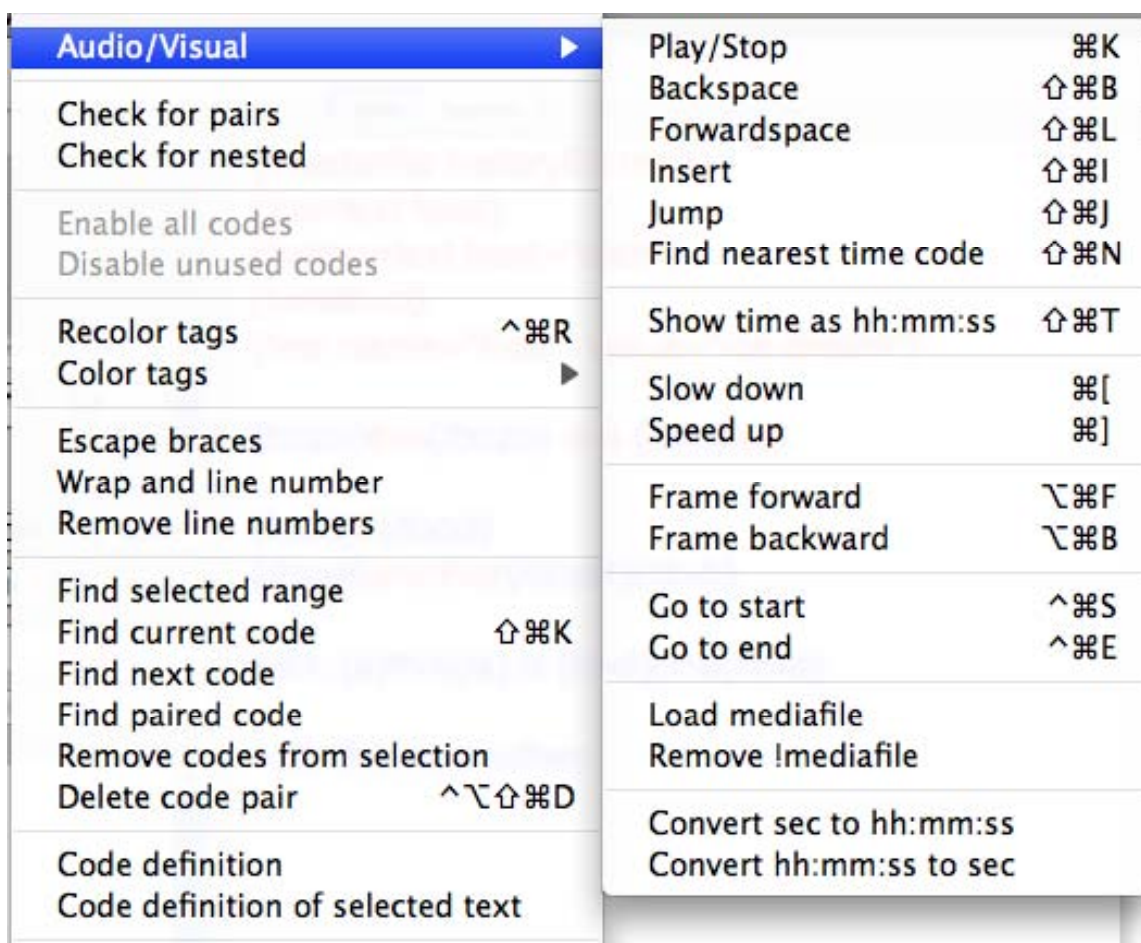


Figure 4. The Audio/Visual Menu

As is clear, this menu also offers additional controls including jumping forward, and fine controls over the speed of play back (1% shifts are possible using the apple-] and apple-[keys). Users can also jump to the start or end of the file, or move through their media one frame at a time.

Note that normally, changing the speed of playback changes the pitch of the sound. This can be overridden through a program preference. The quality of playback is better if the pitch does change.

The show time as hh:mm:ss menu option (also accessible through apple-shift-T for time shows a dialogue that translates the selected text from seconds to hh:mm:ss format. If no text is selected, or the selection is of length 0, then the current position of the media player is translated into hh:mm:ss format and displayed in a dialogue.

Finally one can purge one's transcript of !media metatags. This you might have to do if you move your media file relative to your transcript.

Button summary:

+	Load file	No key equivalent
<<	Backspace	Apple-shift-B
>	Play	Apple-K
x	Stop/Pause	Apple-K
v	Insert time code	Apple-shift-I
^	Jump to selected time	Apple-shift-J
opt-v	User configurable to code time or end time.	

