TAMS Analyzer 1.0 docs

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. However, after many complaints I've been convinced that the existing and out of date documentation needs to be supplemented. This is the first attempt at creating a manual for the current version: 1.0. Here's contact information:

Dr. Matthew Weinstein
Assoc. Professor of Science Ed.
Kent State University
404D White Hall
KSU
Kent, OH 44242
mweinste@kent.edu
http://educ.kent.edu/~mweinste/tams/

rev. 1

I. What is TAMS Analyzer

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

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 or creating them in TA your first job is to select text and indicate what it means.

C. Analyzing

TA's second job is to extract information from a marked up document. Basically TA just compiles a table of text meeting specified criteria. This is called analysis.

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).

II. Getting started with a project

A project consists of a series of files each of which usually represent one thing: an interview, an observation. A project also should consist of a centralized code file. This is a file that contains all the codes used across the files along with their definitions.

A. Case 1: the files already exist

If you have already typed in your interviews, save them as RTF documents and drop them on the TAMS icon; you can also save them as text (make sure they end with .txt or .rtf).

Then create a new file in TA and save it in some appropriate place with a name like "codefile". This will be your centralized code file. Find your workbench window and select the name of the code file and click on the "Code Source" button (second button from the top):

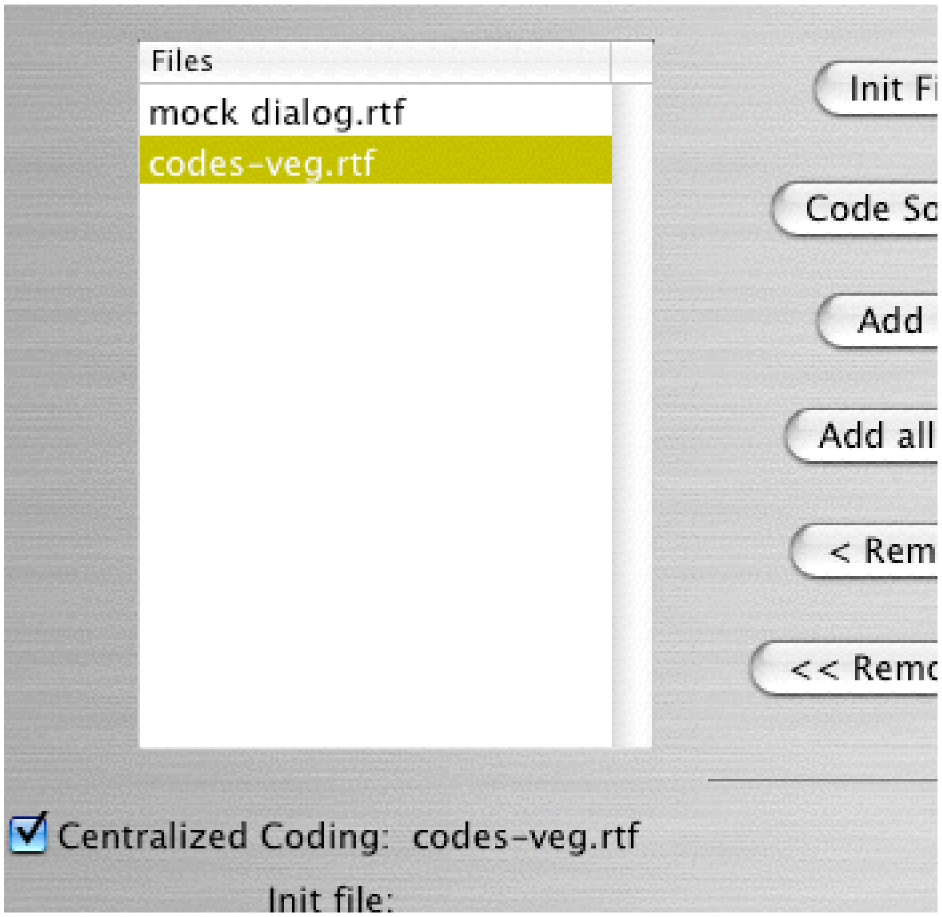


Fig. 1

In the above figure I've just designated a file called codes-veg.rtf as my centralized code file (I did this by selecting it and pressing the second button down, the one with the word "Code" showing).

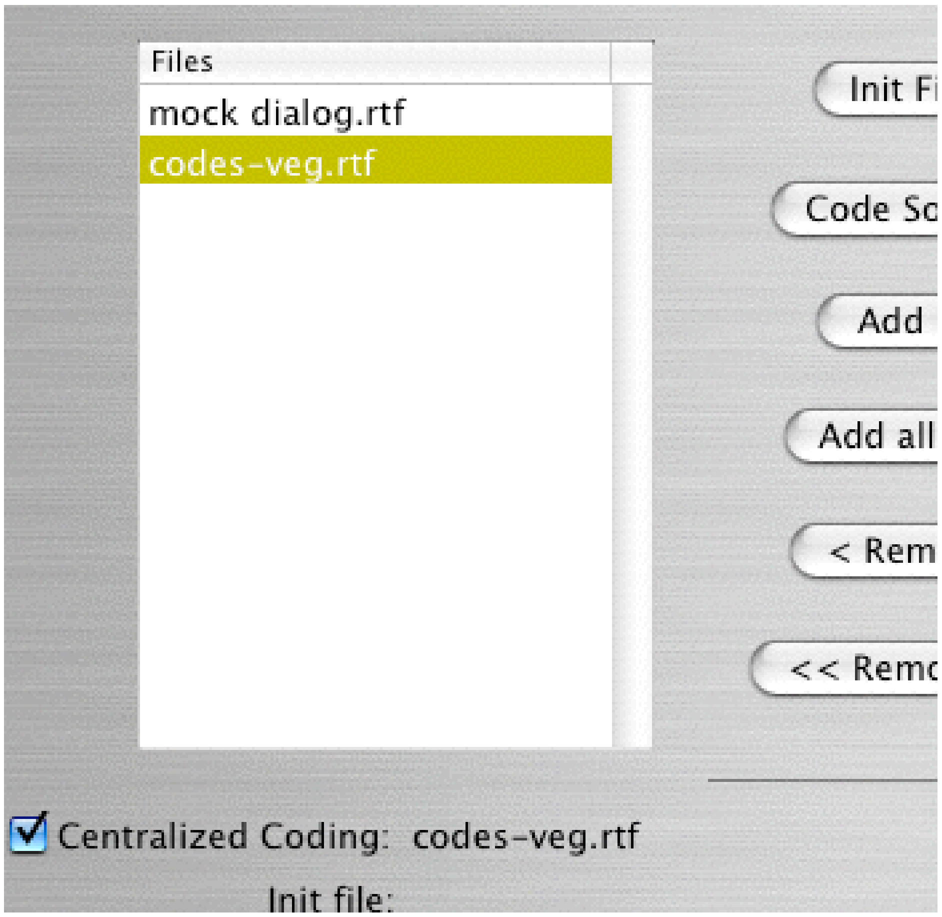
If all has gone well the file name will appear at the bottom and Centralized Coding will be checked.

Now Go to §II.C and save your project.

B. Case 2: creating data files in TA

If you are typing in your files in TA make sure that you save them as RTFDocuments (you can chose this when you do a save, save as, or save to) so that the files keep their formatting. Before typing in your first document create a new empty file and save it with a name like Codes or

Code File; since this will hold the code list and definitions. Then go to the workbench, select this file, and click the "Code Source" button to indicate that this will be your centralized code source.



☑ Centralized Coding: codes-veg.rtf

Fig. 2

If all has gone well, the name of your code file appear at the bottom, and "Centralized Coding" will be checked.

Now you can enter your data; get another new file (from the File menu) and transcribe your interview into the document part of that window. Save often, and save it as an RTFDocument type!!!

NOTE and WARNING: Every time you add a new file to the project remember to save your workbench as well as the file. See §II.C.

C. Saving & restoring projects

TAMS can remember the files involved in your project and open them all and select the central code file.

Once your workbench file list has all the files listed in your project, and your central code file has been designated, you can save it by filling in a name in the field in the lower right corner and pressing the new button.

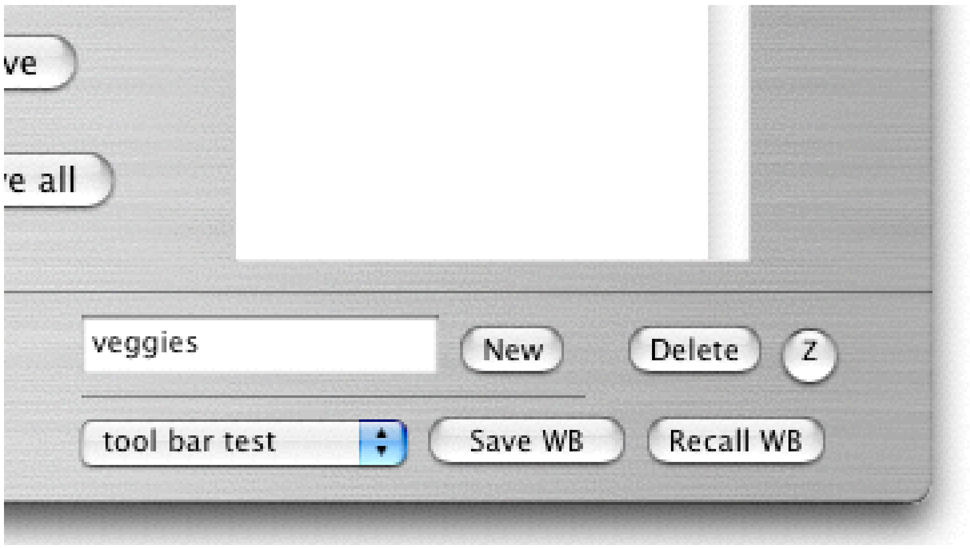


Fig. 3

Here I've called my mock project "veggies".

The next time you open TAMS close the blank window provided, and then select your project from the menu in the lower right corner of the workbench and press "Recall WB". This will load your files and set your central code file.

If you add or subtract any files from the project select the project from the menu at the lower right corner of the workbench (in Fig. 3 it says tool bar test, but it should have the name of YOUR project) and pick "Save WB".

WARNING: That little z (for zap) button will delete all your saved projects, not the files, thank goodness, just the memory of what files were involved in what projects. Delete just removes the project currently selected (tool bar test til I hit the New button and veggies is added and selected)..

III. **Coding**

A. What is a code

A code is a name that identifies the meaning or significance of a passage of text. In TAMS the 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>carrot

    carrot, parsley, and cilantro are subcodes of food. Note that TAMS is case sensitive. Note 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” 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).

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!!!

   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.

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 does not facilitate adding memos (or comments, as I prefer to call them), you just type them in. Now you can see that tags have a lot more than just the code, they also include signatures and commments.

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 and press new.

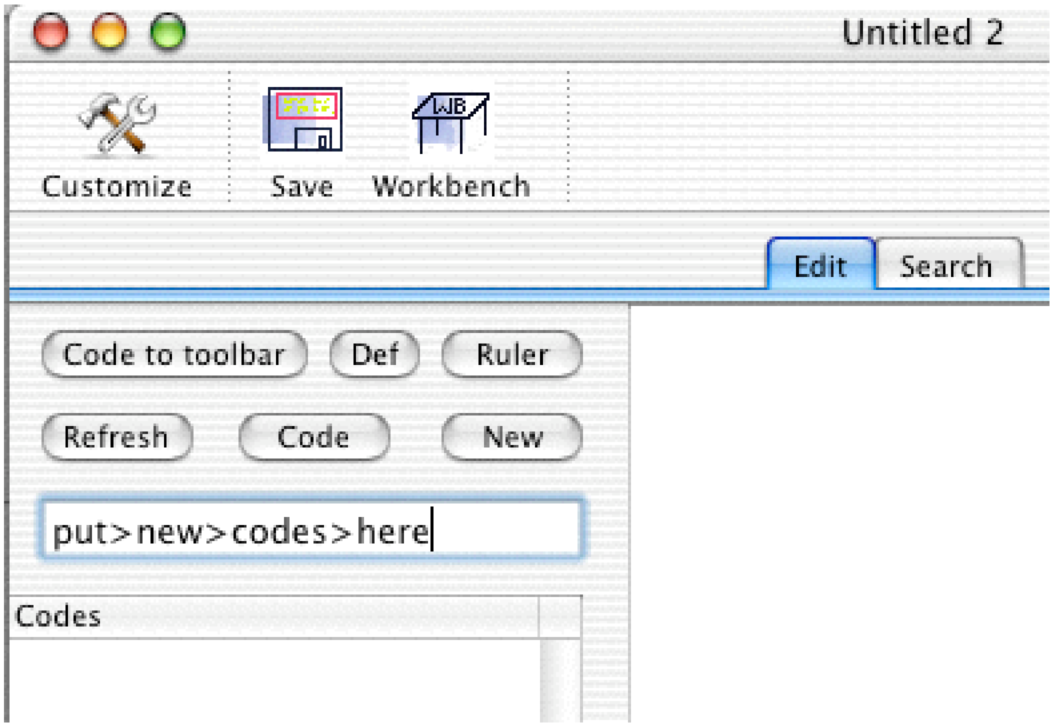


Fig. 4. Entering new codes

Then you'll be prompted for the definition of your new code and when you click ok, and

1. your code and definition will be added to the central code file (which you better save!)

2. added to the codes list under the box you typed in your code name into

3. as noted, applied to the selected text.

WARNINGS:

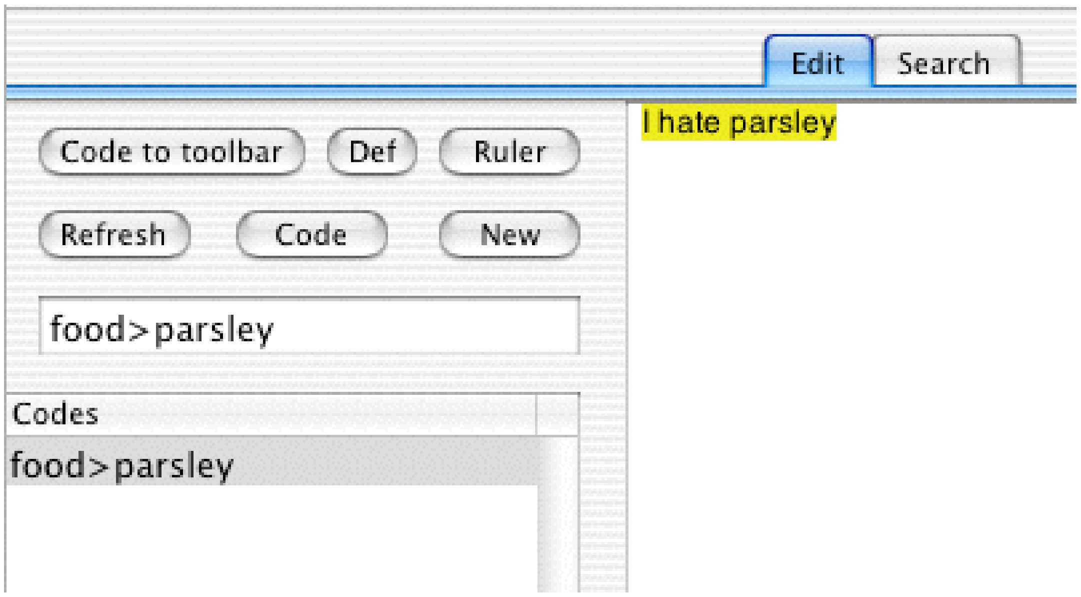
NOW SAVE YOUR FILE AND CODE FILE SINCE BOTH WILL BE CHANGED.

THIS WORKS BEST IF YOU HAVE A CENTRAL CODE FILE IN PLACE. YOU CAN DO THIS WITHOUT, BUT IT IS NOT ADVISABLE.

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.



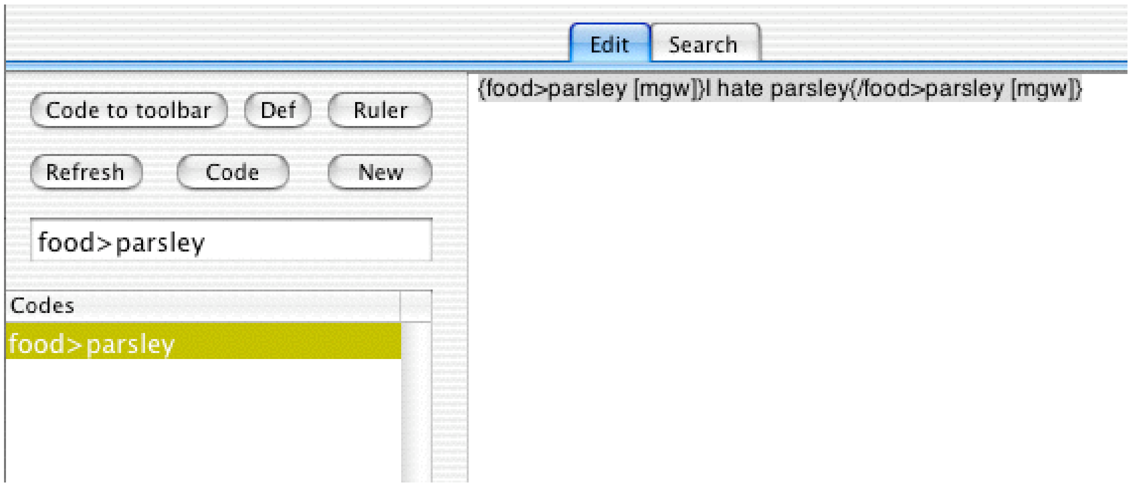
I hate parsley

food>parsley

food>parsley

Fig. 5. coding

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



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

food>parsley

food>parsley

Fig 6. 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 and press "Code to toolbar"

REALLY ADVANCED:

What if you want the button bar to come up each time with certain codes. Put a metatag at the top of your document which 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 need to pick "Build button bar" from the Coding menu.

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.

This weird syntax with the ! is explained in III.E. below.

D. Working with 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. 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. Click in the end tag, pick "Find current code" from the Coding menu. This will select the tag. Now drag it to its new location.

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

The types of codes we've been talking about are data codes. 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 is an interview in this particular file. 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 will fill it with 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 metacode (should fix that at some point) submenu of the Coding menu.

F. Reminding yourself of a code definition.

At some point after 50 or more codes are added, it is useful to see the definiton of a code. No problem, pick the code off of the code list and press the "Def" button. A window explaining the code will pop up! This only works if you set up a central code file!

G. 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. For some reason, TA can't find it's other end.

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. The phrase "trying to code" is not seen by TA. These problems can be found by picking "Check for nested" from the Coding menu.

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

IV. **Documents, sections, and repeat information**

A. Breaking a document into sections

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. 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} tags (or {!endsection} tags, see §IV.C.)

HINT:

You may want to check out Nisus for this (though Nisus at present doesn't work with rtf... coming soon I think) or TexEdit Plus both of which have very fancy search and replace functions which can save a lot of time in marking up documents initially.

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 Metacode submenu of the Coding menu) select it and pick "Turn selection into toolbar button" from the Coding menu. Then it's a simple, single click to stick {!end}s where you need them.

What you need to do is put {!end} tags after each portion of the document: you need to figure out what a portion is, but for interviews it will be after each person's turn speaking, in field notes it will be where you have a time code. etc. It's your decision, but what you're marking are spots that "variables" (who is talking) are going to change. Continue to the next section for how to attach that information to results.

B. Repeat Codes and the Repeat 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 “repeat code”

{!repeat speaker}

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

C. end vs. endsection

Normally, 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 part the previous values are 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 mislead 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}.

V. **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). If you want to search across documents you need to use the workbench.

1. Workbench searches
   To search from the workbench, first put together your search list. This means moving files over by selecting them from the file list clicking the "Add" and "Remove" buttons to move them onto the right hand, search list. Put a code into the "Search" field and hit the button called "search". That's pretty much all there is. Ok, there are a few details still to cover.
2. Document searches
   Really, it works the same way only results will be for this document only.

B. The unlimited search

This simply refers to searching without putting in a code into the search field; just leaving it blank. It 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 an unlimited search.

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

C. Looking for particular codes

Of course, more often you will want to look for particular codes. With centralized coding you just double click the code from the 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 do it through turning on the exact flag (under the search) as well.

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 or good or neighborhood. You could search for “*ood” this basically just searches the code name for a substring.

D. And and Or

I’m no lexical analyzer maker, and TA’s slowness, lack of indexing and lack of proper boolean search features shows that. Sorry. But you can do 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 and there is no grouping with parentheses, so you have to distribute things yourself. Again, so sorry. But at least the feature is there!!!

So 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”

E. Search Flags

In both the document search pane and on the workbench there are four 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. Simple searches
   There are two types of searches that TA can do given a mix of codes. Either it can search for the tags that match what you’ve indicated

and return passages that meet the criteria you've set, or it can 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 tag. If you turn off simple searches, 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 the simple flag checked.

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

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

3. 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...
4. 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. Searching for coders

If you want results 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). 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]”

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.

G. Saving for Excel and databases

When you have results, you may want to save 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 DocumentType from the file format menu. If you accidently forget to switch it to Document type, your file will be named with a .tam at the end. This will confuse Excel (or whatever you are feeding it into); just rename it with a .txt extension.

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 at the top of your documents: {!noquote}. This will turn quotes into \Q for double quotes and \q for single quotes.

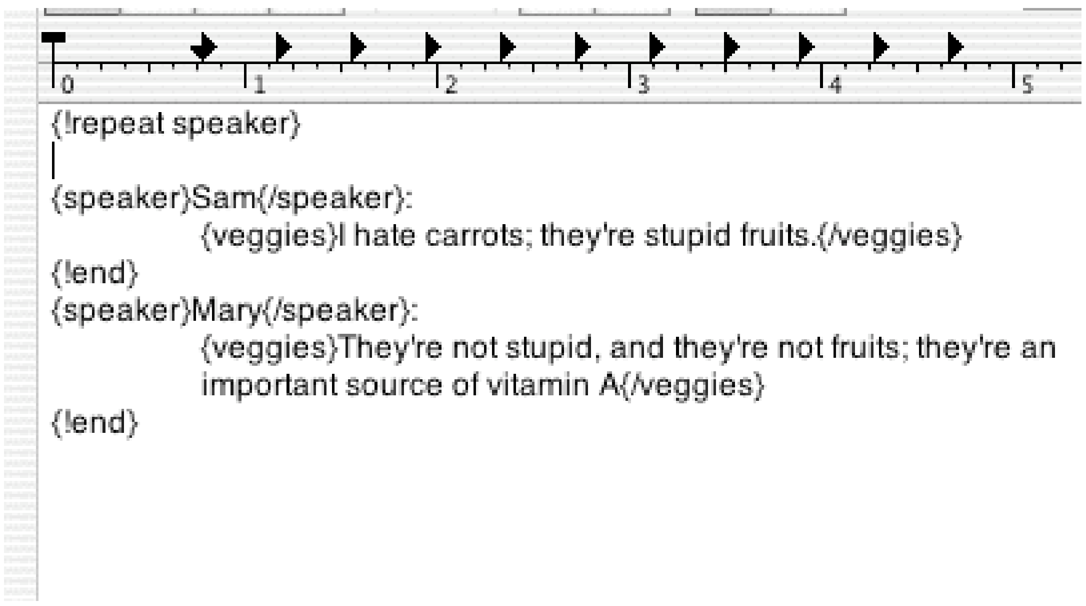
VI. **Interactively reworking your coding**

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.

A. Finding the results in the text

The first way you could do it 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!

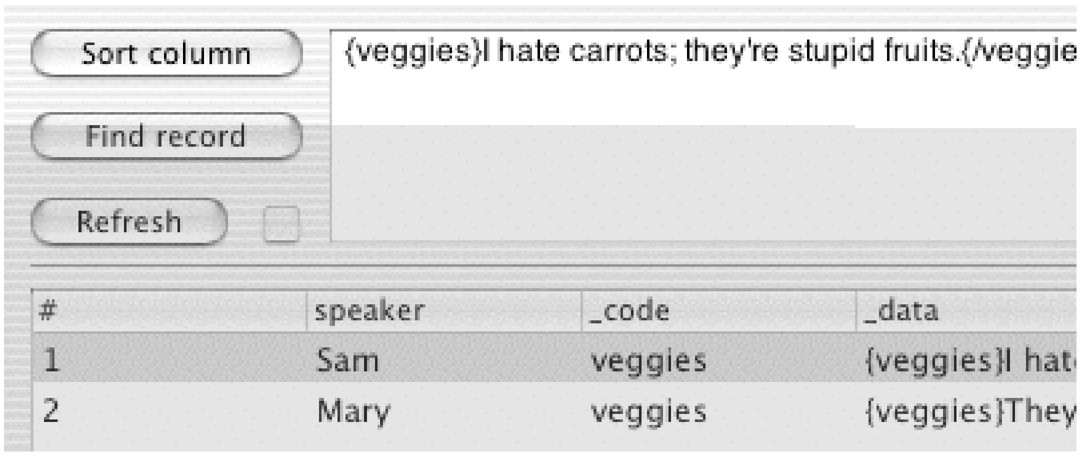
Consider this section of a mock interview



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

Fig. 7. A mock interview

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

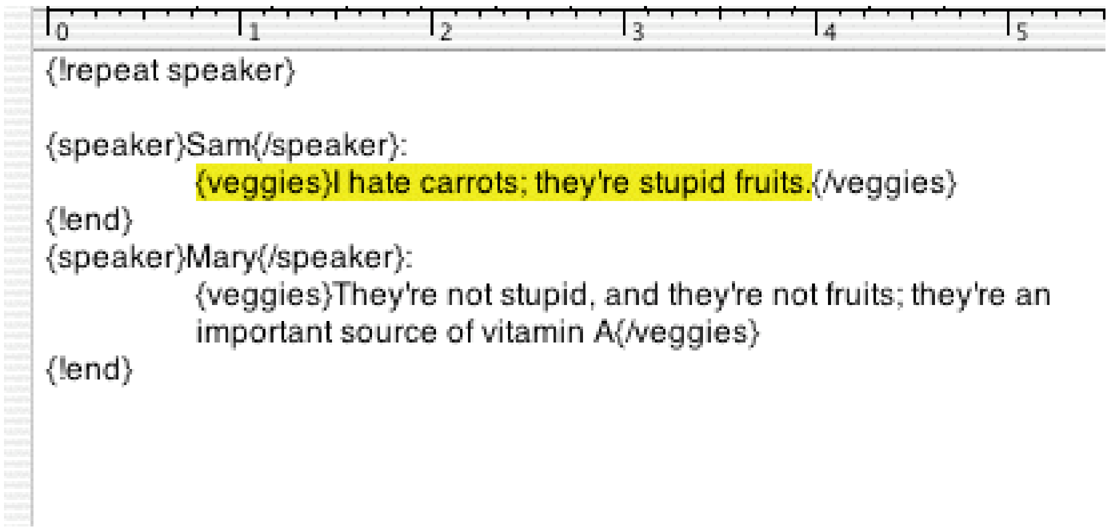


{veggies}I hate carrots; they're stupid fruits.{/veggie

#	speaker	_code	_data
1	Sam	veggies	{veggies}I hat
2	Mary	veggies	{veggies}They

Fig. 8. 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:



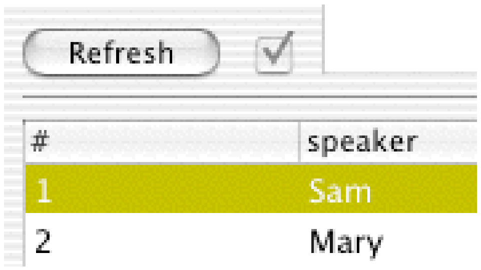
```
{!repeat speaker}

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

Fig. 9. 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.

NOTE: If you want to use this to recode, start with the bottom records first. As you change text the index will get more and more off. You'll know that the original text is changed because a check-mark will appear by the "Refresh" button.:



#	speaker
1	Sam
2	Mary

Fig. 10. 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. This can be a long process, which is why I recommend starting at the bottom and working your way up; you'll be less likely to be off.

B. Marking results

You can also have the program go through and change or add codes to for you. This involves marking the records you want to change and then picking "Add code" or "Recode" from the "Recode" submenu of the Results menu.

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

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) pick Add codes from the Results->Recode submenu. You will get a dialog like this:

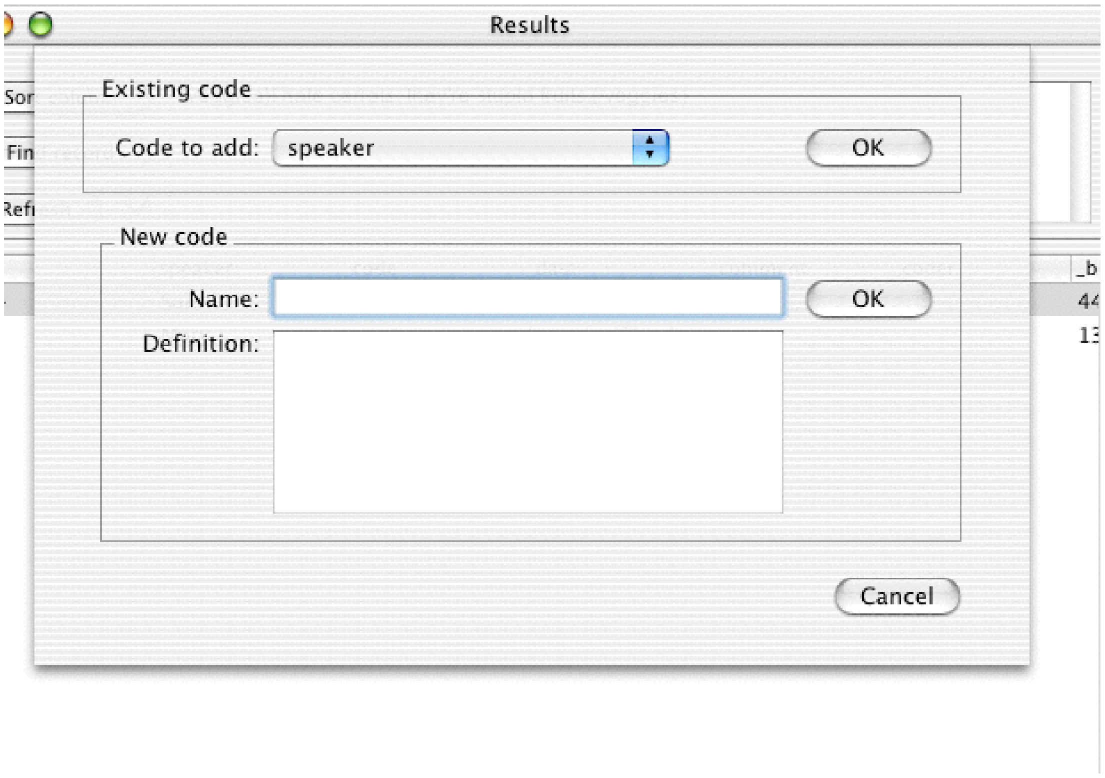


Fig. 11. The Add code dialog box

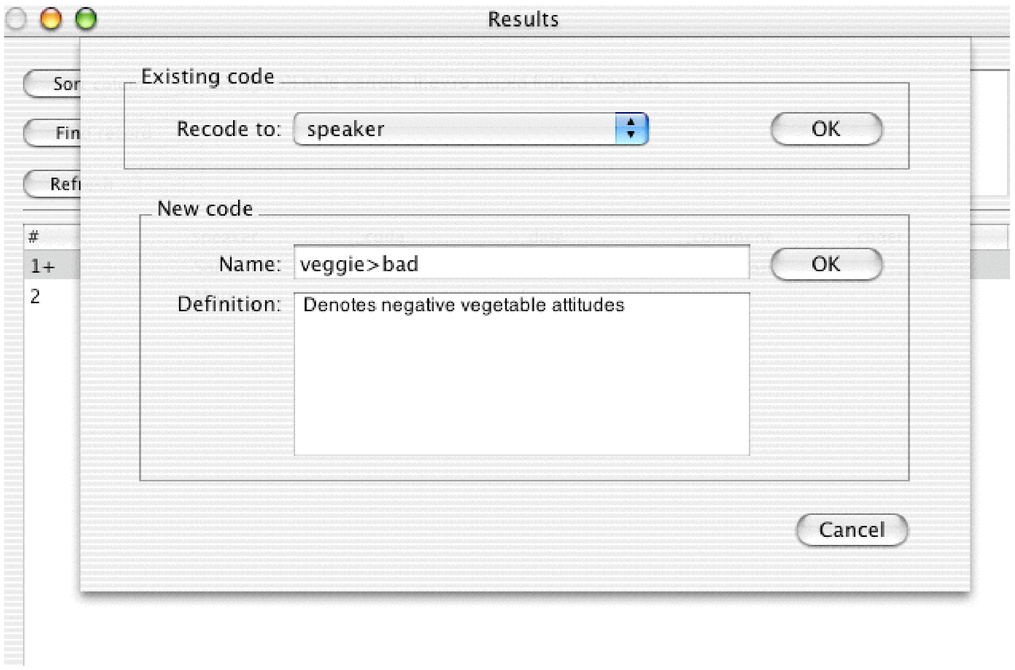
You have two ways to go here: you can pick a code that exists from the menu and press the top OK button, or you can type in a new code into the Name: field and its definition and press the lower OK.

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 than adding codes. This goes through and actually replaces the codes and includes comments that the original codes had. Note that you cannot recode based on any search that involves an "and" or is not simple. So be warned. 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:



veggie>bad

Denotes negative vegetable attitudes

Fig. 12. A filled in Recode dialog box

Here I am recoding the first speaker's comment giving it a negative spin by defining a new code: veggie>bad. I'll click the lower "OK" since I'm filling in the lower information, and voila, the code will be changed. You could 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

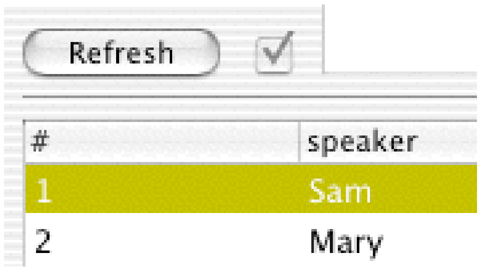
{a}This is {a}some text that {/a} will be recoded {/a}

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 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:



#	speaker
1	Sam
2	Mary

Fig. 13.

Good luck, write if you have problems.

Matthew Weinstein

mweinste@kent.edu

http://educ.kent.edu/~mweinste/