Coding for Beginners with TAMS Analyzer: A tutorial
Matthew Weinstein
mweinste@kent.edu
http://educ.kent.edu/~mweinste/tams/
(updated for TA 2.4)

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

I. **What is coding?**
Coding is simply a way of transforming raw information into data. In qualitative research (i.e., research relying primarily on interviews, observations, document collection) analysis proceeds by sifting through these raw (or if transcribed, semi-raw) pieces of information and deciding what each portion represents. In even modest size projects this can produce what is known as a data burden, i.e., too much information to be comfortably handled. Coding is one way to handle this. Using computers, sticky-notes, or scribbles in margins (and much more complex systems have been devised) relevant passages are "coded," i.e., labeled as to what that passage represents. Single chunks of text should be able to receive multiple codes, and in most modern coding systems they can receive various refinements to those codes, i.e., they can be subcoded (this isn't just an example of X--whatever that may be--but a subtype of X called Y; in TAMS the code would be X>Y; the ">" symbol is used to indicate various levels of sub-coding). An example would help....

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

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

We would like to mark "oink, oink" as the sound a pig makes. Now we may have a lot of codes and collect a lot of different information, including a lot of peripheral information about what children think animals are, so we need to design our code system carefully so we'll be able to keep track of all the information we'll have coming in. For our study, whenever we have an example of a sound we'll mark it with the code "sound" and then a subcode for the type of animal it is. So we

1

would want to mark "oink, oink" with (and I'm using TAMS sytax here; if you use a different system this would be different) with "sound>pig". If we get information about what children know about pigs we will mark it "idea>pig". How we do that will be described later, but it could be done a lot of ways: sticky's hanging off the side with idea>pig on it, for instance. Computer coding usually involves selecting the text and somehow picking the code from a list.

II. What is TAMS?

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

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

III. What is TAMS Analyzer (aka TA)?

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

IV. Starting TA and understanding the parts

When you first run TA you will notice that you have one very complicated window. It is called the workbench or project window. Every research project you have has to have one of these. The file attached to this window (or better stated the file created when you save this window) contains project wide information like the location of your interviews and

other files as well as the codes (which will be explained later) and their definitions. Once you have your project files all showing on the workbench you can save the project by using the Save option on the file menu.

A. The Workbench window
   The workbench or project window ties your files together. The purpose of the workbench is threefold. This is the window you use to define your codes. It is also the window where you can put together multi-file searches. Finally, it is the window that allows you to save and recall your whole project (a topic we wont explore in this introduction; see the documents for a brief discussion of saving projects)

The top part of the workbench is for specifying search criteria for analyzing your data.

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

The right hand panel and buttons in the center are for is for selecting the files to be searched using the criteria entered in the top half of the window.
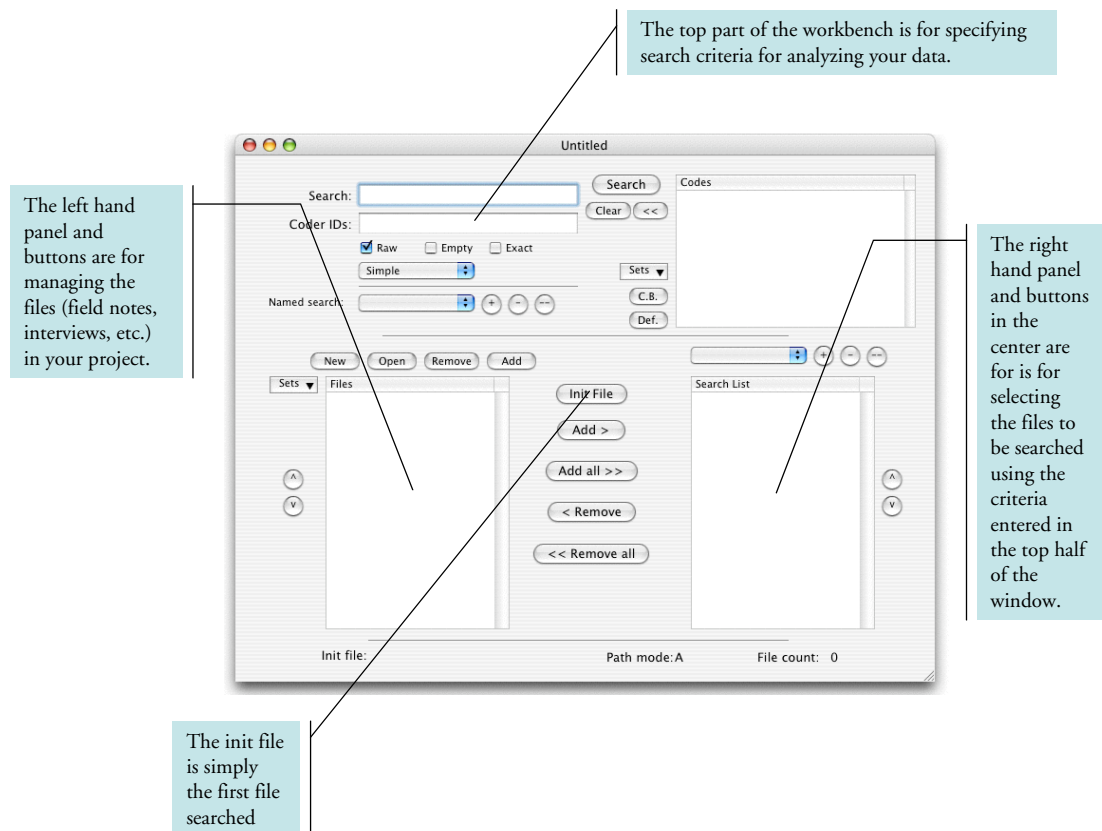
The init file is simply the first file searched

Figure 1. The Workbench

Just so that you know that I'm not trying to hide anything, the init file is simply the first file searched in a multi-file search. It might be an important file for some since one could put special text in that file which controls how your results look.

B. Document Windows
   To make a document window you need to click the new button on
   the project window. The new button is right over the "Files" list box
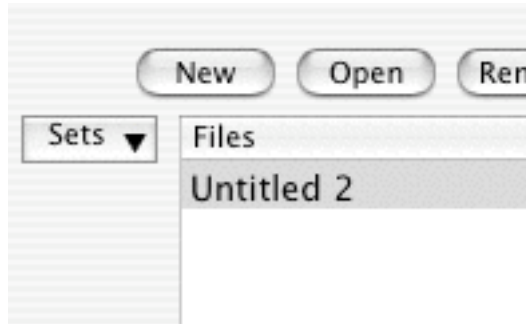   on the right side of the workbench window, under the dividing line:

Figure 2. The New button on the workbench
makes blank documents windows

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

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

The Edit pane is divided (for practical purposes) into two halves: the
left side which has buttons and fields for managing codes (well,
there's one button here which toggles the ruler, but everything else is
about codes on the left side of the window. On the right side is a big
pane which is where you can enter your text. (**Note: this was
written before 2.41 when an additional set of buttons was added
to the left side of the window. While everything in this
documentation holds, you do have to overlook the "media
player": a sort of built-in transcription machine which has
buttons labeled +, <<, >, ^ and v. To read more about the
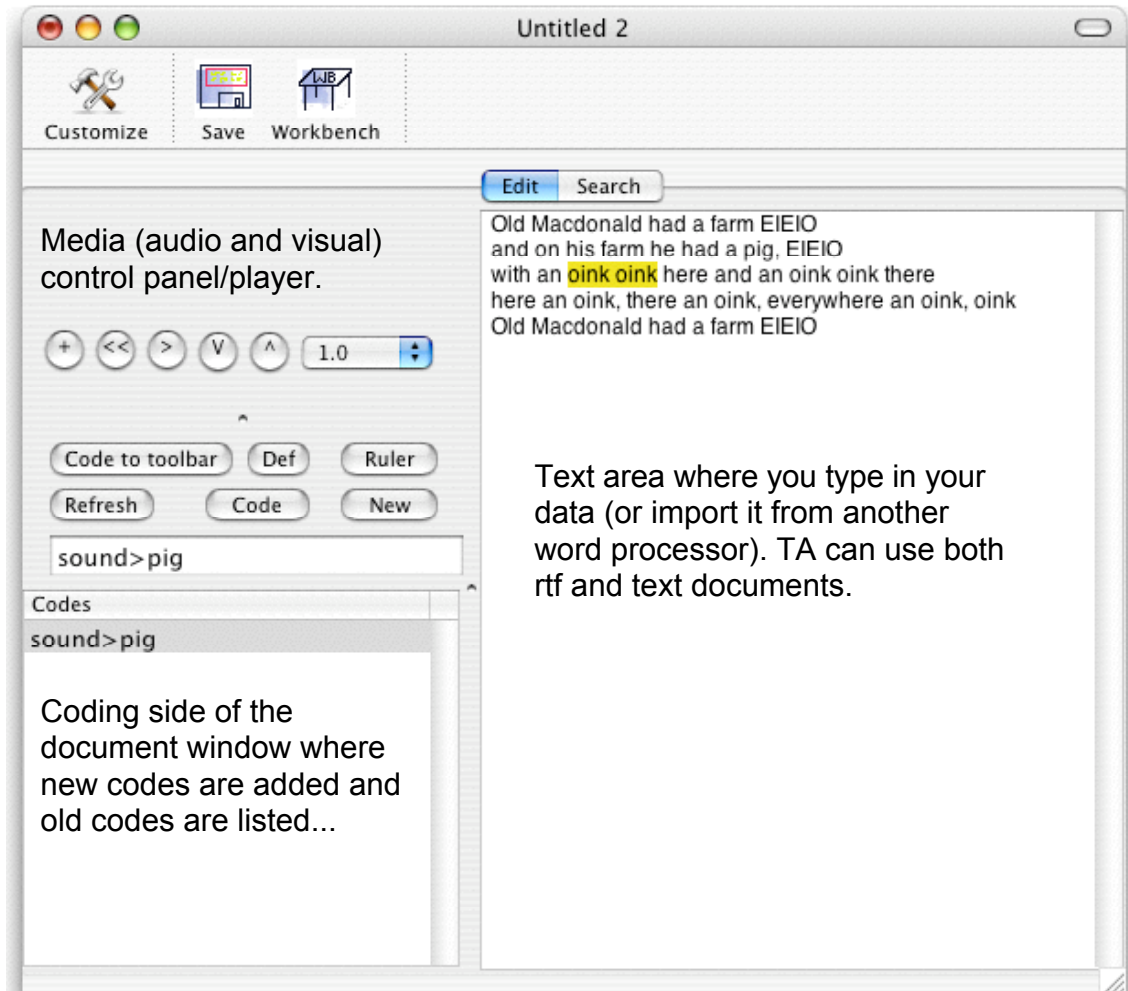media player see the Audio-Visual How To in the How Tos
folder.)**

Figure 3. The document window and its parts (Edit mode)

The buttons are worth a little explanation. The ruler button toggles a ruler in the Text Area (the right hand side), the other five buttons deal with codes.
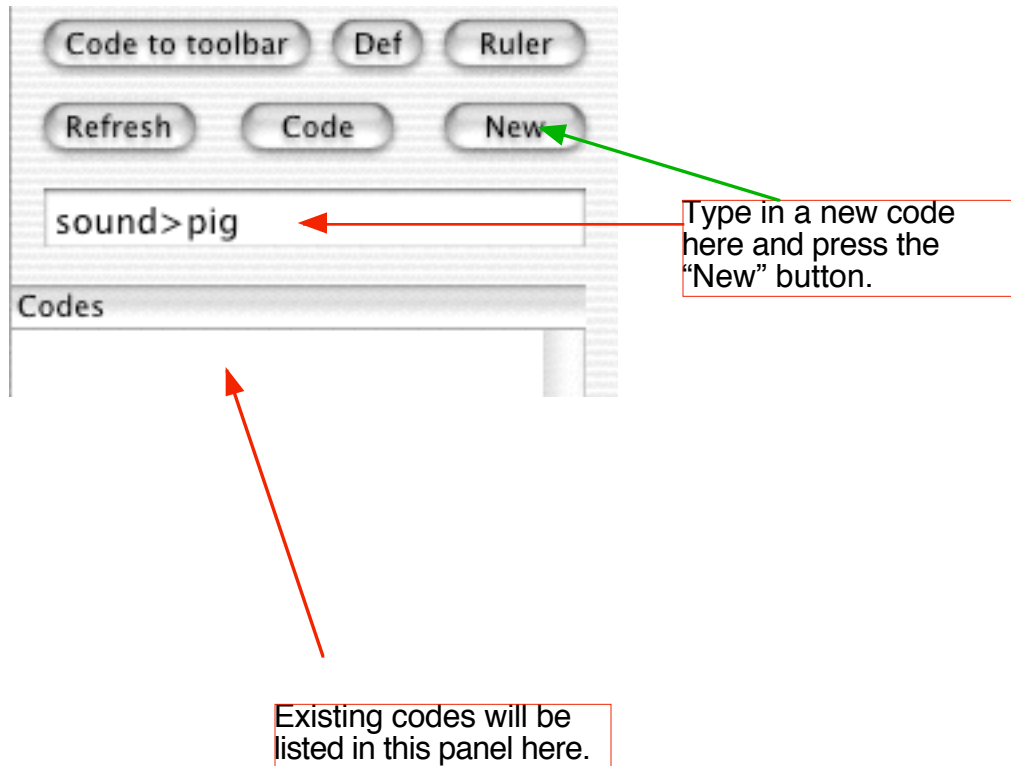
Figure 2. Adding a new code using the button panel

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

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

Refresh: updates the code list (the list under the word codes: here empty) with the latest codes, in case changes have happened (never really needed, a vestigial limb from TA's early years)

Code: Applies the selected code to the selected text. Rarely used. Normally you would just select your text and double click the code name in the code list.

New. Obviously my program isn't telepathic, it has to learn the codes from you to place in the code list. To add a new code you type it in the little window under the button and press "New". (See figure 2)

Once a new code is added it will appear in a list under the word Codes (in the above window, no codes have been added to the project yet). As we will discuss in the next section, the general way

you code text is by selecting the text you want in the right side of the window and double clicking the code from the list on the left side. But this will be handled in the next section: my purpose here is just to point out the anatomy of this window.

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

V. How do I code in TA?

A. Adding codes with the code browser
We've seen one way to add new codes: using the new button. Sometimes, however, a researcher comes to a project with codes already in mind and want to enter them, even before transcribing an interview and applying the codes as you find them. One way to add such "a priori" codes is through the code browser. To get to the code browser go to the workbench window for your project and click the button marked "C.B." Alternatively go to the Project menu and pick code browser. (To save time I'll simply write Project->Code Browser to indicate from the Project window pick code browser, in the future). That should pop up the following window:
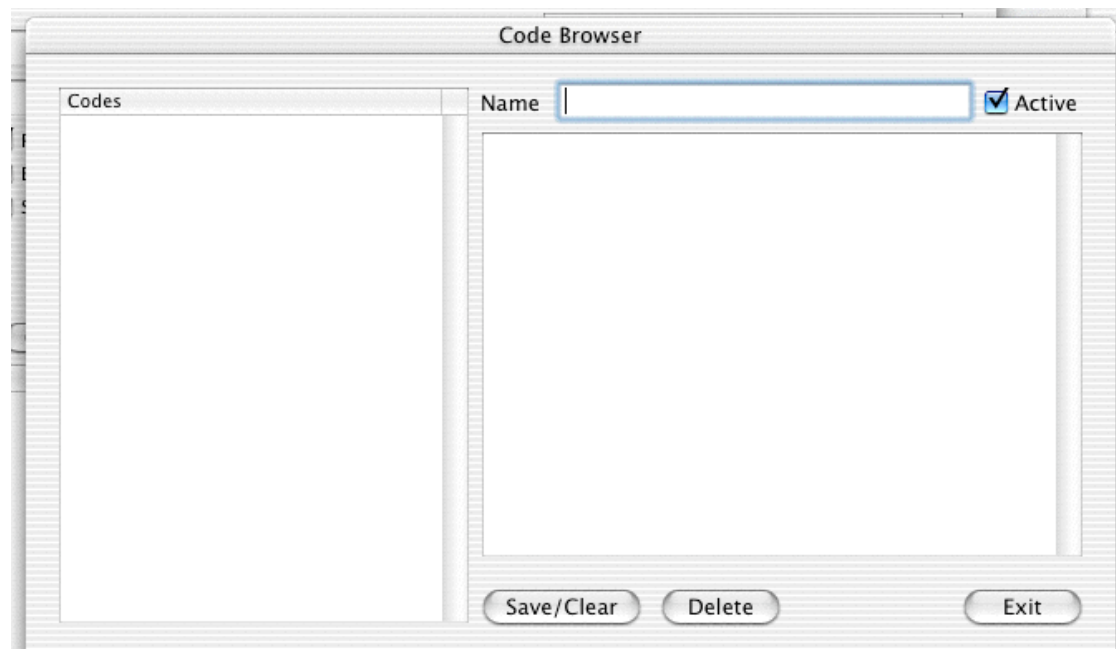

Figure 3. The code browser

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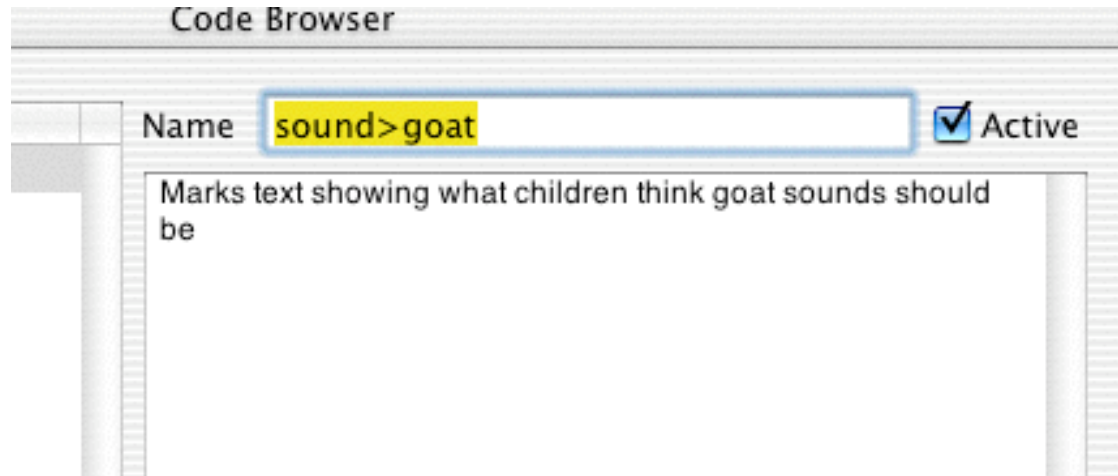

Figure 4. Filling in information in the code browser

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

When you are done typing the definition, click the "Exit" button. The program will save the definition in the project window, but you have to save the project window to save the codes to disk. BUT the workbench wont let you do that until it knows that each of your open document windows have been saved, so if you want to save your definitions and your project file make sure all your other windows are saved first.

B. Adding a new code from the document window
Let's code "Old MacDonald's farm," or at least one verse so that we get the idea. This reiterates what we talked about initially with the "New" button. Make sure you have an open document window. If you don't already, **Click the "New" button on the <u>workbench</u>**.

This is the new button you should be looking for, not the one in the code buttons.

New	Open	Remove	Add

Sets ▼ | Files

Init File

Add >

Add all >>
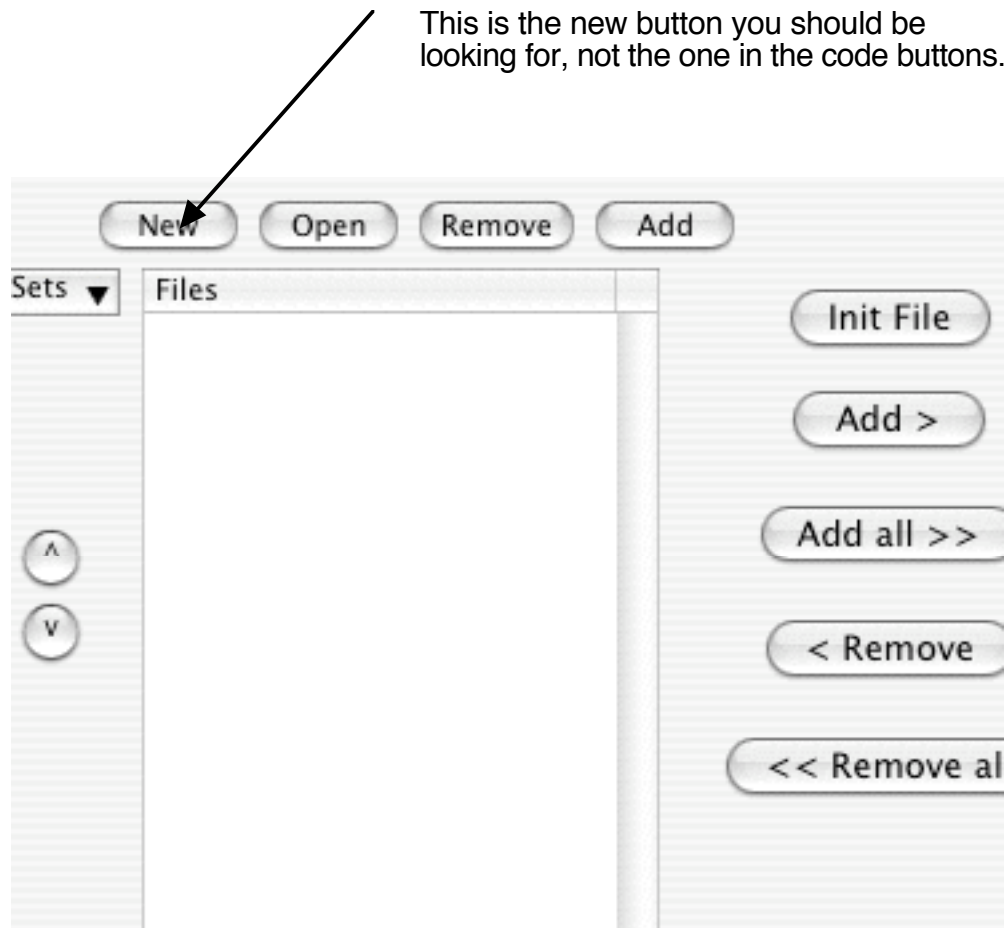
< Remove

<< Remove al

∧

∨

Figure 5. The new button to make a new window

This should provide a blank window.

Type the pig verse into the window:

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

This will go on the right hand side. Let's now assign the code "sound>pig" to the "oink, oink". **Select "oink, oink"** and then on the left hand side type in "sound>pig":
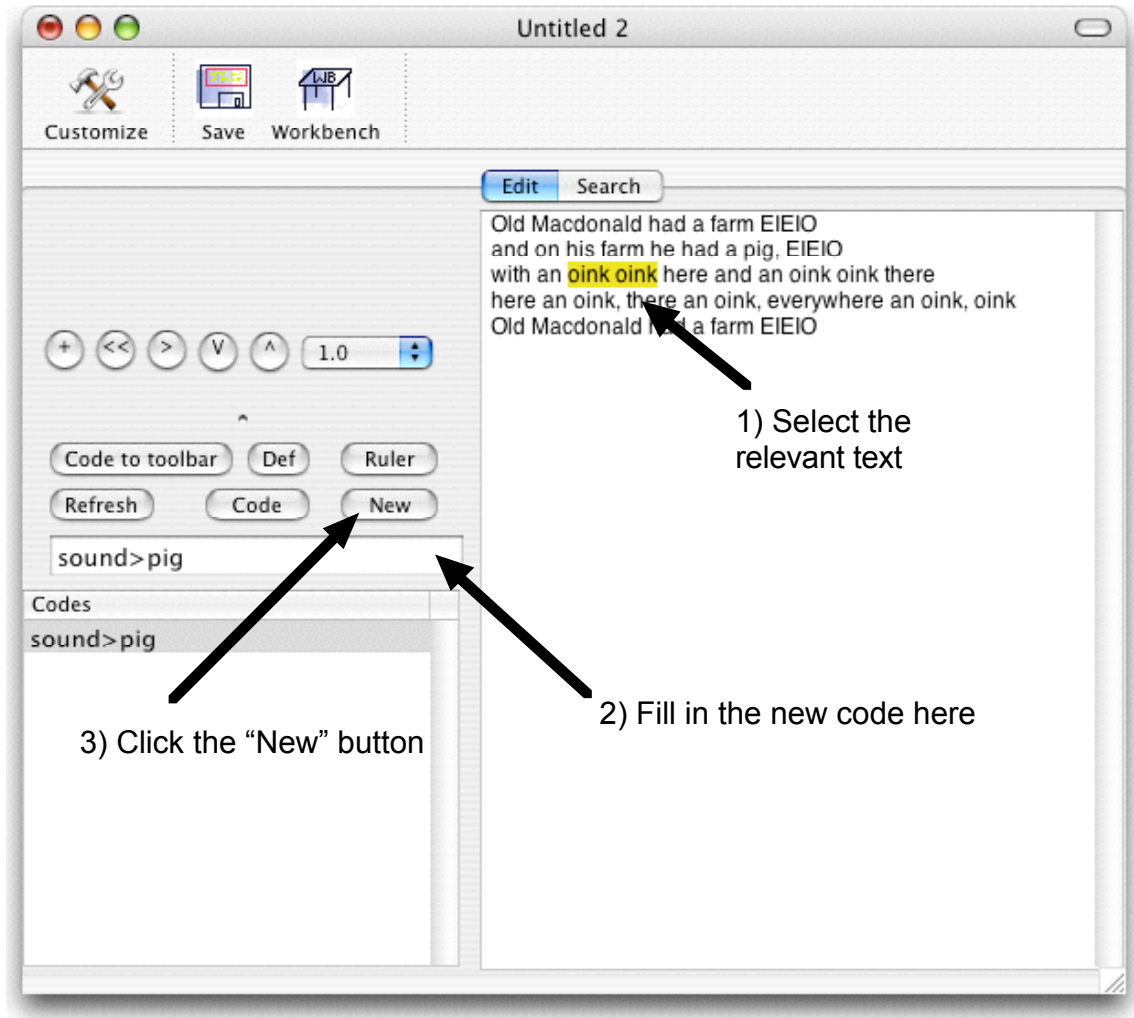
Figure 6. Creating a new code

Then click the New button. This is the one in the button panel. A dialogue will drop down asking for the code's definition.
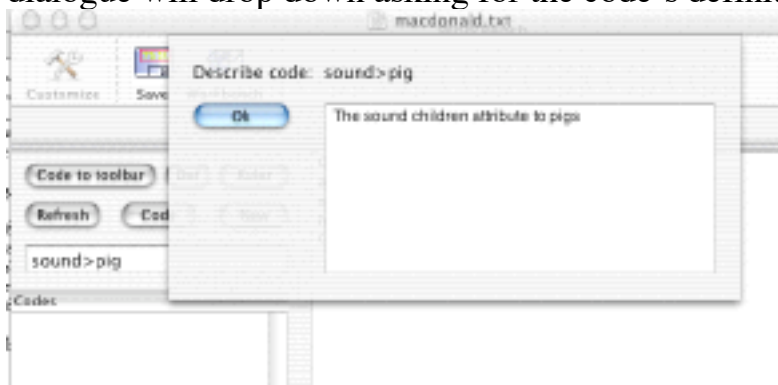

Figure 7. Adding a definition

Clicking the Ok button will have 3 effects:

1. This code and definition will be added to the code list in the
workbench (remember to save it before quiting)
2. The code list on the left of your document window now has a one
code in it (sound>pig) and
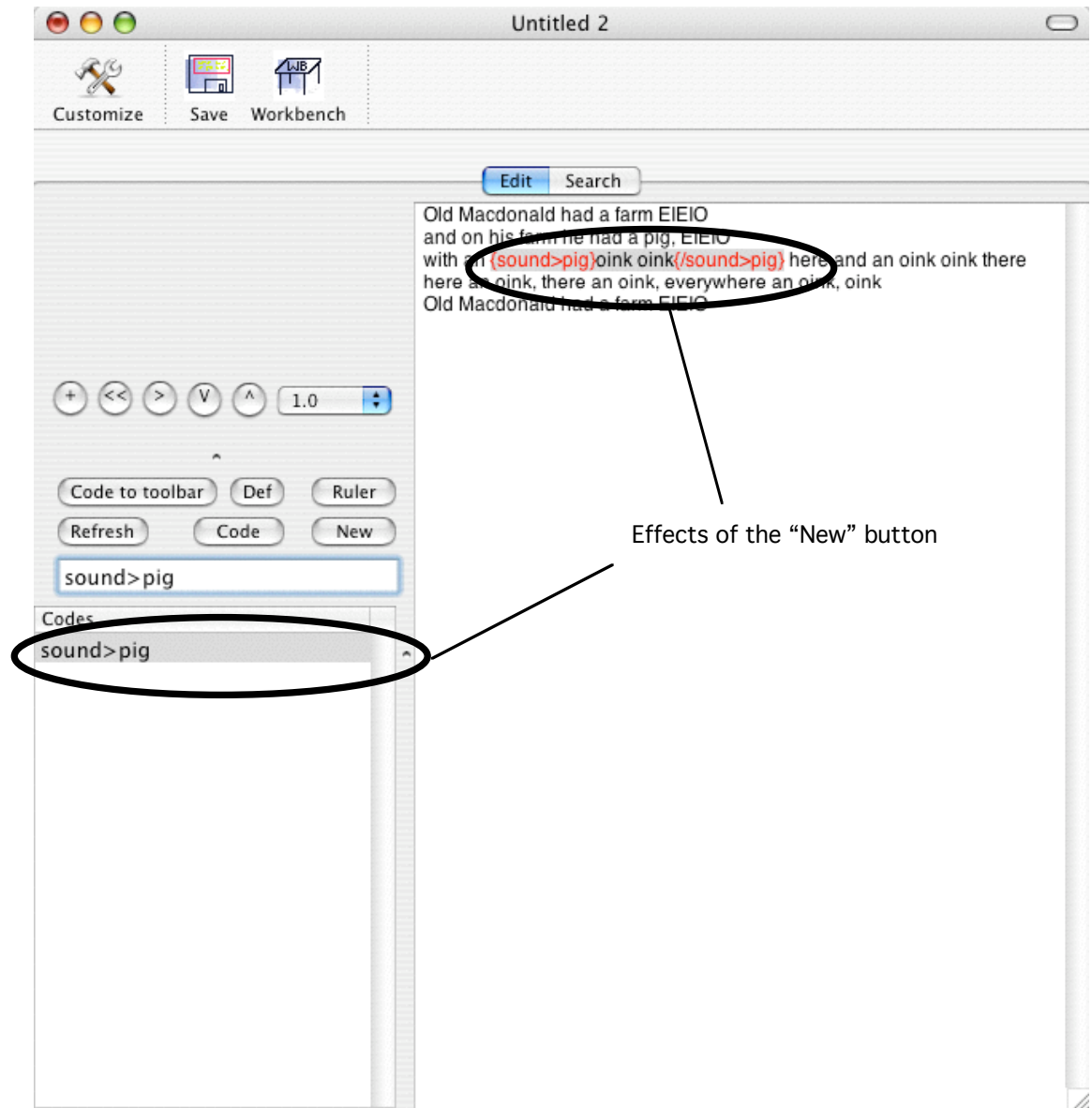3. "oink, oink" has now been coded:

Figure 8. The effects of the New button

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

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

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

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

While this will be dealt with later: codes can be overlapped and nested. Both of the following are perfectly fine. In the following example I've also coded the place that children think animal lives with a setting code, here subcoded to setting>rural:

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

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

D. Recalling the definition of a code
When you have a lot of codes and sub-codes (pig is a subcode of sound in our example) it is often hard to remember the definition you gave a code. No problem. Click one time on the code you're curious about from the code list and click the "Def" button. There's one on the workbench and on your document window. Your definition will pop up:
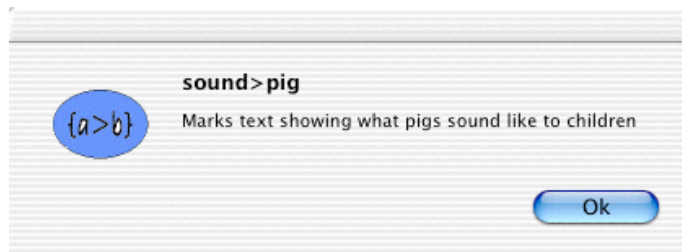
Figure 9. Code definition

E. Working with codes

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

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

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

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

You may also want to find the other end of a code pair. By a code pair I mean the front code (that is the one that looks like {a}) and the back code (the one with the slash: {/a}). Just click in the middle of one end and pick "Find paired code" from the Coding menu.

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

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

13

Finally, all of the codes in a selection of text can be removed by selecting the portion of text you want stripped and picking "Remove codes from selection".

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

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

G. Saving
TA does not automatically document windows save. Save often. Furthermore, your workbench (also called your project file) will quietly be filling with unsaved information as you add new codes. Once your project file is saved it will save automatically as codes are added. Result and document windows you must save on your own.

VI. Concluding comments

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

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

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

- How to assign colors to different codes
- How to group codes together into code sets
- How add comments to a particular coded passage
- How to use results windows to add layers of codes to already coded passages.