Coding for Beginners with TAMS Analyzer: A tutorial

Matthew Weinstein
mweinste@kent.edu
http://educ.kent.edu/~mweinste/tams/

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 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 surrond 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! 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 and it has fairly limited sorting and searching capabilities. For any serious project 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 3/4 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 2 windows to work with. One is called the workbench the other is a “Document” window titled “Untitled.” The latter window is where you will put your data; the workbench is where you juggle the multiple files that make up your research project. Once you have your project files all showing on the workbench you can save the project by

using the menu, field and buttons in the lower right corner of the workbench. The first time you have your project all set up, fill a name into the field and press new. The next time you start TA close the blank document window, pick your project off the menu and click the button called “Recall WB.” See the TA docs for more information.

A. Document Windows

I want to go through the parts of each of these files. In document windows (the “untitled” window) there is a toolbar running across the top which is where you can put text and codes you use a lot. It 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.

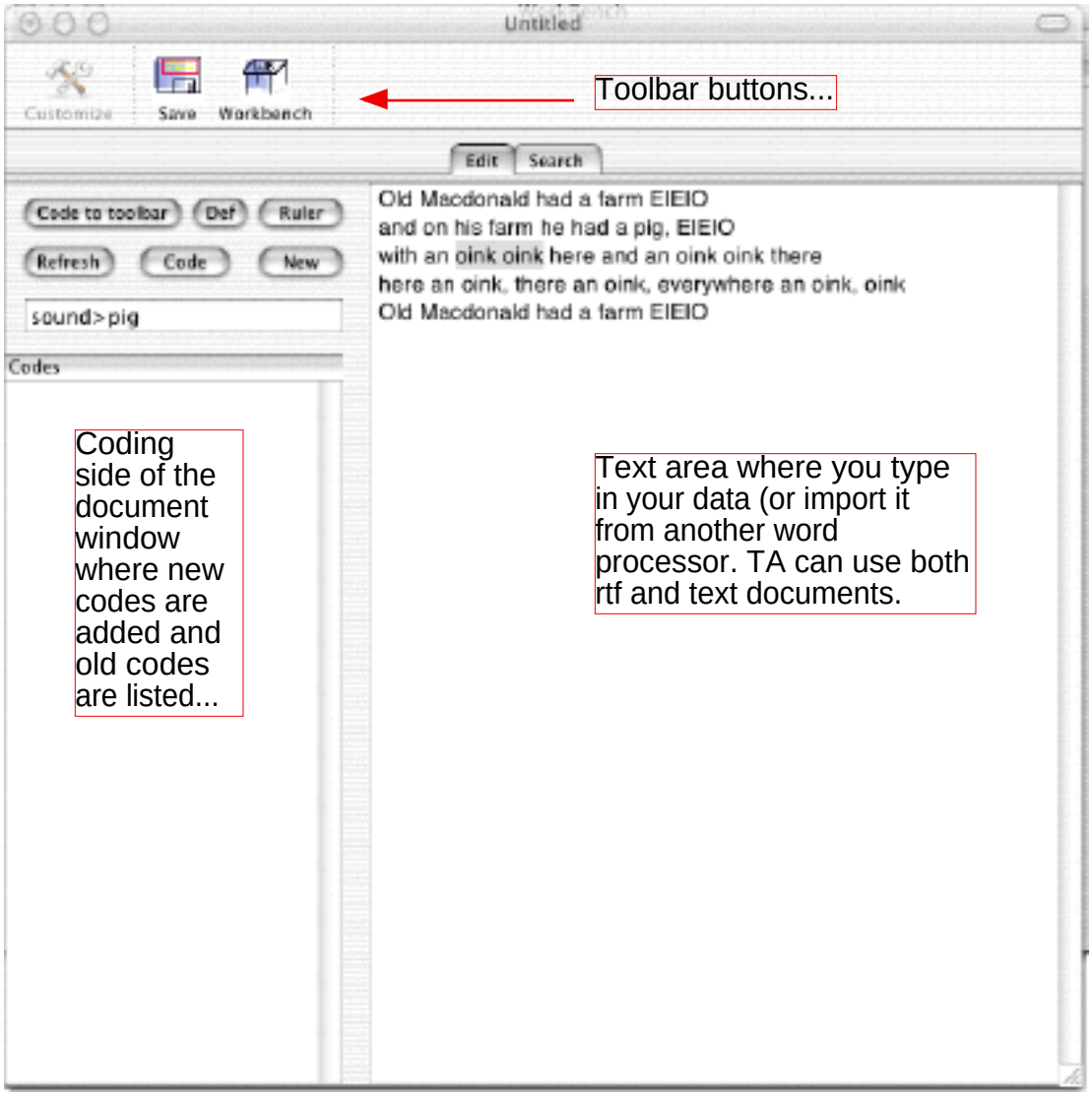

Figure 1. The document window and its parts (Edit mode)

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:

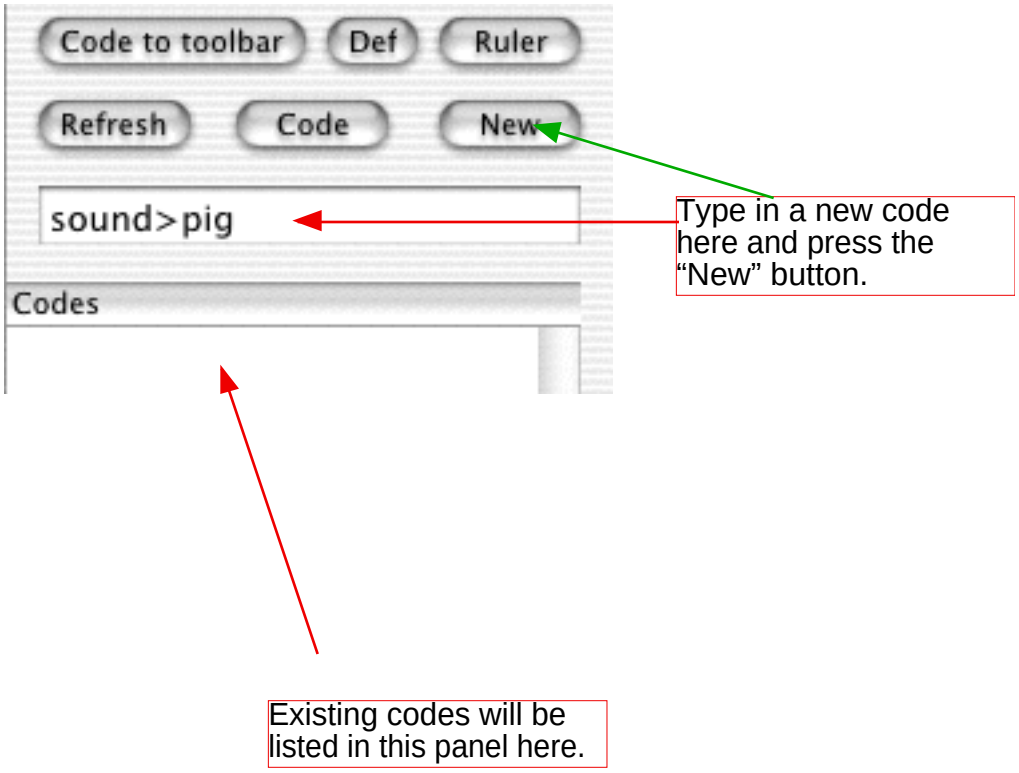

Figure 2. The button panel

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.

Code to toolbar: This button will take whatever code is selected in the code list and make a toolbar button for it...
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

Code: Applies the code to the selected text. Rarely used. Normally you would just double click the code name in the window. This is an alternative in which you click the code name once and then this button.

New. Obviously my program isn't telepathic, it has to learn the codes from you to put them in the code list. To add a new code you type it in the little

window under the button and press “New”.

B. The Workbench window

There is always one workbench window open. The purpose of the workbench is threefold. This is the window you use to indicate your code file, i.e., the file that contains all the codes in your project and their definitions. It is also the window where you can put together multifile 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)

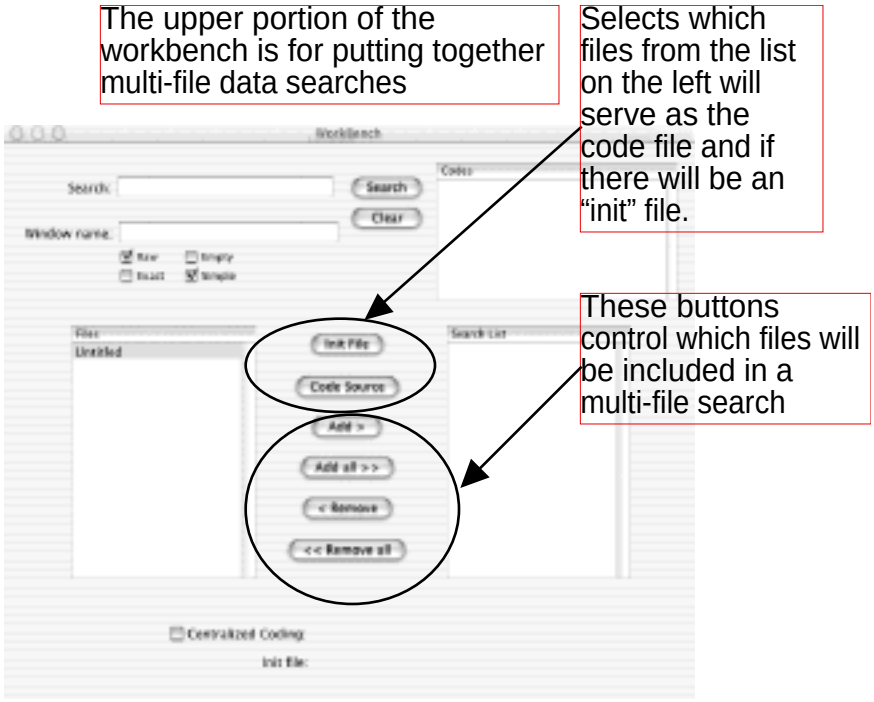

Figure 3. The Workbench

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. To be honest, I rarely use an init file.

The most important thing on this window is the “Code source” button. This will be explained in more detail in the example I explore below.

C. Other windows

There are a couple of other important windows which I’m not going to talk about yet... They are the windows that contain your “results”, by which I

mean the results of searches for relevant data.

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

A. Establishing the code file

The first step to coding in TA is to establish the code file. This is the file that contains all the codes that will be available in your project and their definitions. As noted, each line of the code file has a definition surrounded by code tags:

{sound>goat}Marks text showing what children think goat sounds should be{/sound>goat}
{sound>pig} Marks text showing what pigs sound like to children{/sound>pig}

You could type all of this straight into the code file, but that would be laborious, it's better to let TA do the work for you.

Any file can serve as a code file, but I strongly recommend dedicating one file to it for each project and to not use a file that contains data (an interview or observation notes or memos) as your code file. We'll use the "untitled" document for our code file. Just type code file and hit return and save this file in whatever folder you are keeping your project with some meaningful name like "soundcodes". Notice that TA will put .txt at the end. That's good, don't worry about it.

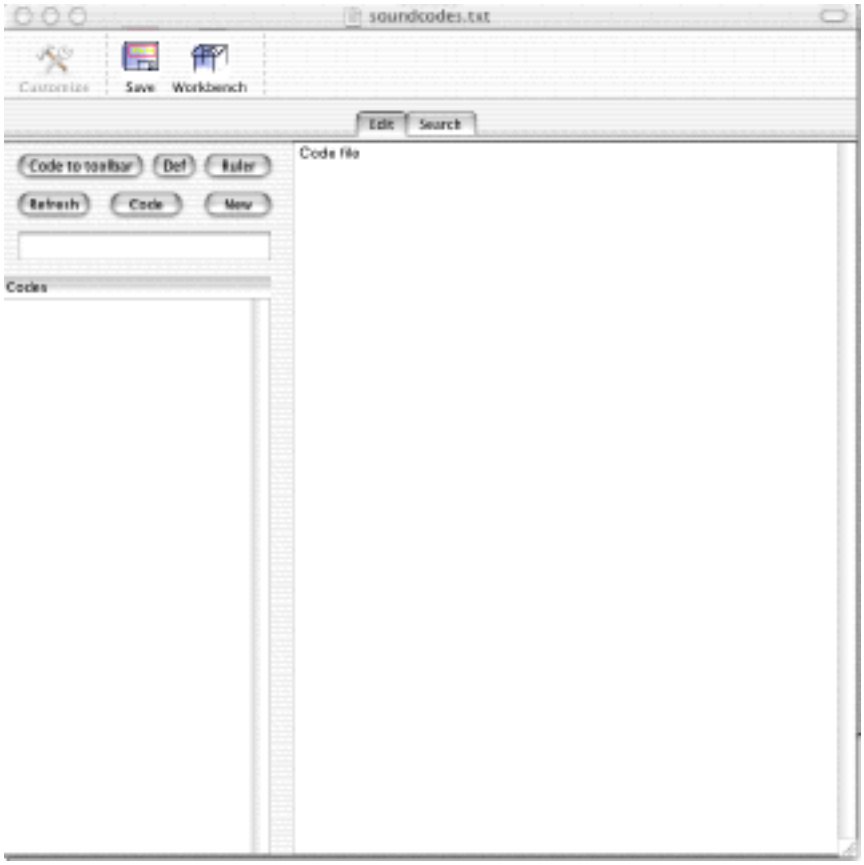

Figure 4. The beginning of creating a code file

There's only one more step. Get the workbench window. On the lower left you will see a list called "Files". Click on "soundcodes.txt" and then press the "Code source" button. You'll know you've succeeded if the name of this file shows up at the bottom as the code file.

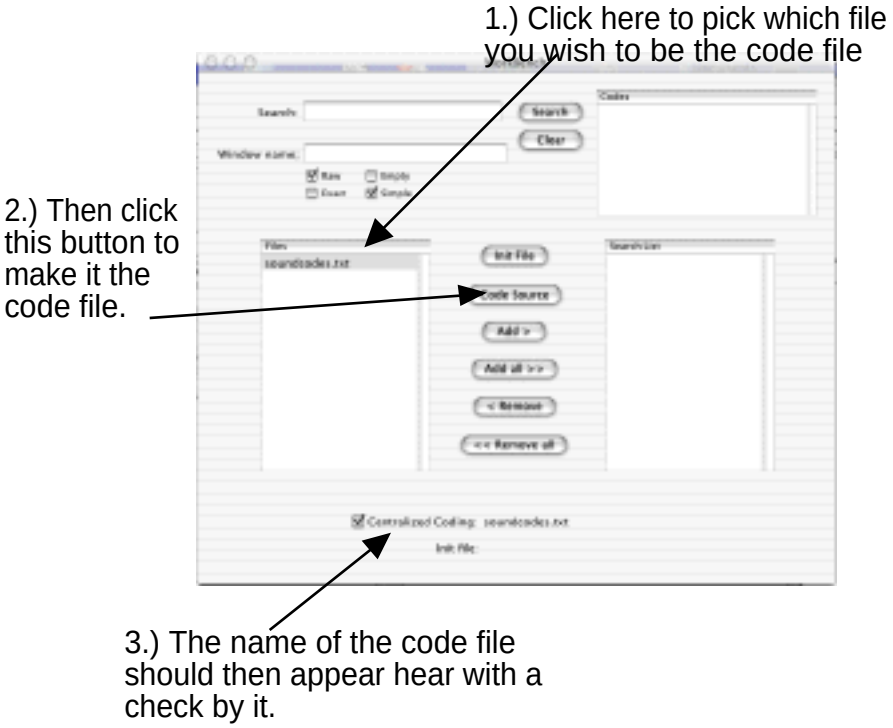

Figure 5. Summary of designating the code file.

Remember, next time you open all the files for this project, one of the first things you should do is come back to the workbench and indicate that soundcodes.txt is the code file. It will not remember between uses of TA. Also, you need to save the code file. See below.

B. Adding a new code

Let's code "Old MacDonald's farm," or at least one verse so that we get the idea. We'll need a new document window. So **Pick "New" from the file menu**. 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":

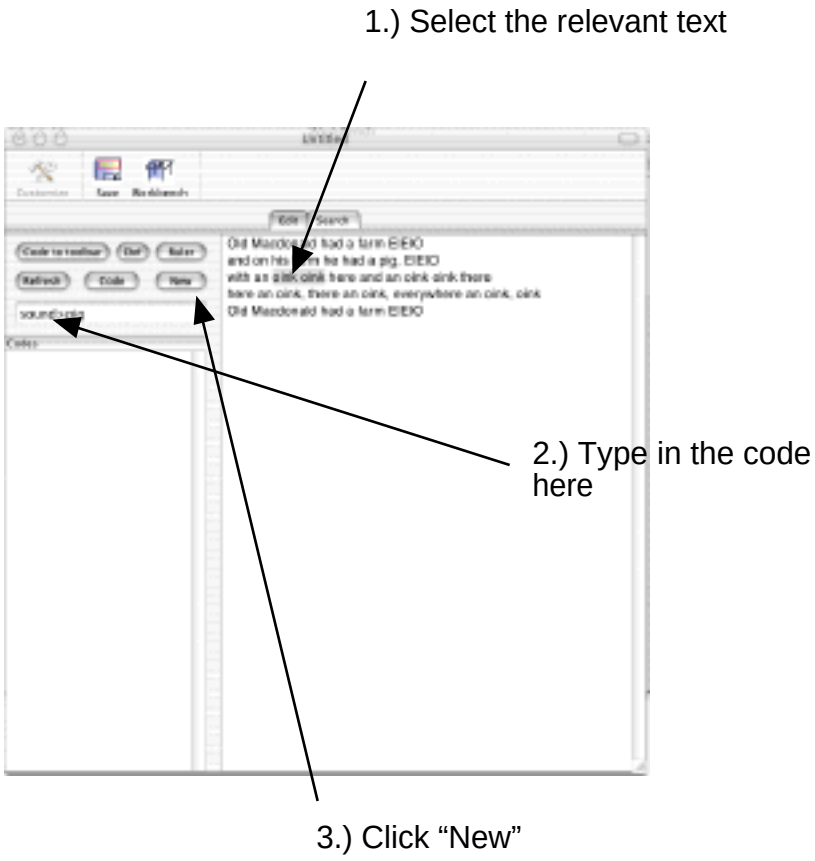

Figure 6. Creating a new code

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

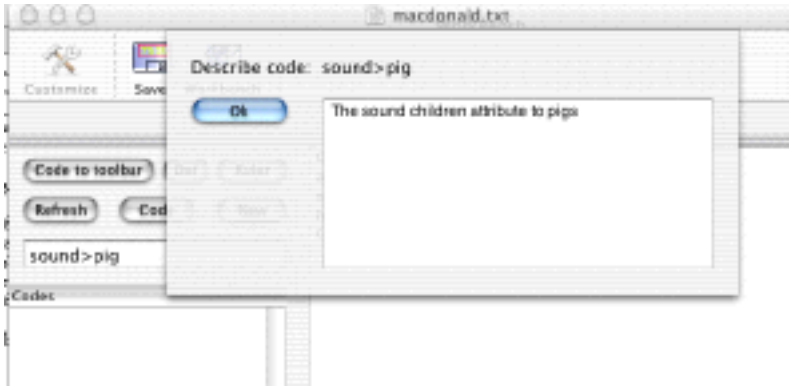

Figure 7. Adding a definition

Clicking the Ok button will have 3 effects:
1. This code and definition will be added to the code file (remember to save it before quiting)
2. The codelist now has a one code in it (sound>pig) and
3. “oink, oink” has now been coded:

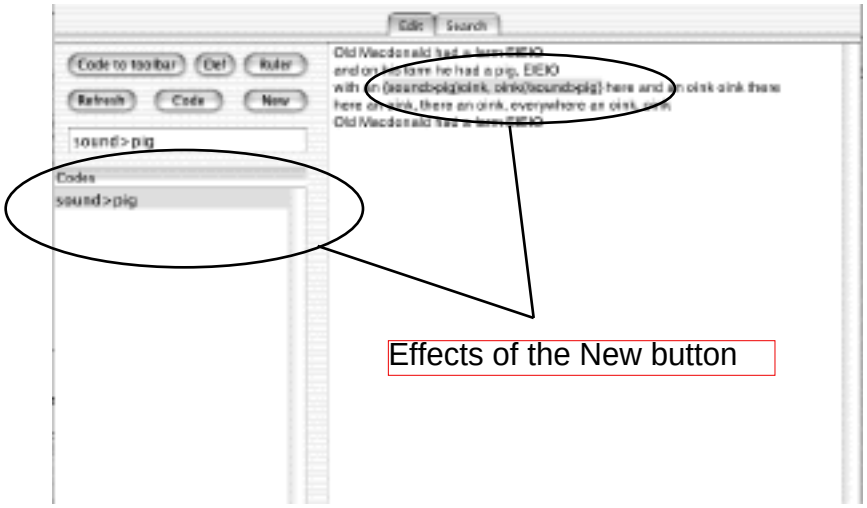

Figure 8. The effects of the New button

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. Your definition will pop up:

Figure 9. Code definition

E. Working with codes
As has been noted codes can be overlapped and nested. Furthermore, related codes can be nested and overlapped. There are no problems for instance with

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

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 TAMS 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. To select a tag just click in the middle of the tag and pick "Find current code" from the menu. That will select the whole code. Now you can use the mouse to drag and drop the code in 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 one end and pick "Find paired code" from

the TAMS 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 TAMS menu.

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 TAMS 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 TAMS menu.

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

G. Saving

TA does not automatically save. Save often. Furthermore, your code file will quitely be filling with unsaved information as you add new codes. Pick your code file every once a while (as in often) and save!!! If you pick exit, always pick review changes, and go through the windows and make sure your work is saved.