Paper 061-31

XSLT-Friendly XML: Generating Structured XML from SAS® Data

Ted Conway, Chicago, IL

ABSTRACT

The XSLT language can be used to transform XML into desired web output, but it sometimes requires more highly structured XML than the default produced by the SAS XML engine. This paper presents a macro that can be used to impose greater structure on XML produced from SAS data, making it easier to use XSLT to obtain desired formatting results. As an added bonus, the example provided produces XML that can be used to help babies to read (really!).

INTRODUCTION

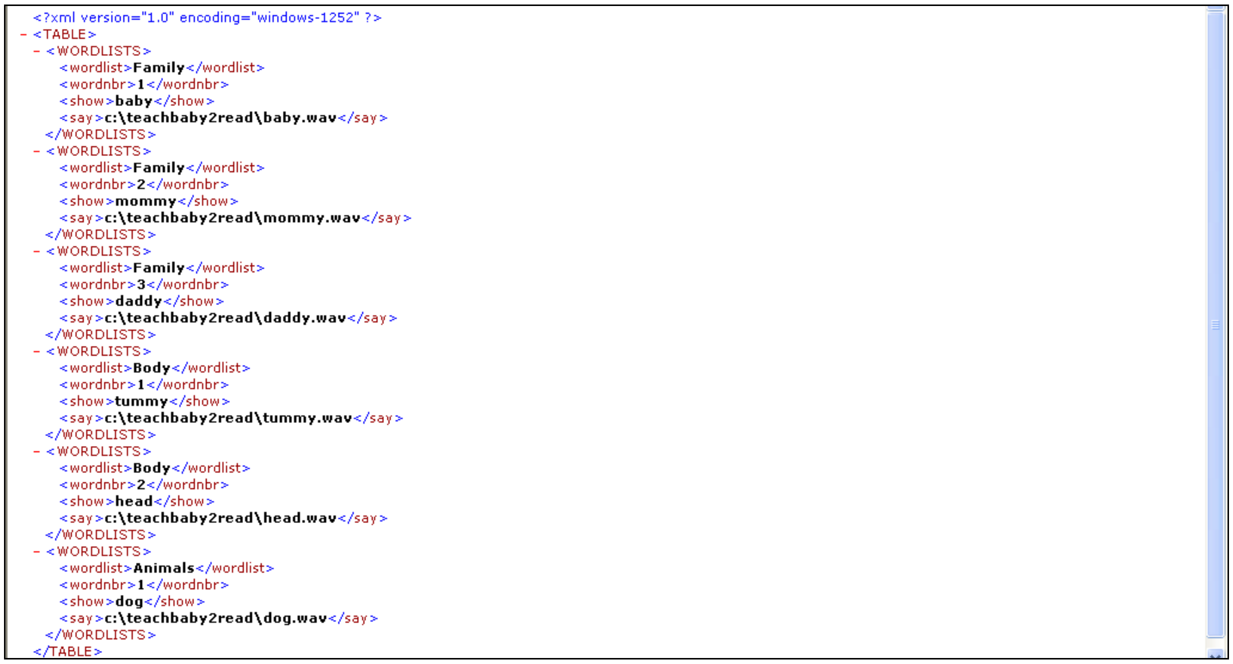
When it comes to generating XML from SAS data, nothing could be easier than using the SAS XML engine. For example, if we have a SAS data set ***wordlists*** like the following (which contains lists of words and associated sound files for a child learning to read!):

```
data wordlists;
infile cards dlm='|';
input wordlist : $255. wordnbr : $255. show : $255. say : $255.;
cards;
Family|1|baby|c:\teachbaby2read\baby.wav
Family|2|mommy|c:\teachbaby2read\mommy.wav
Family|3|daddy|c:\teachbaby2read\daddy.wav
Body|1|tummy|c:\teachbaby2read\tummy.wav
Body|2|head|c:\teachbaby2read\head.wav
Animals|1|dog|c:\teachbaby2read\dog.wav
;
```

We can in fact create an XML file from the SAS data quite easily:

```
libname xmlout xml 'c:\defaultxml.xml';
proc copy in=work out=xmlout;
select wordlists;
```

And here's what the generated XML looks like when viewed with Internet Explorer:



```
  <?xml version="1.0" encoding="windows-1252" ?>
- <TABLE>
  - <WORDLISTS>
      <wordlist>Family</wordlist>
      <wordnbr>1</wordnbr>
      <show>baby</show>
      <say>c:\teachbaby2read\baby.wav</say>
    </WORDLISTS>
  - <WORDLISTS>
      <wordlist>Family</wordlist>
      <wordnbr>2</wordnbr>
      <show>mommy</show>
      <say>c:\teachbaby2read\mommy.wav</say>
    </WORDLISTS>
  - <WORDLISTS>
      <wordlist>Family</wordlist>
      <wordnbr>3</wordnbr>
      <show>daddy</show>
      <say>c:\teachbaby2read\daddy.wav</say>
    </WORDLISTS>
  - <WORDLISTS>
      <wordlist>Body</wordlist>
      <wordnbr>1</wordnbr>
      <show>tummy</show>
      <say>c:\teachbaby2read\tummy.wav</say>
    </WORDLISTS>
  - <WORDLISTS>
      <wordlist>Body</wordlist>
      <wordnbr>2</wordnbr>
      <show>head</show>
      <say>c:\teachbaby2read\head.wav</say>
    </WORDLISTS>
  - <WORDLISTS>
      <wordlist>Animals</wordlist>
      <wordnbr>1</wordnbr>
      <show>dog</show>
      <say>c:\teachbaby2read\dog.wav</say>
    </WORDLISTS>
  </TABLE>
```

At first glance this doesn't look too bad, but if you look closer you'll see a fundamental problem – just like the SAS data set, the generated XML essentially has a flat structure.

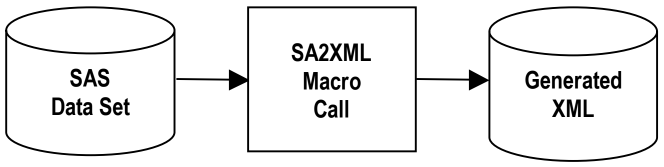
This will make life difficult when we go to display the XML later with XSLT (Extensible Stylesheet Language Transformations), since we're probably going to want to display things in a hierarchical fashion.

To make things more XSLT-friendly, we need to impose greater structure on the XML produced from the SAS data to denote this hierarchy.

How, you ask?

With a handy-dandy macro, of course!

SAS2XML MACRO – OVERVIEW



When ***SAS*** is run, the ***SAS2XML Macro*** creates an ***External XML File*** from the ***SAS Data***, imposing the hierarchical structure specified by the ***XML macro parameter***.

SAS2XML MACRO – SAMPLE USAGE

Before getting into the details of the **SAS2XML** macro, let's first take a look at how one might use it to specify hierarchical relationships for the generated XML, and what exactly that generated XML might look like. Using our earlier ***wordlists*** example, the macro call would look like the following:

```
%sas2xml (data=wordlists, xml=<wordlists<wordlist<wordnbr<show><say>>>>, file=c:\testxml.xml);
```

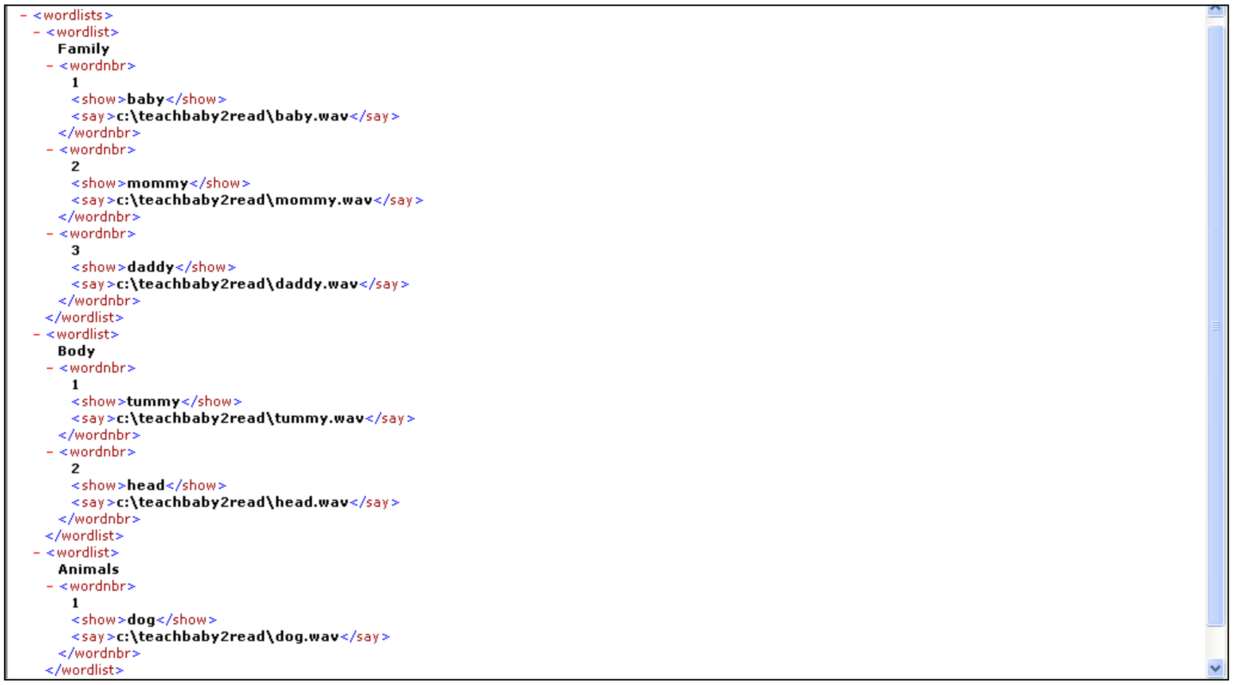
The ***data*** parameter specifies the name of the SAS data set for which XML is to be generated.

The ***xml*** parameter not only specifies the keywords and values that are to be included in the generated XML, it also is used to denote the structure of the generated output hierarchy.

Much in the same fashion that curly brackets are used by the C language to define **STRUCT**ures, less than (<) and greater than (>) signs are specified in the **xml** parameter to respectively indicate when a hierarchy level increases or decreases.

The ***file*** parameter is used to specify the file name of the generated XML.

Our generated XML now looks like the following when viewed with Internet Explorer:



```
- <wordlists>
  - <wordlist>
      Family
    - <wordnbr>
        1
        <show>baby</show>
        <say>c:\teachbaby2read\baby.wav</say>
      </wordnbr>
    - <wordnbr>
        2
        <show>mommy</show>
        <say>c:\teachbaby2read\mommy.wav</say>
      </wordnbr>
    - <wordnbr>
        3
        <show>daddy</show>
        <say>c:\teachbaby2read\daddy.wav</say>
      </wordnbr>
    </wordlist>
  - <wordlist>
      Body
    - <wordnbr>
        1
        <show>tummy</show>
        <say>c:\teachbaby2read\tummy.wav</say>
      </wordnbr>
    - <wordnbr>
        2
        <show>head</show>
        <say>c:\teachbaby2read\head.wav</say>
      </wordnbr>
    </wordlist>
  - <wordlist>
      Animals
    - <wordnbr>
        1
        <show>dog</show>
        <say>c:\teachbaby2read\dog.wav</say>
      </wordnbr>
    </wordlist>
```

The structure of the individual word lists and words is much clearer now – and much more XSLT-friendly!

SAS2XML MACRO – CODE

The actual **SAS2XML** macro is a little messy, but at least it's not too long!

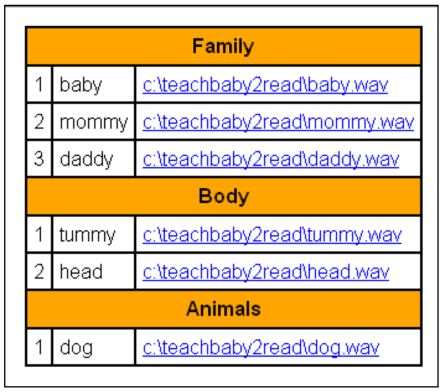
Looking character-by-character through the value specified for the ***xml*** parameter, **SAS2XML** essentially uses the less than (<) and greater than (>) signs it encounters to determine when it should generate opening and closing XML tags.

```
%macro sas2xml(data=, xml=, file=);
data _null_;
set &data end=eof;
file "&file";
if _n_=1 then put "<%scan(&xml,1,'<>')>";
%let n=0; %let byvars=;
%do i=1 %to %length(&xml);
 %if "%substr(&xml,&i,1)"="<" %then %do;
  %let n=%eval(&n+1);
  %let kw_&n=%scan(%substr(&xml,%eval(&i+1)),1,'<>');
  %put n=&n <&&kw_&n>;
  %if &n^=1 %then
   If first.&&kw_&n then put "<&&kw_&n>" &&kw_&n +(-1);;
  %if &n^=1 %then %let byvars=&byvars %str(&&kw_&n);
  %end;
 %if "%substr(&xml,&i,1)"=">" %then %do;
  %put </&&kw_&n>;
  %if &n^=1 %then
   if last.&&kw_&n then put "</&&kw_&n>";;
   %let n=%eval(&n-1);
   %end;
%end;
if eof then put "</%scan(&xml,1,'<>')>";
by &byvars notsorted;
%mend;
```

SAS2XML OUTPUT – SAMPLE XSLT TRANSFORMATION

The following sample XSLT formats the generated XML to produce the IE output shown to the below right – words are grouped in a hierarchical fashion by wordlist name and include a clickable link that plays the associated sound file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method='html' version='1.0' encoding='UTF-8' indent='yes'/>
<xsl:template match="/">
<html>
<body>
<table border="1" bordercolor="BLACK" cellspacing="0" cellpadding="5" align="left">
<xsl:for-each select="wordlists/wordlist">
    <tr bgcolor="ORANGE">
        <th align="center" colspan="3"><xsl:value-of select="text()"/></th>
    </tr>
    <xsl:for-each select="wordnbr">
        <tr>
            <td><xsl:value-of select="text()"/></td>
            <td><xsl:value-of select="show"/></td>
            <td><a href="{say}"><xsl:value-of select="say"/></a></td>
        </tr>
    </xsl:for-each>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```



Family		
1	baby	c:\teachbaby2read\baby.wav
2	mommy	c:\teachbaby2read\mommy.wav
3	daddy	c:\teachbaby2read\daddy.wav
Body		
1	tummy	c:\teachbaby2read\tummy.wav
2	head	c:\teachbaby2read\head.wav
Animals		
1	dog	c:\teachbaby2read\dog.wav

CONCLUSION
When you need to impose greater structure on XML produced from SAS data, give **SAS2XML** a try!

REFERENCES
Be sure to check out the Base SAS Community at *support.sas.com* for XML-related resources.

XML transformations are discussed at *http://support.sas.com/rnd/base/topics/odsxml/0011a.htm#.xml.nowwhat*.

CONTACT INFORMATION
Ted Conway resides in Chicago, Illinois. He can be reached at tedconway@aol.com.

TRADEMARKS

OFF-TOPIC BONUS
So what was the deal with that teaser in the Abstract?

To see how the XML produced in our example could actually be used to help teach babies to read, check out the following two pages!

OFF-TOPIC BONUS
USING XML TO TEACH YOUR BABY TO READ

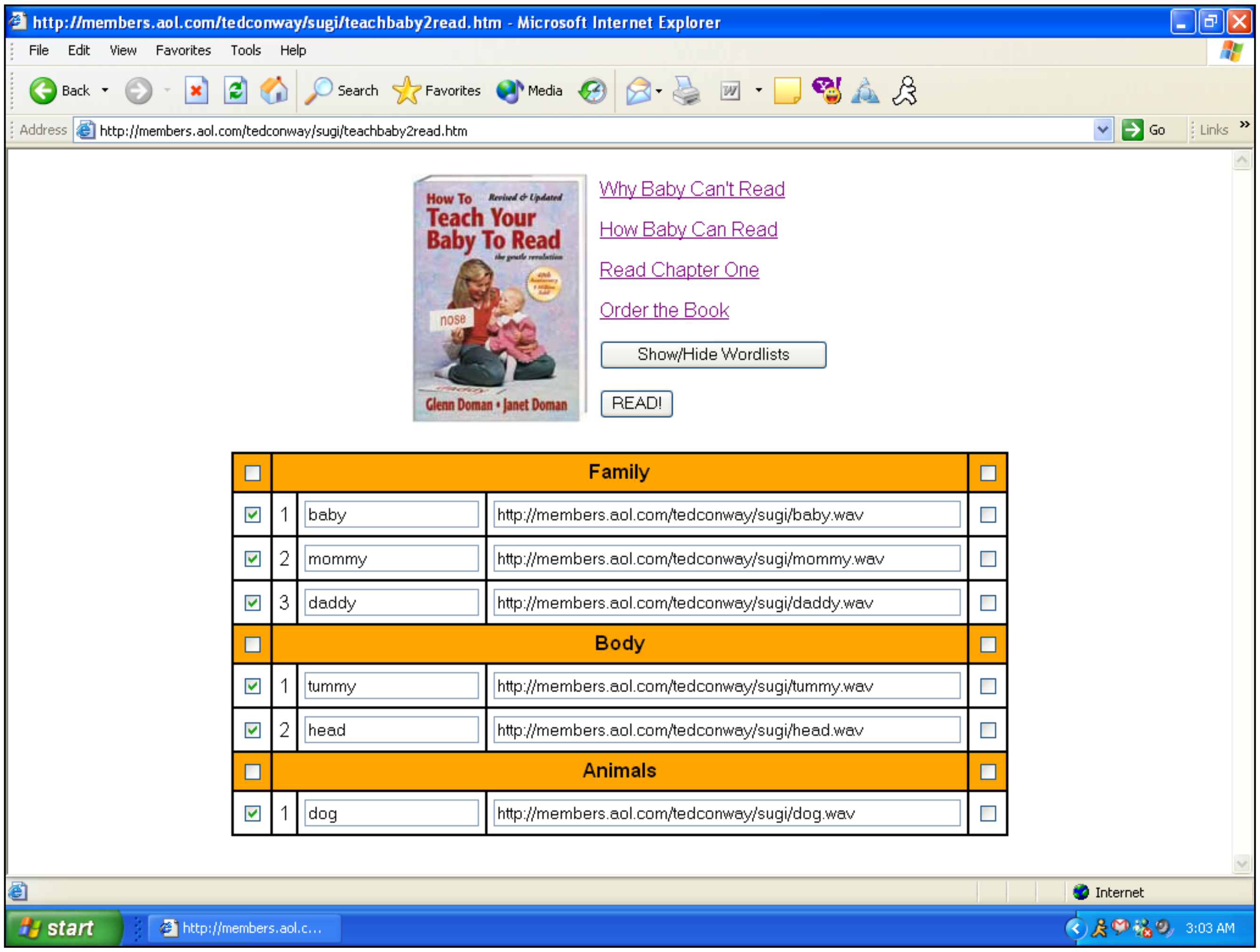
For an idea of how XML reading lists might be used to help teach babies or young children to read, visit the "hack-in-progress" at http://members.aol.com/tedconway/sugi/teachbaby2read.htm.

This is an admittedly crude IE-based prototype of free software that could be used to provide "talking flashcards" for use in conjunction with a *simple* method developed by child-development expert Glen Doman that holds the promise of revolutionizing how children learn to read.

At the risk of oversimplifying, Mr. Doman observed that we make writing **too small** and simply have to **MAKE IT BIGGER** to allow children to read at a much earlier age – much as one doesn't whisper when trying to teach a baby to listen and speak!

Not only do flashcards automatically built from XML reading lists provide an easier-to-use alternative to the hand-lettered flashcards called for by the teaching techniques Doman describes in his landmark book, ***How to Teach Your Baby to Read***, they can also be easily duplicated, accessed from anywhere an Internet connection is found – home, school, library, or even Grandma's – and made available for virtually no cost to a much wider audience than the two million adults who have read Doman's book!

But even if this proof-of-concept doesn't amount to anything (a not-so-unlikely possibility!), I'll have at least learned a whole slew of crazy XML, Javascript, and HTML techniques in the process!


http://members.aol.com/tedconway/sugi/teachbaby2read.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://members.aol.com/tedconway/sugi/teachbaby2read.htm Go Links
How To Teach Your Baby To Read
Revised & Updated
nose
Glenn Doman • Janet Doman
Why Baby Can't Read
How Baby Can Read
Read Chapter One
Order the Book
Show/Hide Wordlists
READ!
Internet
start
http://members.aol.c...
3:03 AM

☐		Family		☐
✓	1	baby	http://members.aol.com/tedconway/sugi/baby.wav	☐
✓	2	mommy	http://members.aol.com/tedconway/sugi/mommy.wav	☐
✓	3	daddy	http://members.aol.com/tedconway/sugi/daddy.wav	☐
☐		Body		☐
✓	1	tummy	http://members.aol.com/tedconway/sugi/tummy.wav	☐
✓	2	head	http://members.aol.com/tedconway/sugi/head.wav	☐
☐		Animals		☐
✓	1	dog	http://members.aol.com/tedconway/sugi/dog.wav	☐

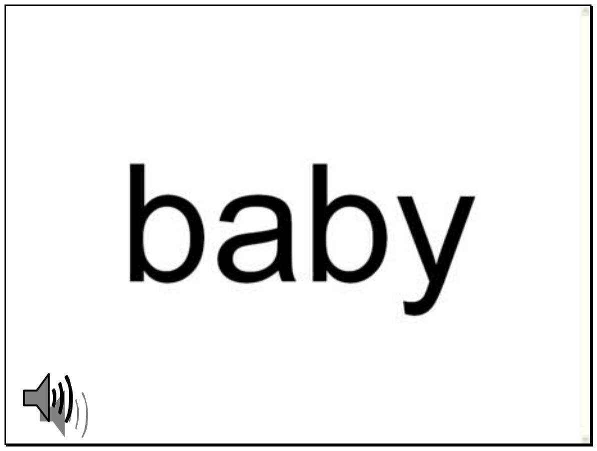
OFF-TOPIC BONUS

A WORD OR TWO ON "TALKING FLASHCARDS"

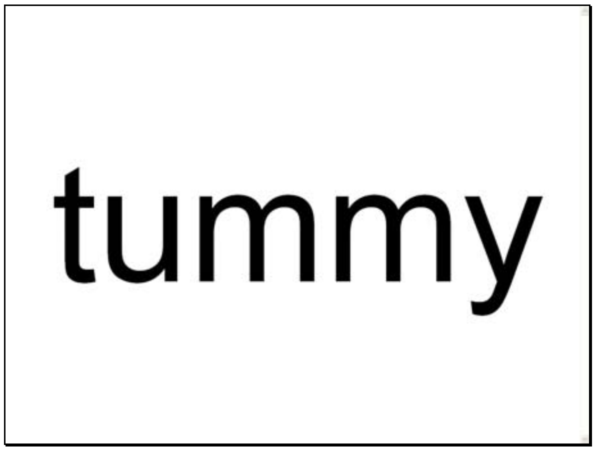
When you press the **READ!** button on the form, all of the checked words will be displayed one-by-one in a new full-screen window and read aloud.

Fonts are scaled dynamically to make each word as large as possible, and the words are "read" by playing the specified .wav files, which could be local files or on the Internet.

BTW, Internet Explorer's HTML+TIME feature is used to synchronize the words and audio.

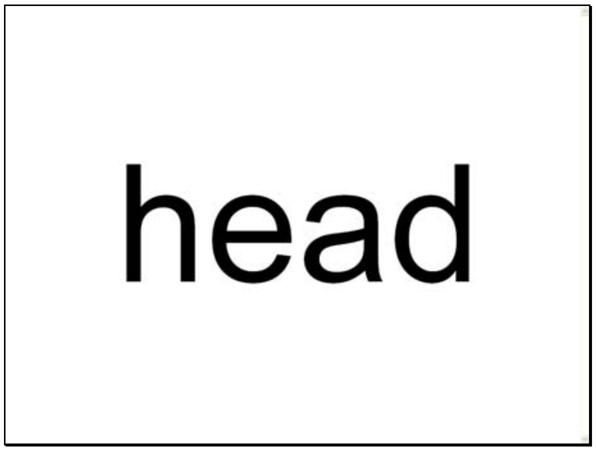


baby

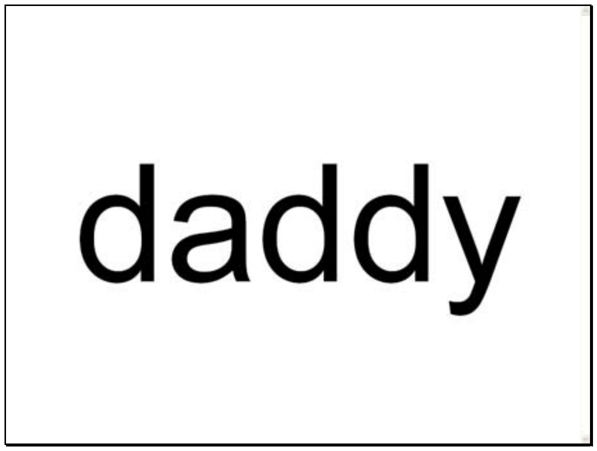


tummy

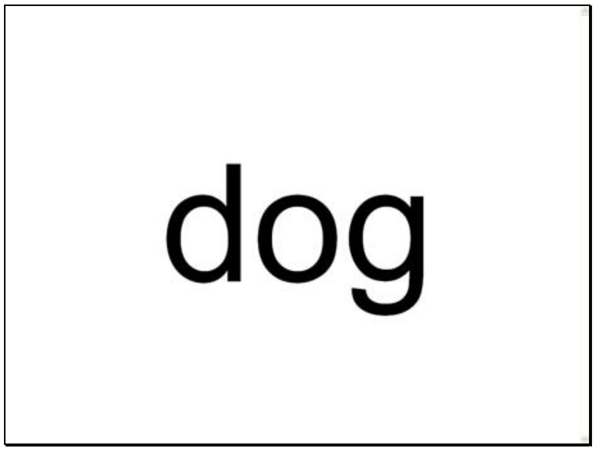
mommy



head



daddy



dog