

Fast Prototyping using SAS/AF® Software

Phil Busby, Live Data Systems, Inc., Apex, NC

ABSTRACT

You wake up in a hospital bed, and you can neither write nor speak, but you can see and think clearly. How do you communicate with others? This system, written in SAS/AF®, makes it possible to communicate with others as long as you know the same language, communicating thoughts as sentences, which are made up of words, which are made up of up to 31 different letters.

INTRODUCTION

Suppose you want to prototype an application to give two people who know the same language a way to communicate when one of them cannot speak nor write. Imagine a person who finds himself incapacitated by having lost muscle control for talking and writing. That person must suffer greatly because he wants to talk, but cannot, and nobody knows what he wants to say.

To assist such people I wanted to test a programmable algorithm for receiving messages from someone who cannot write nor speak. The receiver of the message operates the computer. The sender must be able to see an easy-to-read projection of the computer screen on the wall or ceiling. The working prototype was developed under SAS/AF® 8.2 on a Unix platform. This paper describes the system using screenshots from that prototype. The fast prototype development environment of SAS/AF® Software allowed me to get the screen built quickly.

The goal is a system that allows absolutely clear transmission of messages from people who cannot speak nor write. The output text could feed a speech synthesizer that would allow the incapacitated person to have a voice again. The receiver also benefits from learning exactly what the sender wants to say. The sender may know vital information, where his ability to communicate precisely is important, such as “Who shot you?” or “What’s the combination to the safe?”

USER REQUIREMENTS

Now, before we look at the computer screen for the software, here are some parameters: the person sending the message must be able to answer yes/no questions somehow. Suppose he is so weak that just nodding his head “yes” is a lot of effort. We can still receive this person’s message by agreeing with him that a nod means “yes” and no movement means “no.” We also must agree on what language we are using. As you can see, this system would work with any language that expresses thoughts in “sentences” made up of “words” made up of up to 31 “letters.”

So how could we do it? The sender can answer only “yes” or “no” to our questions. So, we could start by asking the sender, “Is the first letter of the first word of your message the letter A?” And if not, ask, “Is it B?” “Is it C?” And so on, until the sender answers “yes” and we know the first letter of the first word. You can see how tedious this brute force search would be. If we are going to learn the text of the message one letter at a time, then we should seek to minimize the number of “yes/no” questions we must ask. One way would be to ask about each letter in decreasing order of frequency in the language: “Is it E? T? A? O? N? ...” in English, but that’s still a sequential search.

The key to reducing the number of questions needed to identify any letter out of 31 is the 31x5 binary table. The values going down the left side are the row numbers in decimal, and next to that column are five binary digits: a 1 or a 0 for each. To the right of those are the possible letters. The sender, looking at the table, finds the row with the first letter of the first word of his message, and reads across to see the five ones or zeros next to it. The receiver then asks, “Is the digit in the first column of your selected row a one?” And the sender nods “yes” or not to transmit the value in column 1. This process is repeated for columns 2-5. The receiver can then look on the table for the five binary digits that match the sender’s answers, and read off the corresponding letter. The letters 27-31 need not be used, or can be used for other symbols, such as digits. Five zeroes are used as an escape character.

THE BINARY TABLE

1	A	0	0	0	0	1
2	B	0	0	0	1	0
3	C	0	0	0	1	1
4	D	0	0	1	0	0
5	E	0	0	1	0	1
6	F	0	0	1	1	0
7	G	0	0	1	1	1
8	H	0	1	0	0	0
9	I	0	1	0	0	1
10	J	0	1	0	1	0
11	K	0	1	0	1	1
12	L	0	1	1	0	0
13	M	0	1	1	0	1
14	N	0	1	1	1	0
15	O	0	1	1	1	1
16	P	1	0	0	0	0
17	Q	1	0	0	0	1
18	R	1	0	0	1	0
19	S	1	0	0	1	1
20	T	1	0	1	0	0
21	U	1	0	1	0	1
22	V	1	0	1	1	0
23	W	1	0	1	1	1
24	X	1	1	0	0	0
25	Y	1	1	0	0	1
26	Z	1	1	0	1	0
27	1	1	1	0	1	1
28	2	1	1	1	0	0
29	3	1	1	1	0	1
30	4	1	1	1	1	0
31	5	1	1	1	1	1

The first time I used this binary table, it was written on a piece of paper with a pencil. Most people take one look at this table and their eyes glaze over and they think it's just too complicated. That's where the computer comes in. The computer can do the work for us so we don't have to know anything about the table. All the receiver does is record the sender's answers: one or zero. After each set of five binary digits is entered, the computer displays the selected letter and starts building the first word with it. You don't have to memorize binary conversions or do any math. Just type 1 or 0, and eventually you can read the message that the sender is trying to send you.

If the sender can signal both "yes" and "no" then he can send all five binary digits at once, greatly speeding up transmission. For example, if the sender blinks the left eye for "no" and the right eye for "yes" then the letter A would be sent as four winks of the left eye and one wink of the right eye.

Under that scenario the sender could reduce the number of binary digits needed to identify letters using Morse code, which uses a variable number of dots or dashes to spell letters. The problem is, few people needing this system are likely to know Morse code. Precise timing is needed when transmitting in Morse code because the pauses delineate ends of letters, words, and sentences. The learning curve for it is too steep. We need a system that can be learned in just a few minutes. For some of our users, they might not live much longer than that. So using five binary digits allows enough to accommodate the English alphabet and many others. It is true that we could double the supported alphabet size with the addition of each additional row of binary digits. Let's leave that for the truly international version. For the prototype, English support is enough. We type out the English alphabet as the only alphabet file. It is loaded at system initialization time, and the prototype runs in English. So in a couple of hours of interactively coding and testing with SAS/AF® Software, we can show a working system.

INCREASING TRANSMISSION SPEED

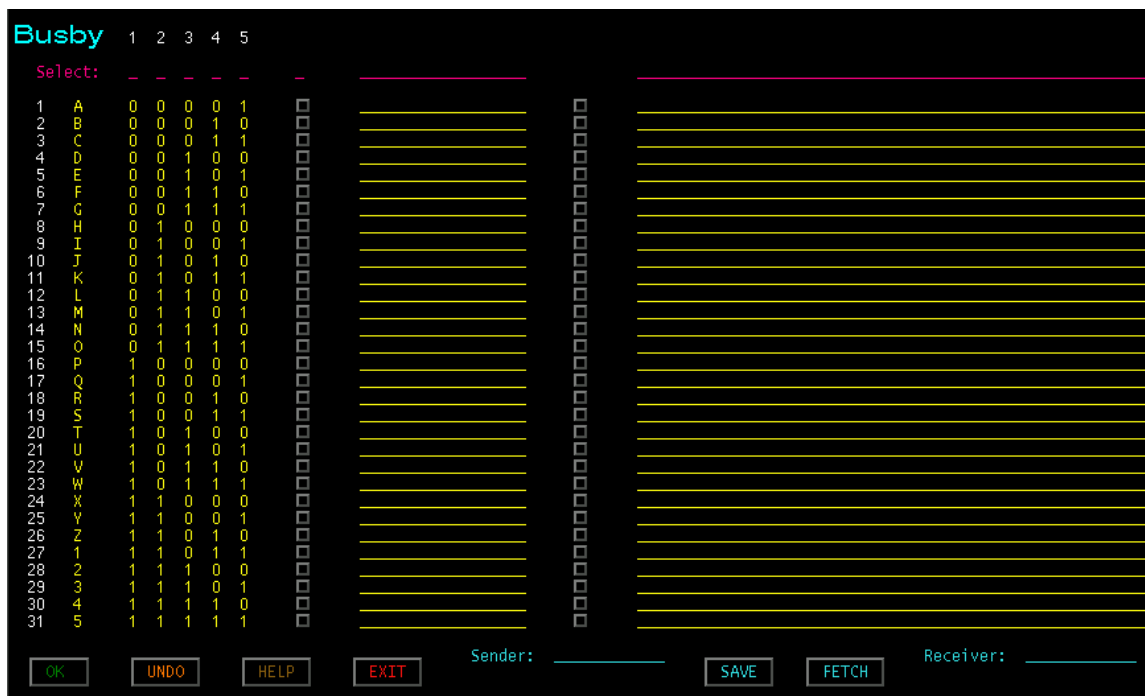
Once the sender sees how to use the binary table to select letters, then it's just as easy to select words and sentences the same way. This allows another huge improvement in communicating with someone who cannot talk nor write. The receiver might have special knowledge of the sender and can anticipate possible words that the sender might like to use in his message.

The ability to change the pre-loaded list of words and sentences is implemented with the SENDER and RECEIVER screen fields and the SAVE and FETCH buttons. When the button is clicked, your SCL code for it runs. The computer stores the lists of words and sentences for each sender/receiver pair. You can switch lists by typing over the receiver's name and by pressing the FETCH button. The functionality is useful because the thoughts the sender would want to say to a family member would likely be very different from the things he would want to say to his doctor, for example. So when the doctor comes in, click SAVE, type in the doctor's name in the receiver field, and click FETCH, and you are ready to receive messages for the doctor.



If the sender can select words using the binary table, he can just as easily select whole sentences, so you can also offer the sender a list of 31 one-line sentences. With these enhancements, you start by asking the sender, “Do you want to select an existing sentence from this list?” And if he answers “Yes” then the sender transmits just five binary digits next to the sentence he wants to say, and we get the whole sentence. If “No” however, we then ask, “Would you like to start a new sentence with one of these 31 words?” And this allows an entire word to be transmitted with just five 1's or 0's. The following screen shot of the SAS/AF prototype. The OK button is clicked when the entire letter, word, or sentence is selected to move that selection into the output fields.

Prototype of the Busby™ System



In the prototype, the receiver clicks OK to process each set of 5 binary digits. If an entire sentence is selected,

The system displays it on the top red line. If a word is selected, then the word is moved to the sentence under construction with the OK button. A second OK click signals the end of the word. If a letter is selected, then the OK button appends it to the word under construction. A second OK click in that mode means that the sender says that the current word is complete. The SCL code behind this screen is only 230 lines. Here is the declaration of the variables and arrays needed:

```
length let $1;
length chkword $20;
array l{31} $1 l1-l31; /* letter */
array n{31} $2 n1-n31; /* number of the letter */
array q{31} $1 q1-q31; /* column 1 binary digit */
array r{31} $1 r1-r31; /* column 2 binary digit */
array s{31} $1 s1-s31; /* column 3 binary digit */
array t{31} $1 t1-t31; /* column 4 binary digit */
array u{31} $1 u1-u31; /* column 5 binary digit */
array w{31} 8 w1-w31; /* word checkboxes */
array x{31} 8 x1-x31; /* sentence_checkboxes */
array c{5} $1 c1-c5; /* column binary value sent */
array sel{31} 8; /* 1=letter selectable 0=not */
array word{31} sug1-sug31; /* suggested words */
array sen {31} sen1-sen31; /* suggested sentences. */
```

Now suppose the sender starts a new word and sends us the letters F, E, and E. After receiving the third letter you must interject the question, “Is THAT the word?” because FEE spells a word in English. If the sender says “no” then you ask for the next letter, which is D. This time the sender answers “Yes, that’s the word.” The second word is transmitted: “dog.” At that point we must interject the question, “Is THAT the sentence?” because the two words might make a complete sentence, or might not.

Not implemented but possible would be the addition of the “auto-complete” feature of many text editors. Usually the receiver can guess the word after just a few letters. If not, the computer can supply the complete word after enough identifying letters are sent. This feature would require an on-line dictionary for the language. Also, the prototype shows a fixed position and full display of all possible lists, but in future versions, the binary table would be better moved next to the list the sender has just selected: sentence, word, or letter, for easier viewing. Having a fast prototyping full-screen development environment to work in such as SAS/AF® Software allows you to see what works fast, and gets you ready for designing the production version.

CONCLUSION

If you have an idea for an interactive computer system, you can prototype it fast with SAS/AF® Software.

CONTACT INFORMATION

Phil Busby
Live Data Systems, Inc.
342 Calico Drive
Apex, NC 27523
Work Phone: 804/952-7275
E-mail: fransioli@hotmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.