AN INTRODUCTION TO VREXX

CRAIG SWANSON
UCSD

An Introduction to VREXX

Craig Swanson
San Diego OS/2 User Group

REXX symposium
La Jolla, California
May 18, 1993

VREXX - Gateway to Graphical REXX for OS/2

is REXX-aware applications for OS/2 2.0 and 2.1 come to market, the system scripting abilities of the language will allow OS/2 users to write a REXX programs to tie together multiple applications to perform complex actions. For example, a REXX script for OS/2 might allow a user to double-click an icon in the Workplace Shell to start a telecommunications program, dial up a remote service such as CompuServe, retrieve stock prices-and news regarding a stock portfolio, and then take that information and send it to a spreadsheet to create new stock trend graphs and update the current value of the portfolio. But even without using such REXX-aware programs as *Borland ObjectVision for OS/2*, REXX programs for OS/2 can have a graphical user interface. VREXX, short for Visual REXX for Presentation Manager, was written by Richard B. Lam of the IBM T.J. Watson Research Center to allow REXX for OS/2 to have a Presentation Manager user interface complete with windows, dialog boxes, text (even in varied fonts and colors) and graphics without the programmer having to learn the intricacies of writing PM programs in C or C++ programming languages. VREXX can be found in the archive called `VREXX2.Z IP` which is available on *OS/2 Connection* bulletiu board in La Jolla (619-558-9475) and many other bulletin boards and FTP sites. It is distributed under the IBM Employee Written Software plan that permits pmgrams to be released free of charge but without any guarantee of product support from IBM.

Simple VREXX Calculator Example

We'll examine a short VREXX program to show the essentials of using the package. Take a look at the program listing labelled VCALC. `CMD`. Please note that the line numbers are not really part of the program but are simply there to make it easier to point out the interesting parts of the program. The first six lines of the program are comments. As you know, every REXX program must start with a comment. I added a few others to note what the program is supposed to do and when it was written. Line 7 is the first `that does any real work. The CALL` instruction transfers control oi the program to a subroutine provided by REXX for OS/2. This subroutine is named `RxFuncAdd` aud will add a new function to the REXX environment called `VInit`. The `VInit` function is found in the `VREXX.DLL` file andiusidethat file has the name `VINIT`. Then online 8, the `VInit` function is called to add all the other VREXX functions to the REXX environment. If it fails, the value `"ERROR" is stored in` the variable "initcode" and the `SIGNAL VREXXCleanup` instructionis run, thus transferring control of the program to code that will shut down VREXX and terminate the program.

Normally the `VInit` call should not fail, so in this case lines 10 and 11 tell the program to jump to the `VREXX-Cleanup` label if the program fails or is asked to end for some reason. Through experimentation, I found that line 15 is required to handle cases where the user types in a bad expression like `"5 / 0"` which causes a divide by zero error. REXX considers this a syntax error. When an error `likethis` happens, `VCALC.CMD assumes it` isbecausethe user made a mistake and then jumps to a block of code that will tell the user a bad expression was entered.

So far the program has set up the REXX environment to permit the use of VREXX. Lines 21 to 23 specify the title for the input window, its width in characters, and the type of buttons it should have. For some reason numbers must be used for button types and the numbers are not very well documented, possibly because VREXX is freeware. I figured out which number to use by examining the sample programs that came with `VREXX2.ZIP`. Lines 28 and 29 set up the set of strings that will be used to prompt the user for input Stem variables are used for this and the variable ending in `".0"` tells VREXX how many prompt strings to expect starting with the one ending in ".1". The variable ending in `".vstring"` is used to specify the initial string `displayed` in the input box entry field. For this program, I didn't want there to be any text in the entry field at first, so the two adjacent double quote marks are used to indicate an empty string. Line 33 finally displays the input dialog box and waits for the user to press the OK or CANCEL buttons. The name of the button that was pressed is stored in a variable `named "but ton"` and the user's input is stored back into `"prompt.vstring"` which on line 35 is then copied into the variable `expr`.

Line 37 checks to see if the OK button was pressed. If it was, then lines 38 to 47 evaluate the expression usiug one `Of the more unique features` of REXX, the `INTERPRET` instruction. The answer is stored in the variable named `"result"` and finally displayed on the screen in a message box that will be displayed until the user clicks on the OK button. Then the program jumps `to the InputLoop`

label to get the next expression from the user.

If line 37 decided that the OK button had not been pressed, the THEN clause would not have been run and instead the next instruction run would have been on line 53. The "CALL VExit" instruction tells the VREXX code to shut itself down. Finally, line 54 terminates the REXX program If you do not do a "CALL VExit" before ending a VREXX program, there is a program file named VREXX.EXE that is left running. Until that program is terminated, other VREXX programs will not be runnable from the session where you started VCALC.CMD.

You may be wondering that if line 54 terminated the REXX program, why are there lines after it? I decided to put the block of code to handle expression errors after the EXIT instruction. Since this block of code is jumped to because of the SIGNAL ON instruction on line 15, it is OK for it to be after the EXIT instruction. Lines 57 to 69 merely display a message box telling the user that the expression typed was bad. After the uses clicks on the OK button in the message box, then the SIGNAL Inpu tLoop instruction causes the program to loop back to get more input.

VREXX has a lot of other abilities that I haven't cov ered, but this program illustrates the basics of calling VREXX functions that you'll need to do anything more complicated. VCALC.CMD may not very useful as a tool, but it was a helpful exercise for me to leam the basics of VREXX by writing a program that accomplished something.. If you run OS/2, type in the program and try it out. If you don't want to retype if you can get a copy of VCALC.CMD in the electronic version of the March 1993 issue of the San *Diego* OS/2 Newsletter which is available as SDIN9303 .ZIP on *OS/2 Connection*. VCALC.CMD is included inside the ZIP archive file.

How VREXX Works

If you are already familiar with OS/2 programming, you might know that REXX programs are usually run by the CMD.EXE command line interpreter using various DLL files stored in the \OS2\DLL directory such as REXX.DLL and REXXAPI.DLL. You might be wondering how a text mode program like CMD.EXE can display PM windows and dialog boxes. The answer is it can't, at least not on its own.

By using the PSTAT, PSPM2, and OS20MEMU tools while running a VREXX program, I've been able to determine that the VREXX program is actually functioning as a client of a PM program that it has spawned to manage the display. When the VInit() function is executed in the REXX program, it appears that a shared memory region named "\MEM\VREXX\V#" (where # is a number representing the particular VREXX program running) is created. Then a PM program named VREXX.EXE is spawned. The CMD.EXE and VREXX.EXE programs communicate via this shared memory region. This allows the client REXX script being run in the CMD.EXE process to request PM services to be provided by the VREXX.EXE process it is using as a server.

VREXX.EXE has two threads. I'd speculate that one of these threads contains the main PM message loop and that the other communicates with the REXX program. It uses the services of two DLL files supplied with VREXX which are DEVBASE.DLL and VREXX.DLL. DEVBASE.DLL appears to be more than just a supporting library for VREXX as inside it has text strings such as "OS/2-AIX Development Base" and what look to be Adobe PostScript commands. What else it might do it unclear to me.

VREXX.DLL appears to be code used by both the CMD.EXE and VREXX.EXE processes. If you kill one of these processes without killing the other, the remaining process appears to be destabilized so it crashes with a protection fault. Also if you do not do a "CALL VExit" in your REXX program, the CMD.EXE process cannot run additional VREXX programs and in fact may disappear entirely in what also appears to be the result of a protection fauii Lastly, it appears that there is a limif on the number of VREXX programs that can be run at one time. I was not able to run more than two at once. Tryiug to start additional VREXX programs resulted in the command line sessions disappearing, probably due to a protection fault while running in the VREXX.DLL code. I do not see any reason why such a low limit is required by the approach that appears to be used to make VREXX function, so perhaps this was an oversight in the original code. After all, it is a 1.0 release. Or maybe something is not being cleaned up properly due to the way VREXX is architected using DLL's and shared memory. While experimenting with VREXX, I've noticed symptoms such as the second of two concurrently executing VREXX scripts not starting up consistently which indicate that the latter might be what is really happening.

Helpful Tools for VREXX Programmers

Since sometimes things go wrong when writing a VREXX program (after all, programmers do make mistakes), it is possible that you- will leave VREXX.EXE processes running when a VREXX program stops with an error before executing "CALL VExit" to terminate the VREXX environment nicely. Therefore I'd recommend that you download a pair of files from *OS/2 Connection* called PROCS21.ZIP and KILLEM21.ZIP. These programs will let you list running processes to find the process ID number of VREXX using the "procs" program and then let you kill the VREXX program using "killem" followed by the process ID number of VREXX. The archive PSPM2.ZIP contains a single PM program to perform the same functions if you prefer graphical user interfaces.

I hope this introduction to VREXX has given you a starting point to experimenting with graphical REXX programs. If you have questions or feedback for me, you can send them to "Craig_Swanson@f354.n202.z1.fidonet.org" on Internet. Please include a reply-to address in your message in case your address is stripped by any mail gateways.

VCALC.CMD

```
 1: /* VREXX simple calculator program ● /
 2: /* San Diego OS/2 Newsletter        */
 3: /* March 1993 edition              */
 4:
 5: /* Program Initialization */
 6:
 7:   CALL RxFuncAdd "VInit", "VREXX", "VINIT"          /* Add VInit function to attach to VREXX */
 8:   initcode = VInit()                                /* Initialize VREXX */
 9:   IF initcode = "ERROR" THEN SIGNAL VREXXCleanup    /* Exit program if VInit() failed ● /
10:
11:   SIGNAL ON FAILURE NAME VREXXCleanup               /* If the program fails or stops for any    */
12:   SIGNAL ON HALT NAME VREXXCleanup                  /* reason, the VREXX cleanup must be done   */
13:                                                     /* in order to leave VREXX in a known state */
14:
15:   SIGNAL ON SYNTAX NAME SyntaxError     /* Syntax errors should only be triggered by bad */
16:                                         /* user input, so when one happens, tell the user ● /
17:                                         /* the math expression was bad.                   */
18:
19:/* Main Program ● /
20:
21:   windowTitle = "VREXX Calculator 1.0"  /* Title of input window ● /
22:   dialogWidth = 50                      /* Input dialog should be 50 characters wide ● /
23:   buttonType = 3                        /* type 3 means use OK and CANCEL buttons */
24:
25:
26: InputLoop:                              /* Label used for looping back to get more input ● /
27:
28:   prompt.0 = 1                                                  /* Only one prompt string */
29:   prompt.1 = CENTER( "Enter a math expression:", dialogWidth )  /* This is the prompt string. ● /
30:   prompt.vstring = ""                                          /* No default expression */
31:
32:   /* Get input from user ● /
33:   button = VInputBox( windowTitle, prompt, dialogWidth, buttonType )
34:
35:  expr  = prompt.vstring                 /* Store the expression the user typed */
36:
37:   IF button = "OK" THEN DO              /* If the OK button was pressed */
38:       INTERPRET "result =" || expr    /* evaluate the expression */
39:
40:       text.0 = 1                        /* and then show a one-line result ● /
41:       text.1 = result                   /* in a message box on the screen */
42:
43:       /* Show the message box */
44:       CALL VMsgBox "Result of <" || expr || ● >", text, 1
45:
46:       SIGNAL InputLoop                  /* Go get the next expression */
47:   END
48:
49:   /* The OK button wasn't pressed so exit the program. ● /
50:
51: /* Program Exit */
52: VREXXCleanup:
53:    CALL VExit                           /* Clean up the VREXX resources */
54:    EXIT                                 /* Terminate the program        */
55:
56:
57:/***** ERROR HANDLER ● ****/
58:
59:/* Display an error message ● /
60: SyntaxError:
61:   SIGNAL ON SYNTAX NAME SyntaxError     /* Reinstall error handler ● /
62:
63:   text.0 = 2                           /* Show a two line display */
64:   text.1 = "Bad expression:"           /* of the mistake          */
65:   text.2 = "  " expr
66:
67:   CALL VMsgBox "Error", text, 1        /* Show the message box with just an OK button */
68:
69:   SIGNAL InputLoop                     /* Go back and get more input */
```