BVDoor v1.0 Wide Beta
Copyright (C) BV Compuworks Group 1995-1996
Release Date: Dec 19, 1996

Wide BETA License Agreement

THIS IS A LEGAL AGREEMENT TO WHICH YOU ARE CONSENTING TO BE BOUND BY. IF YOU CANNOT AGREE TO ALL TERMS OF THIS AGREEMENT, ERASE ALL SOURCES OF DATA PERTAINING TO THIS SOFTWARE. BY NOT ERASING THE DATA, AND CONTINUING TO USE THE SOFTWARE BEYOND THE EVALUATION PERIOD, YOU AGREE TO ALL TERMS SET WITHIN.

1. B.V. Compuworks Group ("BVCom") grants to you a non-exclusive, non-sublicensable, license to use version 1.0 wide beta of the B.V.Door (the "Software"), for evaluation and trial purposes only of no greater than 30 days.

2. Any and all contents, accessed through the Software, are the properties belonging to the applicable content owner; and may be protected by applicable copyright laws. This License gives/grants you no rights to such content.

3. All titles, ownership rights and intellectual property rights, to the Software, shall remain in BVCom and/or its suppliers and distributers. By consenting to this licensing agreement, you acknowledge that the Software, in source code form, remains a confidential trade secret of BVCom and/or its suppliers and distributers. You also agree not to modify the Software or attempt to alter, change, decipher, decompile, disassemble, hack or reverse engineer the Software, except to the extent applicable laws specifically prohibit such a restriction.

4. BVCom may terminate this license, at any time, by delivering notice to you and you may likewise terminate this license, at any time, by simply destroying and/or erasing all copies of the Software and corresponding documentation.

5. BVCOM MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE SOFTWARE, ALL ACCOMPANYING MATERIALS, AND ANY CONTENT OR INFORMATION MADE ACCESSIBLE BY THE SOFTWARE, FOR ANY PURPOSE. THE SOFTWARE IS PROVIDED "AS IS" AND UNDER NO CIRCUMSTANCES, LEGAL THEORY, TORT, CONTRACT OR OTHERWISE, SHALL BVCOM AND/OR SUPPLIERS AND DISTRIBUTERS BE ACCOUNTABLE AND/OR LIABLE TO YOU OR ANY OTHER PERSON FOR AN DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY KIND. THE SOFTWARE COMES WITHOUT EXPRESS OR IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT.

6. For further assistance pertaining to the Software and/or documentation please send all inquiries to the Program Manager
or the programmer:

Greg Varga, Co-Owner/Program Manager
B.V. Compuworks
18130 60th Street
Cloverdale, B.C. (Canada)
V3S 1V6
(604) 576-2219
E-Mail Address: xomorph@skybus.com
Fidonet Address: Greg Varga or Allan Rafuse @
Conceptual Realities BBS (1:153/9118)

Registration Information

Currently there are two registration packages that are available for BVDoor v1.0. The first package is the basic package which is the required by everyone which is $30.00 Canadian; it will be raised fairly soon to $50.00 in upcoming versions. The second package is a for commercial programs. This package is needed if you are going to release your programs under a business name or as a commercial program; excluding Shareware, Freeware, Crippleware programs. The fee for commercial registration is $50.00 Canadian for EACH program that is to be released. This will be raised soon to $80.00 in upcoming versions. See REGISTER.DOC for the registration form. The basic package is good for 2 major releases of BVDoor and is good for all minor versions which will basically be bug fixes, optimization, ect. A major upgrade is comprised of addition of many new functions, a bunch of minor upgrades, or adding addition BBS packages support, ect. Upgrading the basic package to a new major version costs $15.00 no matter what the last version is.

TABLE OF CONTENTS

About BVDoor

BVDoor is a pascal unit for OS/2 that allows programmers to easily create their own door programs. BVDoor makes it easy because the programmer does not have to worry about the comport or user information; BVDoor does it all for you. If you need to change some of the users information, you can do that too. BVDoor was created to encourage programmers to create OS/2 executable online door programs. To make it easy for an already made program to be ported to OS/2, BVDoor was cloned to a well used door unit; JPDoor. BVDoor follows JPDoor as closely as possible. Currently BVDoor has been only with RA 2.xx, Ezycom 1.20 and AdeptXBBS v1.07f. QBBS 2.75+ structures are included, but are only still only in theory because I can't find a copy. Maximus, SuperBBS and PCBoard support will be coming in the near future. Along with Maximus will come Squish message base support.

Authors Notes:

Hope this makes programming a lot easier! If you encounter any errors then send us a message telling us EXACTLY what happened. Please send a copy of your finished program to xomorph@skybus.com so what people are doing with BVDoor.

Getting Help/New Versions

You can get help from the authors of BVDoor at the following places:

Snail Mail:
Greg Varga, Co-Owner/Program Manager
B.V. Compuworks Group
18130 60th Street
Cloverdale, B.C. (Canada)
V3S 1V6
(604) 576-2219

E-Mail Address: xomorph@skybus.com
Fidonet Address: Greg Varga at 1:153/9118 or
Allan Rafuse at 1:153/9118

New versions can be downloaded from the following places:
FREQ: "BVDOOR" from 1:153/9118
FTP SITES: hobbes.nmsu.edu
ftp.cdrom.com
oak.oakland.edu
Web Sites: Virtual Pascal Homepage at:
http://www.fprint.co.uk/vpascal

Chapter I - Getting Started

This chapter will help you get started and familiarized with BVDoor if you are new to programming, BVDoor, or JPDoor. First off, open up a new file to start your program in. The first thing you'll learn how to do get your program off and running. Just follow the examples and include whatever you like. Your program should look like the following:

```
PROGRAM progname;

BEGIN
END.
```

Part I - The USES statement

This follows the commands of JPDoor very closly, but in BVDoor, all you need in the uses statement is **USES DOS, CRT, bvdoor2;** All the variables and calls are located in this unit so you don't have to worry about anything.

Part II - Trapping Exit Codes

BVDoor also allows you to trap halt codes and runtime errors if you use the far call procedure TRAPEXIT.PAS. This procedure checks how the program is exiting. If a runtime error occurred, then it is reported into a logfile in the directory where your program resides with the name 'bvlogname' + node number + '.ERR'. If you left the variables at their defaults a logfile of this name would be BVDOR001.ERR. If a halt code was used, then this procedure will call a procedure called TERMINATE passing the haltcode to it. Here is an example of a terminate procedure.
You can include this following segment into your program to make it easier for you.

```
PROCEDURE terminate(haltcode : BYTE);
 BEGIN
   WRITELN('- ', productname, ' Terminating - ');
   WRITE('Exit Type: ');
   CASE haltcode OF
            0 : WRITELN('Normal');
           1 :
         BEGIN
           WRITELN('Carrier Lost');
          {Do what you want in here}
         END;
```

```
          2 :
        BEGIN
          WRITELN('Timelimit Exceeded');
         {Do what you want in here}
        END;

          3 :
        BEGIN
          WRITELN('Inactivity Timeout');
         {Do what you want in here}
        END;
     4 :
        BEGIN
         IF doorsys THEN
           WRITELN('Can not find ', ckpath(exitinfopath),                    'Door.Sys')
         ELSE IF sessioninfo THEN
           WRITELN('Can not find ', ckpath(exitinfopath),
                 'Session.Info')
         ELSE
           WRITELN('Can not find ', ckpath(exitinfopath),                     'Dorinfo', getnode,
'.Def');
        END;
     5 :
        BEGIN
         WRITELN('Can not find ', ckpath(exitinfopath),
                  'Exitinfo.BBS');
        END;
     6 : BEGIN
          WRITELN('Change Directory Error');
          {Should close all files}
        END;
     8 : BEGIN
          WRITELN('RAXIT Semaphore File Found');
        END;
   END;
END;
```

Once you have the above copied into your program add the following right after it: **{$I trapexit}**
Now in the main body of the program add the following line: **doorexit := trapexit;**
Once you have done those three things, your program will be able to trap exit codes. Your program should look like this:

```
PROGRAM progname;

USES DOS, CRT, bvdoor2;

PROCEDURE terminate(haltcode : BYTE);
 BEGIN
   ...
   ...
```

```
END;

{$I trapexit}
```

```
(* Main Program *)
BEGIN
 doorexit := trapexit;
 getdorinfo('1', 'C:\BBS');
END.
```

Now if the program was ran, it would look for the Dorinfo1.Def file in 'C:\BBS'. If it couldn't find the Dorinfo1.Def file then a haltcode of 4 would be called. If it couldn't find the Exitinfo.BBS file, then a haltcode of 5 would be called. If everything went ok, then the program would terminate normally with a haltcode of 0.

Part III - Trapping 'Special' Keys

BVDoor allows you to define up to 20 key sequences for both the sysop keyboard and the remote keyboard. This allows you to call a procedure if a certain key is pressed. For example, if ALT-C was pressed on the sysop keyboard, a chat procedure could be called. Another example is, the user could press ALT-T to display how much time they have left. Lets trap call a chatting procedure when ALT-C is pressed on the sysop keyboard. To trap the ALT-C key stroke, add the following into your main program:

```
 sysopkey[1] := #0 + #46;
 sysopproc[1] := chatproc;
 sysopkeys := TRUE;
```

Now create a procedure called chatproc.

```
PROCEDURE chatproc;
 BEGIN
   sysopkeyson := FALSE;   { Turn of key trapping for now }
   chatting;               { Call your chat procedure     }
   sysopkeyson := TRUE;    { Turn key trapping back on    }
 END;
```

When the sysop presses ALT-C the chatproc will be called. The sysopkey trapping is turned off so you can't keep calling the chatproc from within the chatproc. Don't forget to turn it back on if you want to continue trapping keys after the chatting procedure is done. ALT-C is known as an extended key because it sends a null character (ASCII #0), then a second extended key. Included is a program called KEYFIND.EXE which tells you the extended codes of keys you typed.

Lets trap CTRL-T on the users keyboard for some more practice. Add the following into your main program:

```
 userkey[1] := #47;
 userproc[1] := timeleft;
 userkeyson := TRUE;
```

Now create a 'timeleft' procedure.

```
PROCEDURE timeleft;
 BEGIN
   userkeyson := FALSE;
   crlf;
   display(0, 7, 0, 'Time left: ' + itoa(timeremaining));
   userkeyson := TRUE;
 END;
```

Below is what you should have so far:

```
PROGRAM progname;

USES DOS, CRT, bvdoor2;

PROCEDURE terminate(haltcode : BYTE);
 BEGIN
   ...
   ...
 END;

PROCEDURE chatproc;
 BEGIN
   ...
   ...
 END;

PROCEDURE timeleft;
 BEGIN
   ...
   ...
 END;

{$I trapexit}

(* Main Program *)
BEGIN
 doorexit := trapexit;
 sysopkey[1] := #0 + #46;
 sysopproc[1] := chatproc;
 sysopkeys := TRUE;
 userkey[1] := #47;
 userproc[1] := timeleft;
 userkeyson := TRUE;  getdorinfo('1', 'C:\BBS');
END.
```

Part IV - Getting the Users Information

Now to get the users information, you'll have to know the current node number and path to the BBS dropfile. This can be accomplished in many ways. The most common way is through passed parameters. Just assume for now that the program is on node 1 and in the path to the BBS drop files is 'C:\BBS'. Now in the program put the following line:

getdorinfo('1', 'C:\BBS');

Now if the user loads the program from node 1 and the Dorinfo1.Def file and the Exitinfo.BBS file are in the 'C:\BBS' path, then the user information will be loaded in. If your door doesn't need to load in the ExitInfo.BBS file, then set the ckexitinfo variable to false:
ckexitinfo := FALSE.
This will tell BVDoor that you don't want to read in the Exitinfo.BBS file. If BVDoor is reading in a Dorinfox.Def dropfile and it can't find the ExitInfo.BBS file and the 'ckexitinfo' variable is TRUE, then a haltcode of 5 will be called. Set the 'ckexitinfo' variable to false if your reading in a Dorinfox.Def file and you don't need it any information from it.

By default, 'getdorinfo' always reads a Dorinfo1.Def file no matter what the node number is. If you want 'getdorinfo' to read a Dorinfo2.Def file or Dorinfo3.Def and so on, set the following variable to false:
forcenode := FALSE;
This tells BVDoor to readin a Dorinfox.Def file instead of always wanting to readin the Dorinfo1.Def file.
To tell 'getdorinfo' to read a Door.Sys dropfile, set the 'doorsys' variable to true before calling getdorinfo:
doorsys := TRUE;
To tell 'getdorinfo' to read a Session.Info dropfile set the 'sessioninfo' variable to true before calling getdorinfo:
sessioninfo := TRUE;

BVDoor works with an RA 2.xx ExitInfo.BBS structure. Therefore all other structures are translated into the RA 2.xx structure for your use and then converted back to their original structure. This is only done if the ExitInfo.BBS file is read in.
The main program of your program should look similar to:

```
(* Main Program *)
BEGIN
 doorexit := trapexit;
 sysopkey[1] := #0 + #46;
 sysopproc[1] := chatproc;
 sysopkeys := TRUE;
 userkey[1] := #47;
 userproc[1] := timeleft;
 userkeyson := TRUE;
 ckexitinfo := FALSE;    { Uncomment if you don't want to read
                                                the ExitInfo.BBS File }
```

```
 doorsys := TRUE;        { Uncomment if you want to read a Door.Sys dropfile. }
 sessioninfo := TRUE;    { Uncomment if you want to read a Session.Info dropfile. }
 getdorinfo('1', 'C:\BBS');
END.
```

Part V - The Status Bar

By default BVDoors 2-line status bar is off. If you would like the 2 line status bar on the sysop screen on, set the 'statusline' variable to true;

```
 statusline := TRUE;
```

If the statusline is on, then it contains some of the users information like, their name, timeleft, current time, node number and other things. BVDoor has 9 different bars containing different lines of information. To toggle the bars, press the function keys F1-F9. F10 will turn the statusline off. To turn this it back on, just hit any of the 9 function keys. You can also customize the status bars by putting your information into the following variables where [x] is the function key number:

```
 statuslinea[x]
 statuslineb[x]
```

'statuslinea' is the top line, 'statuslineb' is the bottom line and both are STRING[79].

The following codes are converted on the statusline:

Code	Value
~A	[ANSI] [ASCII] or [MONO]
~B	Baud Rate
~C	Current Time
~R	Time Remaining
~S	Security Level
~W	Snoop Mode: ON or OFF

Unlike JPDoor, BVDoor doesn't have a maximum amount of characters for each code. BVDoor was made so you can format the text how ever you want.

Here are the defaults for BVDoors' statlinex[1]:

```
   statlinea[1] := ' ' + ljust(username + ' of ' + user.city + '                    at ' + baudstring + '
BPS', 68) + 'Line [' + itoa(atoi(node)) + ']';
   statlineb[1] := ' ' + ljust('Security: ~S', 15) +              ljust('Time: ~R Mins', 25) +
ljust('~A', 7) +                        rjust(productname, 20);
```

BVDoor Unit Variables

BVDoor has some global unit variables which you can use in your program. Most of these are variables are initialized upon program startup but are actually filled in with the user information when 'GETDORINFO' is called.

activecount	BYTE	Keeps track of keyboard activity. This is decremented every minute. Once'activecount' reaches 0 a beep sounds on the users terminal. Once 'activecount' reaches 0, the program terminates with a halt code of 3.
adept175	BOOLEAN	True if an AdeptXBBS Session.Info drop file was read in.
aflag, bflag, cflag, dflag	STRING[8]	Holds the users flag setting. The flags are read from the ExitInfo.BBS file. If it's not read, then they will hold 8 spaces. If the Exitinfo file was read then it will be in the format of: 'XX-XX--X'
ansi	BYTE	Is set to 1 if the user has ansi turned on. If the user's ansi is turned off, then it is set to 0. This can also hold 5 if the user has Mono-Graphics.
baudstring	STRING[35]	This is the users baud rate and comport settings. Some packages add a '-R' after the baud rate indicating an Error-Correcting Connection.
buffers	BOOLEAN	Initially set to FALSE; this variable determines whether or not output buffering will be used. BVDoor does not use this. It is only here for JPDoor compatibility.
bvlogname	STRING[5]	These are used to name the file
bvlogext	STRING[3]	of the error.log.

bvidlehook	PROCEDURE	This is a kind of a special procedure. It is called every time BVDoor is in idle. It give up timeslices for multitasking systems. If you don't want your door to give up timeslices then

		set bvidlehook := NIL; You could also call your own timeslicing routines if you wish by assigning bvidlehook := your_procedure. For more information on this, see the 'everyminutehook' variable.
bv_debug	BOOLEAN	Initially set to FALSE. If this is set to TRUE, then BVDoor writes its' activity to a file called in the same directory as your door program with the name of DEBUGnnn.Log, where nnn is the node number. You can allow your users to turn this on by setting the environment variable in the batch file by: SET BV_DEBUG=ON. You could also add a command line parameter such as /D or /DEBUG that also turns this on.
checkactivity	BOOLEAN	If this is TRUE, then keyboard activity will be monitored. You might set this to FALSE if your program spends time doing something else and isn't calling BVDoor routines. Then your user won't be thrown off for keyboard inactivity.
ckexitinfo	BOOLEAN	If set to FALSE, then BVDoor won't look for a ExitInfo.BBS file. This is for someone who does not require it in their program or for compatibility for other BBS's.
ckmsgs	BOOLEAN	When set to TRUE, BVDoor will check for online messages every minute. If a message is found, then the variable 'msgwaiting' is set to TRUE.
clubchar	CHAR	Holds the character that BVDoor draws in the DRAWCARD procedure. It defaults to ASCII #5 which is the club character.
counter	INTEGER	Used for anything you want. Enjoy.
c_back	BYTE	This holds the current background color. Color codes are defined in

		the DISPLAY procedure.
c_blink	BYTE	This variable contains the selected state of the blinking attribute. A 0 means no blinking and anything else means blinking is in effect.

c_fore BYTE This holds the current foreground attribute. Color codes are defined in the DISPLAY procedure.

date STRING[20] This holds the date of when your program was loaded.

doorexit PROCEDURE This is a null procedure when BVDoor is initialized. This procedure is called when your program exits. If you want to trap halt codes or do anything else EVERY time the program exists, then assign doorexit to your exit procedure. See Trapping Halt Codes.

doorsys BOOLEAN This tells GETDORINFO to read in a Door.Sys file rather than the standard DORINFOx.DEF file if it's set to TRUE. This must be set to TRUE before a call to GETDORINFO. The following variables are filled in with Door.Sys:

username
userfirst
userlast
portnum
baudstring
timeremaining
ansi

The following are not filled in and are still at their defaults:

```
systemname := 'BBS';
sysopfirst := 'SYSOP';
sysoplast  := 'NAME';
sysopname  :=  'SYSOP NAME';
usercitystate := 'Unknown';
usersecurity := 0;
```

You can change any of these if you want after you call GETDORINFO.

dosver STRING[30] This strings contains the DOS/OS2 version. It will be either: 'DOS vX.X' or 'OS/2 X.X'.

dv BOOLEAN Is TRUE if desqview is detected.

dvver STRING[30] This is blank unless desqview was detected. It holds the desqview version and window number.

		'DESQview X.X [Window X]'
everyminutehook		
	PROCEDURE	This procedure gets called every minute. This is a null procedure when the program starts up. This

allows you to assign it a procedure to call to do you own maintenance every one minute. For example you could make your computer beep everyone minute. Example; define a procedure:

```
{#F+}  {MUST be a Far Call Proc}
PROCEDURE dobeep;
        BEGIN
                WRITE(#7);
        END;
```

Now in your main program type the following:

```
everyminutehook := dobeep;
```

Now every minute when BVDoor does it's own internal maintenance, you can do any cleanup you want.

everyfivesecondhook PROCEDURE — Same as 'everyminutehook' except that this is called every 5 seconds.

exepath STRING[79] — This holds the full drive and path of where your program is running. This is useful for if your program is not run in it's directory and you need to access some of it's own files. This allows you to run your door from any directory you want like a BBS drop path directory and your program can still find it's data files.

exitconverted BOOLEAN — If the ExitInfo.BBS file was converted to a RA 2.xx format during a call to GETDORINFO, then this will be TRUE, otherwise it will be FALSE. DO NOT MODIFY THIS YOURSELF AS IT WILL SCREW UP THE DROPFILES!!!

exitinfo exitinforecord — This holds the ExitInfo.BBS file data. If the ExitInfo.BBS was converted to an RA 2.xx format then some of the fields will be blank.

exitinfobbsname STRING — Contains the full path and file name of the ExitInfo.BBS file.

It is initialized when GETDORINFO is called and if CKEXITINFO is TRUE.

exitinfopath STRING Contains only the path to the ExitInfo.BBS File. It is " until GETDORINFO is called. It is set to

		the directory that is passed to GETDORINFO. Use this whenever you want to find the dropfiles.
extendedchars	BOOLEAN	Defaults to FALSE. If you want users to be able to input extended characters in input routines, then set this to true. When it's FALSE, it helps act as a safeguard against line noise.
forcenode	BOOLEAN	Forces GETDORINFO to read only a DORINFO1.Def, no matter what node is passed to it. It defaults to TRUE so if you want to read a DORINFO2.DEF set this to FALSE.
fossilhot	BOOLEAN	If this is set to TRUE, then the fossil driver will be de-initialized upon program exiting. BVDoor does not use this. It is only here for compatibility with JPDoor.
fossilinstalled	BOOLEAN	Is TRUE if a fossil driver was detected during a call to 'initfossil'. Returns true in 'local' mode. BVDoor does not use this. It is only here for compatibility with JPDoor.
fossilrev	INTEGER	Contains the fossil revision level.
fourdos	STRING[30]	Defaults to ". If 4Dos is detected, then this will contain: '4DOS X.X [Shell X]'
funckey	BOOLEAN	Returns TRUE if the last key pressed was an extended character in any of the keyboard input commands or if an extended hotkey was pressed.
hdr	msghdrrecord	This variable is of type 'msghdrrecord' and is only used by BVDoor in the 'postmsg' function.

hmboverride	BOOLEAN	If your door doesn't use the ExitInfo.BBS fill and you want to use the postmsg function, then you must set this to TRUE.
hotkey	CHAR	If you do not tell BVDoor to kill

		the hotkey after it is pressed, then this is the variable that it will be stored in.
hotkeyon	BOOLEAN	Set to FALSE on program startup. This is set during calls to 'hotkeys_on' and 'hotkeys_off'. See section on Hotkeys.
hotkeypressed	BOOLEAN	Set to FALSE on program startup. If hotkeys is TRUE and a valid hotkey is pressed, then this will be TRUE. See section on Hotkeys.
hr,mn,sec,dum	LONGINT	Used internally for the time routines.
ignorecarrier	BOOLEAN	Defaults to FALSE. If this is TRUE, then any call to 'carrier' will return TRUE.
inactiveval	BYTE	This is the amount of time allowed before the program is terminated due to inactivity. This defaults to 2. If you set it to anything else, set it higher than 1 because BVDoor sends a beep and message to the user when 'activecount' gets to 1.
inusefile	STRING[79]	This variable is used in the 'gameinuse' function. See gameinuse for more information.
killhotkey	BOOLEAN	Toggled TRUE/FALSE by 'sethotkeys_on' procedure by the variable passed to which is either kill or nokill. If a hotkey is pressed and 'kill' is set, then you can not find out which key was hit. If you wish to know what key was pressed then call 'sethotkeys_on' with the 'nokill' parameter. See 'sethotkeys_on' for more information.
lastactivity	INTEGER	This is the time that the last key was entered. This is minutes since midnight.

local	BOOLEAN	Initially set to FALSE. It is set to TRUE if the user has logged on locally. Initialized by the GETDORINFO procedure.

localkey	BOOLEAN	This is TRUE if the lastkey pressed was from the local keyboard. You may want to use this somewhere like to write a splitscreen sysop chat program.
msgbuff	ARRAY[0..100] OF STRING[80]	This is the array which holds the message text for the 'postmsg' function. msgbuff[0] is reserved for internal use.
msgtome	STRING[45]	This holds the fullpath and filename for an online message to the current node. This is only used with RA/QBBS and if 'multinode' is TRUE. IE: 'C:\RA\SEM\NODE1.RA'.
msgwaiting	BOOLEAN	If 'multinode' is TRUE and the current node has a message waiting for them, then this is TRUE, otherwise it returns FALSE. This is updated every 20 seconds when BVDoor checks for an online message.
multinode	BOOLEAN	Default is FALSE. If it is left at FALSE, then ALL of the multinode functions are disabled. They will just exit with nothing happening. If you don't need to use any of the multinode functions, then turn this off to speed your program up just that much more.
nocomcts	BOOLEAN	Set this to TRUE if you wish to ignore the CTS signal for any reason.
node	STRING[3]	Node defaults to '1'. This is used to specify which line number is in use.
nwver	STRING[30]	Defaults to ". If BVDoor detects Novell Network upon program startup, then this will be the following: 'Novell Netware Shell vX.X'. If Novell Netware is detected, then file shareing will

		be used regardless of 'share'.
os2	BOOLEAN	If OS/2 is detected on program startup, then this is set to TRUE and the version number is stored in 'dosver'. See 'dosver'.

portnum	BYTE	This is the comport in use. If the user is local, the the portnum will be 0. If the user is connected to a pipe, then the pipename is stored in the variable 'pipename'.
pipename	STRING	Holds the name of the currently connected pipe.
portstat	ARRAY [0..maxport] OF WORD	Contains the last status word from the last call to the comport/pipe.
productname	STRING[20]	Defaults to 'BVDoor Program'. This can be used to display the name of your program and is display on the statusbar by default.
qbbs275	BOOLEAN	If 'ckexitinfo' is TRUE, then this will be TRUE if the ExitInfo.BBS drop file is in QBBS v2.75 format. BVDoor then uses this to know what structure to use for multinode functions. The drop file is then converted to a RA 2.xx format while it is read in and then written back to this format upon exit.
qbbs276	BOOLEAN	See the variable 'qbbs275'.
qbbs280	BOOLEAN	See the variable 'qbbs275'.
ra110	BOOLEAN	If 'ckexitinfo' is TRUE then this will be TRUE if the ExitInfo.BBS drop file is in RA v1.10 format. BVDoor then uses this to know what structures to use for multinode functions. The drop file is then converted to a RA 2.xx format while it is read in and then written back to this format up exit.
ra200	BOOLEAN	IF 'ckexitinfo' is TRUE then this will be TRUE if the ExitInfo.BBS drop file is in RA 2.xx format. BVDoor then uses this to know what structures to use for multinode functions. No conversion is done.

rip	BOOLEAN	IF RA 2.xx ExitInfo.BBS says that the user has rip, then this is set to TRUE.
super117	BOOLEAN	See the variable 'qbbs275'.

range	charset	Used internally to hold which hotkeys can be pressed. You set this variable like this: range := ['A', 'B', 'C'];
raxit	BOOLEAN	If this is TRUE, then BVDoor will check the 'semaphorepath' directory for a 'RAXIT'nodenum'.eee. The 'nodenum' is the node number and the 'eee' is the error level to exit RA with, NOT THE DOOR! See the procedure 'chkraxit' for more information.
regs	REGISTERS	This is used when calling interrupts and is simply the REGISTERS type defined in the DOS unit. Not implemented by BVDoor.
scrap_entry	STRING[35]	Used by the 'getline' function. This holds the previously wrapped string of text. See the procedure 'getline' for more information.
scrlen	BYTE	Defaults to 24. This is the users screen length and is Initialized when the ExitInfo.BBS file is read. If the ExitInfo.BBS file is not read in or any other weird numbers are read in, then it will stay at the default of 24.
semaforepath	STRING	If RemoteAccess is used then set this to Remote Access's semaphore path so that it can find the online messages. If it isn't set to RA's semaphore path, then you won't receive any online messages.
sessioninfo	BOOLEAN	Defaults to FALSE. Set this to true if you want to read in a Session.Info dropfile.
showwindow	BOOLEAN	Defaults TRUE. If this is set to FALSE, then the blowup window at the start of program execution will not be shown for the 2 second period.

snoop	BOOLEAN	If this is set to FALSE then all output to the local screen is suspended. Not used if the user is logged on locally.

sofar	STRING	This is a temporary variable used by the getline' routines. If you press a key to chat with the user, then all his text will be out of wack. You can this redisplay this variable and pass it back to the getline routine that you called after you are done chatting.
statlinea	ARRAY[1..9] OF statstr	This is the array that holds the text for the top line of the status bar. Use this if you wish to modify any of the status bars.
statlineb	ARRAY[1..9] OF statstr	This is the array that holds the text for the bottom line of the status bar. Use this if you wish to modify any of the status bars.
statusbar	BYTE	This is used to tell which status bar to display. Defaults to 1. If it was set to 6, then statlinea[6] and statlineb[6] would be displayed on the bottom of the screen if the 'statuslineon' variable is TRUE.
statuslineon	BOOLEAN	Defaults to FALSE. If this is set to TRUE, then a two line status bar is drawn on the bottom of the screen.
storedpassword	STRING[15]	Since RA stores it's password in a CRC, the password is not saved anywhere for you to use. This is for compatability with QBBS which still saves it's password in text format. If qbbs275, qbbs276 or qbbs280 is TRUE, then the users password is stored here so you can change it if you want.
sysopfirst	STRING[35]	This is initialized during the call to 'getdorinfo' and reading of the 'DorinfoX.Def' file. If you read the 'Door.Sys' file then this stays at it's default: 'Sysop'.

sysopkey	ARRAY [1..maxkeys] OF STRING[2]	This variable holds the local sysop keys. If one of these keys is pressed during a call to one of the keyboard functions then it's corresponding 'sysoppproc' will be called. See 'Sysop and Userkeys'.
sysopkeyson	BOOLEAN	Must be set to TRUE if you want to be able to trap the sysopkeys. It defaults to FALSE.
sysoplast	STRING[35]	See 'sysopfirst'. This defaults to 'Name'.
sysopname	STRING[35]	This is the sysops full name. sysopfirst + ' ' + sysoplast. If these variables are at their defaults, then 'sysopname' would be: 'Sysop Name'.
sysoppproc	ARRAY [1..maxkeys] OF PROCEDURE	This variable stores the list of sysop far call procedures. If sysopkey[2] was pressed then sysoppproc[2] is called. See 'Sysop and Userkeys'.
systemname	STRING[40]	This is the name of the BBS that is read from the 'DorinfoX.Def' file when 'getdorinfo' is called.
thisnode	STRING[3]	Defaults to '001'. It is used to display the default status bar. It is up to you to change this as all it is here for is for display on the statusbar.
time	STRING[10]	This is the time that your program was loaded up.
timeremaining	INTEGER	This is the users time left in minutes. If you need to know the seconds then see the 'get_time_left' procedure. DO NOT CHANGE THIS YOURSELF. If you do,

it will just go back to what it was before. If you wish to alter the users time, see the procedures on 'Time Limit Routines'

twominutewarning	PROCEDURE	This procedure is called when the use has two minutes left. Use this to do whatever you want when this occurs. See 'everyminutehook' for more information on how to use this.
txtiobuffer	ARRAY [1..4096] OF CHAR	This is a buffer used for textfile reading. You can use this for your own use if you need to.
usehandle	BOOLEAN	Defaults to FALSE. If this is TRUE, then handles/aliases will be shown in the 'useron/useronnode' procedures.
uselock	BOOLEAN	If share is detected, then this will be set to TRUE and file locking will be used. If this is FALSE then all file locking routines will just exit. This is initialized on startup.
usercitystate	STRING	Initialized when the 'DorinfoX.Def' file is readin when 'getdorinfo' is called.
usefirst	STRING[35]	See 'sysopname'. Defaults to 'User'.
userhungup	BOOLEAN	Defaults to FALSE. If TRUE, then If your door is running on a QBBS v2.76+ and your using the ExitInfo.BBS file then the QBBS field is updated on exit. You must set this automatically.
userkey	ARRAY [1..maxkey] OF STRING[2]	This array stores the keys that the user can press to activate a certain procedure. See 'Sysop and Userkeys' for more information.

userlanguage	BYTE	If using RemoteAccess, then this is the users language number.
userlast	STRING[35]	See 'sysoplast'. Defaults to 'Name'.

username	STRING[35]	See 'sysopname'. Defaults to 'User Name'.
userproc	ARRAY [1..maxkey] OF PROCEDURE	This variable stores the list of far call procedure that corresponds to userkey. See 'Sysop and Userkeys' for more information.
usersecurity	WORD	This is initialized from the 'getdorinfo' procedure.
winver	STRING[30]	If Windows is detected on startup, then this contains the version and mode windows is running in. 'Windows 3.xx Std. Mode' or 'Windows 3.xx Enhanced Mode'.

CLEARSCREEN

```
PROCEDURE clearscreen;
```

This clears the remote and local screen. Use this command instead of CLRSCR because CLRSCR will not clear the remote screen and it will not scroll the local correctly if the statusbar is on.

Example:

```
BEGIN
  getdorinfo('1', 'C:\Ra');
  clearscreen;
END;
```

CRLF

```
PROCEDURE crlf;
```

Writes a carriage return and line feed to the remote and local screen. Can be very useful after a call to SDISPLAY.

Example:

```
BEGIN
  getdorinfo('1', 'C:\Ra');
  crlf;
END;
```

CLRLN

```
PROCEDURE clrln;
```

Clears the current line and moves the cursor to the beginning of the line.

Example:

```
BEGIN
        getdorinfo('1', 'C:\Ra');   sdisplay(0, 7, 0,
        'Once you hit   any key, this line text will disappear!');
  clrln;
END;
```

DISPLAY

PROCEDURE display(bg, fg, blink : BYTE; s : STRING);

Writes 's' to the local and remote screens with a cr/lf. Equivalent to the WRITELN command. Again do not use the WRITELN command as it will not scroll the local screen correctly and will not write 's' to the remote screen. 'bg' is the background color 's' will be displayed on. 'fg' is the foreground. 's' will
blink on the screens if 0 is not passed to it.
Valid 'bg' colors are:

0 : black	4: red
1 : blue	5: magenta
2 : green	6: brown
3 : cyan	7: grey

Valid 'bg' colors are:

0 : black	8 : dark grey
1 : blue	9 : light blue
2 : green	10 : light green
3 : cyan	11 : light cyan
4 : red	12 : light red
5 : magenta	13 : light magenta
6 : brown	14 : yellow
7 : grey	15 : white

Example:
display(1, 10, 0, 'Hello World');
The above would display 'Hello World' on a blue background in light green text.

display(0, 14, 1, 'Your ansi mode is: ' + itoa(ansi))
The above would display 'Your ansi mode is: 1' in flashing yellow.

See Also: SDISPLAY, CHAROUT, ITOA, WTOB, WTOH

DISPLAYLOC

PROCEDURE displayloc(bg, fg, blink : BYTE; s : STRING);

Writes 's' to the local screen ONLY without a cr/lf. Equivalent to the WRITE command. Do not use the WRITE command as it will not scroll the local screen correctly. 'bg' is the background color 's' will be displayed on. 'fg' is the foreground. 's' will blink on the screens if 0 is not passed to it.

Example:
displayloc(1, 10, 0, 'Hello World');
Would display 'Hello World' on a blue background in green text on the sysop's screen only!

displayloc(0, 14, 1, 'Your ansi mode is: ' + itoa(ansi))
Would display 'Your ansi mode is: 1' in flashing yellow on the sysop's screen only!

See Also: DISPLAY, SDISPLAY, CHAROUT, ITOA, WTOB, WTOH

DRAWCARD

```
PROCEDURE drawcard(cardnum : STR2; suit : cardsuit; row, col,
                   cardcolor : BYTE; edge : BOOLEAN);
```

Draws ANSI graphic playing cards within your door. Each card is 3 rows high by 5 columns wide. Black cards are displayed black on grey, red cards are displayed red on grey. 'cardcolor'allows you to draw different colors of blank cars. If 'cardcolor' is 0, then it will default to 7. This procedure will exit if the variable 'ansi' is set to 0. The parameters for drawcard are:

```
'cardnum'   is '1' through '10','J','Q','K','A' OR 'B' for          a blank card.
'cardsuit'  is C, D, H or S (for the clubs, diamonds, hearts        spades).
'row'       is the row where the card is to be drawn.
'col'       is the column where the card is to be drawn.
'cardcolor' the background color of the blank cards.
'edge'      if true, a border will be drawn around the card.
```

Example:

```
BEGIN
  getdorinfo('1', 'C:\Ra');
  sdisplay(0, 7, 0, 'Hit a key;the line text will disappear!');
  IF ansi = 0 THEN
    BEGIN
      display(0, 10, 0, 'Sorry, but you need ansi graphics'
                                              + 'for this game.');
      sdisplay(0, 14, 0, '**Pause**');
      HALT(0);
    END;
  clearscreen;
  drawcard('A', S, 2, 2, 7, TRUE);
END;
```

This example checks to see if the user has ansi graphics. If not, then the door kicks the user out. If the user is capable of ansi graphics, then an ace of spades is drawn on line 2, col 2 with a border.

GAMESCREEN

```
PROCEDURE gamescreen(filename : STRING);
```

GAMESCREEN is a faster method of displaying an ANSI graphic screen (such as a full screen game board, etc.) than SHOWFILE. SHOWFILE works the same but is a bit slower since it checks for sysop/hot keys and a few other routines. GAMESCREEN does not check for ANYTHING. The screen is cleared before the file is shown and page pausing is NOT in effect. If 'filename' does not exist then the procedure just exists. If 'bvdebug' is true and 'filename' doesn't exist, then an error message will be written to the debug log file.

See Also: ASC, SHOWFILE

SDISPLAY

PROCEDURE sdisplay(bg, fg, blink : BYTE; s : STRING);

Writes 's' to the local and remote screens WITHOUT a cr/lf. Equivalent to the WRITE command. Again do not use the WRITE command as it will not scroll the local screen correctly. 'bg' is the background color 's' will be displayed on. 'fg' is the foreground. 's' will blink on the screens if 0 is not passed to it.

See Also: DISPLAY, CHAROUT, ITOA, WTOB, WTOH

SETCOLOR

PROCEDURE setcolor(bg, fg, blink : BYTE);

Sets the current color for outputing text. This procedure is called by internal functions (ie DISPLAY, SDISPLAY), but you may need to use it some of the other functions like: CHAROUT, BACKSPACE. 'bg' is the background color, 'fg' is the foreground color of the text and if blink is not 0, the text will blink.

See Also: DISPLAY

SETGRAPHICS

FUNCTION setgraphics : CHAR;

This function looks at the users ansi setting 'ansi' and returns:
'A' ... if 'ansi' is 0 (Ascii)
'C' ... if 'ansi' is 1 (Ansi color)
'M' ... if 'ansi' is 5 (Monochrome Graphics)

Example:

```
BEGIN
  getdorinfo('1', '');
  IF setgraphics = 'A' THEN
    BEGIN
      display(0, 1, 0, 'Sorry, but you need ansi graphics to' +          ' play this game');
      HALT(0);
    END;
END.
```

SHOWFILE

PROCEDURE showfile(initcolor, ptype, pcolor : BYTE;
filename, path : STRING);

Displays an ansi/ascii/text file to the local and remote screen. If the filename has no extension,

then 'ansi' variable and use either 'ANS', 'ASC'. If the 'ansi' is 1 or 5 then it will use filename.ANS. If this file doesn't exist it will display filename.ASC. Showfile exits if the file cannot be found and will log it using bv_log.

SHOWFILE parameters:
'initcolor' is the initial drawing color if it is not an ansi file.
'ptype' is the prompt type Either 1, 2, 3, 4.
'pcolor' is the foreground color the prompt will be displayed in.
'filename' is the file to be displayed.
'path' is the path where 'filename' exists.

Page pausing with
Type 1 = '==PAUSE=='
Type 2 = 'More (Y/n)'
Type 3 = 'More (Y/n/=)'
Type 4 = 'Hit [Enter] to continue'

Example:

showfile(0, 0, 0, 'Welcome.Ans', 'C:\Ra\TxtFiles');
The above would display the welcome file in the RA textfile directory without any pausing and would start drawing with the current foreground color.

showfile(0, 2, 14, 'Welcome.Ans', 'C:\Ra\TxtFiles');
The above would display the welcome file in the RA textfile directory with the prompt type 2 (More (Y/n)) at every 'scrlen' with the color yellow.

Keyboard Input Routines

ASK_YN

FUNCTION ask_yn(prompt : STRING; fg : BYTE) : CHAR;

Displays 'prompt' to the screen and adds '(Y/N): ' to the end of 'prompt' and waits for a user to press 'Y' or 'N'.

Example:

```
REPEAT
  ch := ask_yn('Keep answering this question?', 14);
UNTIL (ch = 'N');
```

CHARIN

FUNCTION charin : CHAR;

Returns the next character in the input queue. If the input queue is empty, then a null character will be returned (#0).

See Also: GETCHAR

GETCHAR

```
FUNCTION getchar : CHAR;
```

Returns the next character in the input queue. If the input queue is empty, then GETCHAR will wait until a character is read in.

See Also: CHARIN

CHAROUT

```
PROCEDURE charout(ch : CHAR);
```

Sends a character to the output buffer. It does not display it to the local screen.

See Also: CHARIN

GETBIRTHDATE

```
FUNCTION getbirhdate : STRING;
```

Gets a valid birthday from the remote/local user. It will be returned in the format of 'MM-DD-YY'.

GETINTEGER

```
FUNCTION getinteger(VAR int : LONGINT; min, max : LONGINT;
                    usebefore : BOOLEAN);
```

Inputs a number from user and waits for them to press enter. The number inputed will be inputed only between 'min' and 'max'. If 'usebefore' is true, then GETINTEGER will display:

'New Info (0-10): 20'

Example:

```
BEGIN
  int := 5;
  display(0, 7, 0, 'Enter a number 1-10: ');
  getinteger(int, 1, 10, FALSE);
END
```

The above would only accept a number between 1 and 10.

```
BEGIN
  int := 5;
  display(0, 7, 0, 'Enter a number 1-10: ');
  getinteger(int, 1, 10, TRUE);
END
```

The above would look like the first example but the screen displays would look like this. 5 is the default because that is what 'int' was set to.

'Enter a number 1-10:'

'New Info (1-10): 5'

GETLINE

FUNCTION getline(len, tablen, color : BYTE);

Gets a line of input from the user with word wrapping. If the user hits the tab key, it is expanded to 'tablen'. Word wrapping occurs at 'len' and the text inputed is displayed in color 'color'. The variable 'scrap_entry' must be initialized before a call to GETLINE. 'scrap_entry' is the previously wrapped string from GETLINE and is displayed before input is accepted in the GETLINE function.

Example:

```
display(0, 7, 0, 'Type in 5 lines:');
scrap_entry := '';
FOR counter := 1 TO 5 DO
  msg[counter] := getline(75, 4, 10);
FOR counter := 1 TO 5 DO
  display(0, 12, 0, msg[counter]);
```

GETPHONE

FUNCTION getphone(style : BYTE) : STRING;

Inputs a USA or free-style format phone number including area code. This function automatically formats the user input field depending on the 'style' used. Valid 'style's are:

0 : USA Format (Area Code) Prefix-Suffix. The formatted result returned is: ###-###-####
1 : Free-Style - Stores input number as entered by the user.
Allows input of up to 18 characters including digits and dashes ('-').

INREADY

FUNCTION inready : BOOLEAN;

Returns true if a key is in the input buffer. Otherwise false is returned.

PROMPT

FUNCTION prompt(fore : BYTE; ptype : BYTE);

Displays one of the following prompt strings in color 'fore'.
'ptype' is one of the following:

Type 1 = '==PAUSE=='
Type 2 = 'More (Y/n)'
Type 3 = 'More (Y/n/=)'
Type 4 = 'Hit [Enter] to continue'

SETHOTKEYS_OFF;

PROCEDURE sethotkeys_off;

Turns off hotkey usage. The following variables are initiailized upon a call to SETHOTKEYS_OFF.

```
hotkeyon := FALSE;
hotkeypressed := FALSE;
range := [#0];
```

See Also: SETHOTKEYS_ON

SETHOTKEYS_ON;

PROCEDURE sethotkeys_on(killkey : killer ; range : charset);

'range' is a set of characters which are valid hot-keys. These should be specified in upper case format. If any of the specified characters are pressed, all display output is stopped (nothing is echoed to the either the local or remote screens), and the 'hotkeypressed' variable is set to TRUE. Once this call is made, the 'hotkeyon' variable is also set to TRUE. If 'kill' is specified in killkey, then the hot-key pressed is killed from
memory. That is to say that it is not "remembered". If 'nokill' was specified, the Hot-Key is "remembered" and will be automatically returned to the next call to GETCHAR. This is intended for uses such as breaking out of a text file display.

Example:

```
BEGIN
        sethotkeys_on(kill, ['A', 'S']);
        showfile(7, 0, 0, 'Disclamr', 'C:\Ra\TxtFile);
  sethotkeys_off;
END;
```

The above example would display a BBS disclaimer and set the hotkeys to 'A'bort and 'S'top. If the user pressed those keys during the display of the text file it would exit the showfile procedure and continue in the calling part of the program.

BACKSPACE

PROCEDURE backspace(ch : CHAR);

Writes a destructive backspace to the output buffer leaving 'ch' behind. If you would want to remove a character from the screen and leave nothing behind, you would pass a null character to 'ch' or a ' '. The current screen colors are used to display 'ch'.

Example:
```
sdisplay(0, 7, 0, 'Now you see this');
getchar;
FOR counter ;= 1 TO LENGTH('this') DO
  backspace(' ');
sdisplay(0, 7, 0, 'something different!');
```

CENTER

FUNCTION center(s : STRING; len : WORD) : STRING;

Centers 's' in the middle of 'len'. 's' is padded with ' ' to the beginning of it. No padding is done to the end of 's'.

See Also: CENTERFILL

CENTERFILL

FUNCTION centerfill(s : STRING; len : WORD) : STRING;

Centers 's' in the middle of 'len' which will normally be the users screen width. 's' is padded with ' ' to the beginning of it. Padding is done to the end of 's' with ' ' to make it a length of 'len'. This function is perfect for making a title bar.

Example:
```
clearscreen;
display(1, 14, 0, centerfill('Welcome to BVDoor', 79));
```
This example would clear the screen and display 'Welcome to BVDoor' in the middle of the screen on a blue background along the whole line.

CHANGECASE

FUNCTION changecase(s : STRING) : STRING;

Returns a string in upper case.

See Also: FORMATSTR

CKPATH

FUNCTION ck_path(path : STRING) : STRING;

Checks to see if a pathname ends with a trailing backslash. If no backslash is present it is added, otherwise the path remains unchanged. This does not check to see if the path exists like FIXPATH does.

See Also: EXISTPATH, FIXPATH

COMMASTR

FUNCTION commastr(int : LONGINT) : STRING;

Adds commas into a number to make it more readable.

Example:
```
display(0, 7, 0, 'Did you know that ' + commastr(987654321) +
                 ' is a large number');
```
Output would look like:
Did you know that 987,654,321 is a large number

CURSORPOS

PROCEDURE cursorpos(y, x : BYTE);

Moves the cursor position to row y, column x. Exists if the y is less than 0 or greater than 25 or if x is less than 0 or is greater than 80.

DATESTR

FUNCTION datestr : STRING;

Returns the current date as MM-DD-YY.

Example:
```
display(0,15,0,'The Current Date Is ' + datestr);
```

FIXPATH

FUNCTION fixpath(path : STRING) : STRING;

Calls the CKPATH function and then checks to see if the path exists. If the path does not exist a haltcode of 6 is returned.

FORMATSTR

FUNCTION formatstr(s : STRING) : STRING;

Shifts all the first letters of a word to uppercase and the rest of the word to lowercase.

Example:
s := formatstr('tHis iS aN eXamPLE);
The result would be:
'This Is An Example'

GETWORDS

FUNCTION getwords(numwords : INTEGER; s : STRING) : STRING;

Returns the first 'numwords' from 's'. Words must be separated by spaces. If the number of words is less than 'numwords' then the whole string is returned untouched.

Example:
s := 'This contains only a couple words'
s2 := getwords(3, s);

The result of 's2' would be:
'This contains only'

LJUST

FUNCTION ljust(s : STRING; wide : BYTE) : STRING;

Left justifies 's' to a length of 'wide'. Padding is done with adding ' ' to the end of 's' to make it 'wide' long.

Example:
s := 'Hello';
display(0, 7, 0, ljust(s, 10));

Output would be:
'Hello '

See Also: RJUST

LASTPOS

FUNCTION lastpos(sub : STRING; s : STRING) : STRING;

Exaclty the same as the POS command, but returns the position of the last found substring 'sub' in the string 's'.

Example:

```
 s := 'Hello hi, no hi, no Hello, hi? Huh??');
 display(0, 7, 0, 'Last pos of HI is:', itoa(lastpos('hi', s)));
```

Output would be:

```
 'The last pos of HI is: 28'
```

LEFTS

FUNCTION lefts(s : STRING; charstokeep : INTEGER) : STRING;

Returns the first 'charstokeep' from string 's'.

Example:
display(0, 7, 0, lefts('Welcome to BVDoor', 6));

Output would be:
'Welcom'

See Also: RIGHTS

LTRIM

FUNCTION ltrim(s : STRING) : STRING;

Removes the leading ' ' from 's';

Example:
display(0, 7, 0, ltrim(' Hello '));

Output would be:
'Hello '

See Also: RTRIM

LOWCASE

FUNCTION lowcase(ch : CHAR) : CHAR;

Converts a character to lower case. Does the oppostite of the UPCASE command.

See Also: CHANGECASE

MKSTRING

FUNCTION mkstring(long : INTEGER; ch : CHAR) : STRING;

Makes a string of 'ch' of length 'long'. Used for drawing underlines and other things.

Example:
display(0, 7, 0, mkstring(10, -));

Output would be:
'----------'

MORE

PROCEDURE more(prompt : STRING; fg : BYTE);

Displays 'prompt' to the screen jusing a foreground color of 'fg'. This procedure also waits for a key to be pressed.

RIGHTS

FUNCTION rights(s : STRING; charstokeep : INTEGER) : STRING;

Returns the first 'charstokeep' from string the end of 's'.

Example:
```
s:= 'Welcome to BVDoor';
display(0, 7, 0, rights(s, 5));
```

Output would be:
```
 'VDoor'
```

See Also: LEFTS

RTRIM

FUNCTION rtrim(s : STRING) : STRING;

Removes the trailing ' ' from 's';

Example:
```
s := '   Hello  '
display(0, 7, 0, rtrim(s));
```

Output would be:
```
'   Hello'
```

See Also: LTRIM

TIMESTR

FUNCTION timestr : STRING;

Returns the current time as HH:MM:SS in 24 hour format.

Example:
```
display(0, 15, 0, 'The Current Time Is ' + timestr);
```

Number/String Conversions

ATOI

FUNCTION atoi(s : STRING) : INTEGER;

Converts a string into an integer.

BTOA

```
FUNCTION btoa(bool : BOOLEAN) : STRING;
```

Returns 'TRUE' or 'FALSE' as a string depending on the state of the boolean passed to it.

Example:

```
VAR
  bool : BOOLEAN;
BEGIN
  display(0, 7, 0, 'bool is true' + btoa(bool));
END;
```

ITOA

```
FUNCTION itoa(int : INTEGER) : STRING;
```

Converts an integer to a string. Very convienent to use with the display commands.

Example:

```
display(0, 7, 0, 'Your ansi setting is: ' + itoa(ansi));
```

WTOA

```
FUNCTION wtoa(num : INTEGER) : STRING;
```

Converts a variable of type WORD into a string.

See Also: ITOA

WTOH

```
FUNCTION wtoh(num : WORD) : STRING;
```

Converts a number into it's hex form.

Example:

```
display(0, 7, 0, 'Hex of 45 is: ' + wtoh(45));
```

The above would display: 'Hex of 45 is: 002D'

File Routines

EXIST

FUNCTION exist(filename : STRING) : BOOLEAN;

Returns true if a file on harddrive exists. Pathnames included with a filename are valid.

See Also: CKPATH, FIXPATH

EXISTPATH

FUNCTION existpath(d : STRING) : BOOLEAN;

Tests to see if a fixed path exists. If the path doesn't exist, a return code of FALSE is returned, otherwise TRUE is returned.

DELETEFILE

PROCEDURE deletefile(s : STRING);

Deletes a specified file. If the file does not exist, no error is returned.

SIZEFILE

FUNCTION sizefile(s : STRING) : LONGINT;

Returns the total size of the file in bytes. If the file does not exist, a filesize of 0 is returned. Remember, that files can exist with only 0 bytes so you may want to use the EXIST command on it first.

See Also: EXIST

Device (Modem/Pipe..) Routines

CARRIER

FUNCTION carrier : BOOLEAN;

Checks to see if there is a carrier. This function always returns true if LOCAL is set to TRUE or if the variable IGNORECARRIER is TRUE.

See Also: IGNORECARRIER, LOCAL

FLUSHCOMMBUFFER

PROCEDURE flushcommbuffer;

This flushes out the input buffer so it is empty. You may want to use this before input routines so the user can not type ahead.

OUTPUTBUFEMPTY

FUNCTION outputbufempty : BOOLEAN;

This checks to see if there is anything in the output buffer. If there is then this function will return FALSE, otherwise it will return TRUE.

Time Limit Routines

GET_TIME_LEFT

PROCEDURE get_time_left(VAR min, sec);

This returns how much time left the user has in minutes and seconds.

DECREASETIME

PROCEDURE decreasetime;

Decreases the variable 'timeremaining' by 1 and checks to see if it is 0. If 'timeremaining' is 0 then CHKCONNECT will catch this and terminate the program with a HALT(2). CHKCONNECT is called from inside this procedure.

INCREASETIME

PROCEDURE increasetime;

Increases the variable 'timeremaining' by 1 and calls CHKCONNECT.

Multi-Node Routines

CHKMSG

FUNCTION chkmsg : BOOLEAN;

If CKMSGS, MULTINODE, and either QBBS275, QBBS276, RA110, or RA200 is TRUE then BVDoor calls this function once a minute. If a message is waiting for the node that your door is running from, the variable 'MSGWAITING' is set to TRUE. If you are running Remote Access, you must have the 'SEMAPHOREPATH' variable set to RA's semaphore path or the current directory is scanned and you won't be able to retrieve online messages.

GETMSG

PROCEDURE getmsg;

If an online message is waiting for the current node, then the message will be displayed.

See Also: CHKMSG, WAITINGMSG

GAMEINUSE

FUNCTION gameinuse : BOOLEAN;

This function checks to see if 'INUSEFILE' isn't blank. If 'INUSEFILE' is blank this function exists with FALSE. If 'INUSEFILE' isn't blank and EXIST(inusefile) returns FALSE, then a filename of 'INUSEFILE' is created and the function returns FALSE. If 'INUSEFILE' isn't blank and EXIST(inusefile) returns TRUE, then it means that someone else is using the door. BVDoor then displays a message to the user saying that the door is in use and returns TRUE.

Example:

```
BEGIN
  inusefile := 'C:\Onlines\MyDoor\Inuse.Flg';
  IF gameinuse THEN
    HALT(0);
END
```

If gameinuse returns TRUE, the program will continue. If it returns false, then the following message will be displayed:

productname ' is being use by another line.' Please try again later...'
Hit ENTER to continue

See Also: GAMENOTINUSE

GAMENOTINUSE

PROCEDURE gamenotinuse;

If 'inusefile' isn't blank and EXIST(inusefile) is true, then 'inusefile' will be deleted and other users will be able to load up the door. YOU MUST call this procedure if you call GAMEINUSE, otherwise nobody will be able to load up the door until 'inusefile' is deleted!

MAXNODES

FUNCTION maxnodes : BYTE;

This function scans the UserOn.BBS file in the 'exitinfopath' and returns the maximum amount of nodes your BBS has.

USERONNODE

FUNCTION useronnode(nodenum : BYTE) : namestr;

This function scans the UserOn.BBS file in the 'exitinfopath' and returns the name of the user on node 'nodenum'. If no one is on the node, then the function will return ''.

WHOSON

PROCEDURE whoson;

If MULTINODE, CKEXITINFO and either of RA200, RA110, QBBS275, QBBS276, QBBS280 are true, then this procedure will display a Remote Access style of the who's online screen.

User Record Routines

NOTE ** The EXITINFO.BBS File MUST be read in for these to work UNLESS the 'SESSION.INFO' file was read in **

FLIPFLAG

PROCEDURE flipflag(flag : flagset);

Toggles the flag passed to the procedure and updates the exitinfo.bbs file.

Example:

```
BEGIN
  ...
  flipflag(A1);
END;
```

If the users A1 flag was off before the call, it would then be on after the call to flipflag.

SETFLAG

PROCEDURE setflag(flag : flagset; stat : on_off);

Changes the state of 'flag' to the passed variable 'stat' which will be 'ison' or 'isoff'. This updates the exitinfo.bbs file.

See Also: FLIPFLAG

SETSECURITY

PROCEDURE setsecurity(sec : WORD);

Changes the users security level in the exitinfo.bbs file to 'sec'.

SETSUBDATE

PROCEDURE setsubdate(sub : ra_date);

Sets the users subscription date in the exitinfo.bbs file to 'sub'.

CHKRAXIT

FUNCTION chkraxit : BOOLEAN;

Checks to see if the sysop has forced the node down with the semaphore file RAXIT'nodenum'.???. If the file exists, then CHKRAXIT returns TRUE.

POSTMSG

PROCEDURE postmsg(long : BYTE; msgpath : STRING);

This function exists if both of the variables RA200 AND RA110 are FALSE. This function writes a message in the RA message base. The actual message text is you want to write is stored in the varible MSGBUFF which is a global array (msgbuff : ARRAY[1.100] OF STRING[80]).

Fill 'msgbuff' with the message starting with msgbuff[1] as msgbuff[0] is used internally to add a product ID kludge line in the message. You MUST make sure to add a character 13 (Carriage Return) to the end of each line, include blank lines. The passed variable of 'long' is the number of lines long your message is. If the last line of your message is msgbuff[6] then pass 6 to the postmsg call. 'msgpath' is the path to the RA message base. If successful, postmsg will return a 0, if not, -100

'postmsg' will determine the correct message number for the system, but you are required to fill out the remaining part of the structure.

```
msghdrrecord = RECORD
      destnet,
      destnode,
      orignet,
      orignode : WORD;
      destzone,
      origzone : BYTE;
      cost : WORD;
      msgattr,
      netattr,
      board : BYTE;
      posttime : ra_time;
      postdate : ra_date;
      whoto,
      whofrom : namestr;
      subject : String[72];
   END;
```

The global variable that is assigned to this record is:

```
 hdr : msghdrrecord;
```

Example:

```
PROCEDURE post_sysop_msg;
 BEGIN
   clearscreen;
   Display(0,15,0,'Writing a Message to the Sysop!');
   WITH hdr DO
    BEGIN
      whoto     := formatstr('Sysop');
      whofrom   := username;
      subject   := 'BBS Door';
      board     := 1;       (* Sysop Area *)
      postdate  := datestr;
      posttime  := timestr;
      destnet   := 0;
      destnode  := 0;
      orignet   := 0;
      orignode  := 0;
      destzone  := 0;
      origzone  := 0;
      cost      := 0;
      msgattr   := msg_local;
      netattr   := 0;
    END;
   msgbuff[1] := 'Dear Sysop Person'#13;
   msgbuff[2] := '' + #13;
   msgbuff[3] := ' I was just writing you a message to see how'
         + ' it works'#13;
   msgbuff[4] := ' when you write it from inside a BVDoor' + ' +                              '
program'#13;
   msgbuff[5] := ' It isn't hard at all. :)'#13;
   msgbuff[6] := ''#13
   msgbuff[7] := '   Regards'+ #13;
   msgbuff[8] := '     ' + username + #13;
   crlf;
   display(0, 14, 0,'Posting message');
   postmsg(8, 'C:\Ra\MsgBase');
 END;
```

TIMETOEVENT

```
 FUNCTION timetoevent : INTEGER;
```

This returns how many minutes until the next system event. This procedure reads in the events.ra file in the 'exitinfopath' directory and then calculates how many minutes there are until the next event. If the the event data file can't be found, then this function returns a -1.

DOCONVERT

PROCEDURE doconvert;

This is the procedure that reads in the ExitInfo.BBS files. This procedure checks to see what format of ExitInfo.BBS and converts it to the RA 2.xx format if it is not already. If the ExitInfo.BBS file was converted, then the variable 'exitconverted% is set to TRUE. Upon exit of the program, the files are rewritten back into the original form with any updated information from you door program. The following variables are set when this procedure is called from GETDORINFO:
ra200, ra110, qbbs275, qbbs276, qbbs280, adept175.

See Also: GETDORINFO

GETDORINFO

PROCEDURE getdorinfo(nodenum : STRING; exitpath : STRING);

This is probably the most important procedure in the whole package. This procedure reads in the drop files and initializes all the system and user variables. If 'forcenode' is true (which is default), then GETDORINFO will read in the 'DORINFO1.DEF' only; no acceptions. If you set the variable 'forcenode' to false, then GETDORINFO will read in the 'DORINFOx.DEF where 'x' is the nodenum passed. If a 'DORINFO' file is read in, then the ExitInfo.BBS file is automatically read in unless the variable 'ckexitinfo' is false. IF 'ckexitinfo' is true, the 'DORINFO' file was readin and the ExitInfo.BBS was not found, then a haltcode of 5 is called. If you wish to read in a Door.Sys file, then set the variable 'doorsys' to TRUE BEFORE you call GETDORINFO. If you wish to read in a 'Session.Info' file (AdeptXBBS) then set the variable 'sessioninfo' to TRUE BEFORE you call GETDORINFO. If you read in the 'Door.Sys' file, ckexitinfo is true and the ExitInfo.BBS file is not found, then no error is created as some BBS packages only create a Door.Sys file. If the ExitInfo.BBS file is found then you just have more information you can change. No harm done!

Example:
```
BEGIN
  node := '000005';
  getdorinfo(node, 'C:\Ra');
END.
```
The above is a standard call to GETDORINFO. It will only read in Dorinfo1.Def as 'forcenode' is TRUE by default.

Example:
```
BEGIN
  forcenode := FALSE;
  node := '000005';
  getdorinfo(node, 'C:\Ra');
END.
```
The above is a standard call to GETDORINFO. It will read in Dorinfo5.Def because 'forcenode' is false.

```
VAR
  node : STRING;
BEGIN
  node := '1';
  doorsys := TRUE;
  getdorinfo(node, 'C:\Ra\Node1');
END.
```
The above reads in the 'Door.Sys' file from the '\Ra\Node1' path because the variable 'doorsys' is set to TRUE.

```
VAR
  node : STRING;
BEGIN
  sessioninfo := TRUE;
  node := '000001';
  getdorinfo(node, 'C:\Ra');
END.
```

The above reads in the 'Session.Info' drop file.

BV_LOG

```
PROCEDURE bv_log(debug_txt : STRING);
```

This procedure writes 'debug_txt' into a file named: DEBUG'thisnode'.LOG if the variable 'bvdebug' is TRUE. If 'bvdebug' is false, then the procedure exits and nothing is written into the logfile.

GETNODE

```
FUNCTION getnode : BYTE;
```

This function takes a look at the variable 'thisnode' and returns it back as a BYTE.

Example:
```
BEGIN
  FOR counter := 1 TO getnode DO
    display(0, 14, 0, 'User on node ' + itoa(counter) + ':' +
                    useronnode(counter));
END.
```

SHOW_STATUS

```
PROCEDURE show_status;
```

If the variable 'statuslineon' is true, then the status line will be forced to redraw itself.

DESQVIEW_ACTIVE

PROCEDURE destview_active;

This function takes a look at the operating system and updates the following variables on it's findings:

os2
dosver
dv
dvver
fourdos
nwver
winver

Only For Compatablity

The following routines are not implemented because of porting problems. If you really need one of these routines implemented, then please contact us at BV Compuworks Group.

```
FUNCTION  unlock(VAR df : FILE; startblock,
                                lenblock : LONGINT) : WORD;
FUNCTION  lock(VAR df : FILE; startblock,
                                lenblock : LONGINT) : WORD;
PROCEDURE set_retries(lockretries, delay : INTEGER);
FUNCTION  setfossil : BOOLEAN;
FUNCTION  setflaguser(path : pathstr;
                        user : namestr; onoff : on_off) : BYTE;
PROCEDURE showfilesbbs(VAR lncount : INTEGER; n, pause,
                                     fc,sc,dc, ec, mc,bc : BYTE;
                                     fpath, fn : STRING;
                                     VAR cont: BOOLEAN;
                                     oneline : BOOLEAN);
FUNCTION  share : BOOLEAN;
PROCEDURE deposittime(path, name : STRING; depnum : INTEGER);
PROCEDURE clearregs;
```

BVDoor CONST Section:

```
maxport    = 255;        { Max Com Ports Supported  }
bvdversion = '1.0';  { BVDoor Version            }
compiled   = 'Compiled on thisday';
                                  { Date BVDoor was compiled    }
beta       = FALSE;  { True in Beta versions     }
ansicls    =  #27 + '[40m' + #27 + '[2J';
                                   { Ansi Codes to clear screen  }
maxkeys    =  20;        { Max User keys supported    }
lockmde    =  64 + 2;
                                  { File Locking opens files as }
                                  { Shared/Read/Write           }
softcr     = #141;       { Use this in messages for CR }
cr         = #13;        { Carriage Return                     }
```

```
(* The Following are Used in PostMsg attributes *)
msg_delete  = 1; { (Attr AND          1) =      msg deleted }
msg_out_net = 2; { (Attr AND          2) =      netmail to be scanned out}
msg_net     = 4; { (Attr AND  4) =      Netmail Message }
msg_private  =  8; { (Attr AND 8) = Private Message }
msg_received = 16; { (Attr AND 16) = Message Received}
msg_out_echo = 32; { (Attr AND 32) = Echomail to be scanned out }
msg_local = 64; { (Attr AND 64) =    ALWAYS SET msg typed locally}
```

BVDoor TYPE Section:

```
pathstr = STRING[79];
statstr  = STRING[79];
str18    = STRING[18];
str5             = STRING[5];
str2             = STRING[2];
str1             = STRING[1];
namestr          = STRING[35];
charset          = SET OF CHAR;
flagset = (A0, A1, A2, A3, A4, A5, A6, A7, A8,
                        B1, B2, B3, B4, B5, B6, B7, B8,
                        C1, C2, C3, C4, C5, C6, C7, C8,
                        D1, D2, D3, D4, D5, D6, D7, D8);
on_off = (null, ison, off);
killer   = (kill, nokill);
cardsuit         = (C,D,H,S);

asktype                  = (yes, no, ask, only);
videotype                = (auto, short, long);
msgkindstype  = (both, private, public, ronly);

msgtxt = RECORD
         line : STRING[255];
 END;

msgtoidx = RECORD
```

```
        user : STRING[35];
 END;

msgtype                 = (localmail, netmail, echomail);
orphantype      = (ignore, create, ra_kill);
flagtype                = ARRAY[1..4] OF BYTE;
ra_time                 = STRING[8];
```

```
ra_date                 = STRING[8];
longdate                = STRING[9];
netaddress = RECORD
        zone,
        net,
        node,
        point : WORD;
 END;


languagerecord = RECORD
        name            : STRING[20];
        attribute : BYTE;
        defname,
        menupath,
        textpath,
        quespath        : STRING[60];
        freespace : ARRAY[1..200] OF BYTE;
 END;

msginforecord = RECORD
        lowmsg,
        highmsg,
        totalmsgs    : WORD;
        totalonboard : ARRAY[1..200] OF WORD;
 END;

msgidxrecord = RECORD
        msgnum : INTEGER;
        board  : BYTE;
 END;

msgtoidxrecord = STRING[35];

msghdrrecord = RECORD
        msgnum          : INTEGER;
        prevreply,
        nextreply,
        timesread  : WORD;
        startblock : WORD;
        numblocks,
        destnet,
        destnode,
        orignet,
        orignode        : WORD;
        destzone,
        origzone   : BYTE;
        cost            : WORD;
        msgattr,
        netattr,
```

```
        board    : BYTE;
        posttime         : ra_time;
        postdate         : ra_date;
        whoto,
        whofrom          : msgtoidxrecord;
        subject          : STRING[72];
END;
```

```
        msgtxtrecord = STRING[255];

        (* RemoteAccess v1.1x UserOn.BBS *)
        useron110record = RECORD
                name            : msgtoidxrecord;
                handle    : msgtoidxrecord;
                line            : BYTE;
                baud            : WORD;
                city            : STRING[25];
                donotdisturb : BOOLEAN;
                status      : BYTE;
                attribute    : BYTE;
         END;

        (* RemoteAccess 1.00 UserOn.BBS *)
        useron100record = RECORD
                name            : msgtoidxrecord;
                line            : BYTE;
                baud            : WORD;
                city            : STRING[25];
                donotdisturb : BOOLEAN;
                status       : BYTE;
         END;

(*
  Remote Access Status Byte -
        0 : Browsing the BBS (In a Menu)
        1 : Uploading/Downloading
        2 : Reading/posting messages
        3 : In a door/external program
        4 : Answering questionnaire
*)


        (* QBBS 2.75 UserOn.BBS *)
        q_useronrecord = RECORD
                name            : STRING[35];
                line            : BYTE;
                baud            : WORD;
                city            : STRING[35];
                donotdisturb : BOOLEAN;
         END;

        (* Remote Access v1.10 LastCall.BBS *)
        lastcall110 = RECORD
                line            : BYTE;
                name            : msgtoidxrecord;
                handle : msgtoidxrecord;
                city            : STRING[25];
                baud            : WORD;
```

```
        times    : LONGINT;
        logon    : STRING[5];
        logoff   : STRING[5];
        attribute : BYTE;                (* Byte 0 : Hidden *)
END;
```

```
(* RemoteAccess v1.00 /QBBS Lastcall.BBS *)
lastcallrecord = RECORD
        line       : BYTE;
        name       : msgtoidxrecord;
        city       : STRING[25];
        baud       : WORD;
        times  : LONGINT;
        logon  : STRING[5];
        logoff : STRING[5];
 END;


combinedrecord = ARRAY[1..200] OF WORD;

usersidxrecord = RECORD
        namecrc32,
        handlecrc32     : LONGINT;
 END;

uline = RECORD
        name                : STRING[35];
        handle     : STRING[35];
        line                : INTEGER;
        baud                : WORD;
        city                : STRING[25];
        calls       : WORD;
        doing      : STRING[20];
        hidden     : BOOLEAN;
        wantschat     : BOOLEAN;
        ranetmgr            : BOOLEAN;
        donotdisturb : BOOLEAN;
        ready      : BOOLEAN;
        msgwaiting    : BOOLEAN;
 END;


bytearray32 = ARRAY[1..32] OF BYTE;
```

```
        useronrecord = RECORD
                name,
                handle : msgtoidxrecord;
                line            : BYTE;
                baud            : WORD;
                city            : STRING[25];
                status,
                attribute : BYTE;
                statdesc        : STRING[10];
                freespace : ARRAY[1..98] OF BYTE;
                nocalls         : WORD;
         END;
(*
        Status byte -
        0 : Browsing (in a menu)
        1 : Uploading/downloading
        2 : Reading/posting messages
        3 : In a door/external utility
        4 : Chatting with sysop
        5 : Answering questionnaire
        6 : RTC
        7 : New user logon
  255 : User-defined - display StatDesc

        Attribute -
 Bit 0 : Hidden
        1 : Wants chat
        2 : Reserved for RANETMGR
        3 : Do not disturb flag
        6 : Ready (0=busy)
*)

(* RA 2.0x User Record *)
        usersrecord = RECORD
                name                    : msgtoidxrecord;
                location                : STRING[25];
                organisation,
                address1,
                address2,
                address3                : STRING[50];
                handle          : STRING[35];
                comment                 : STRING[80];
                passwordcrc     : LONGINT;
                dataphone,
                voicephone      : STRING[15];
                lasttime                : ra_time;
                lastdate                : ra_date;
                attribute       : BYTE;
(*              Bit 0 : Deleted
                1 : Clear screen
```

2 : More prompt
3 : ANSI
4 : No-kill
5 : Xfer priority
6 : Full screen msg editor
7 : Quiet mode

*)

```
        attribute2      : BYTE;

(*      Bit 0 : Hot-keys
        1 : AVT/0
        2 : Fullscreen msg viewer
        3 : Hidden from userlist
        4 : Page priority
        5 : No echomail in mailbox scan
        6 : Guest account
        7 : Post bill enabled }
*)
        flags           : flagtype;
        credit,
        pending                 : LONGINT;
        msgsposted,
        security                : WORD;
        lastread,
        nocalls,
        uploads,
        downloads,
        uploadsk,
        downloadsk,
        todayk          : LONGINT;
        elapsed                 : INTEGER;
        screenlength    : WORD;
        lastpwdchange           : BYTE;
        group           : WORD;
        combinedinfo    : combinedrecord;
        firstdate,
        birthdate,
        subdate                 : ra_date;
        screenwidth,
        language,
        dateformat      : BYTE;
        forwardto       : STRING[35];
        msgarea,
        filearea        : WORD;
        defaultprotocol: CHAR;
        filegroup       : WORD;
        lastdobcheck    : BYTE;
        sex                     : BYTE;
        xirecord                : LONGINT;
        msggroup                : WORD;
        freespace       : ARRAY[1..48] OF BYTE;
    END;

    usersxirecord  = RECORD
        freespace : ARRAY[1..200] OF BYTE;
    END;
```

```
sysinforecord  = RECORD
        totalcalls : LONGINT;
        lastcaller : msgtoidxrecord;
        extraspace : ARRAY[1..128] OF BYTE;
 END;
```

```
        timelogrecord  = RECORD
                startdate   : ra_date;
                busyperhour : ARRAY[0..23] OF WORD;
                busyperday  : ARRAY[0..6] OF WORD;
         END;

        eventrecord = RECORD
                status : BYTE;(* 0=Deleted 1=Enabled 2=Disabled *)
                starttime   : ra_time;
                errorLevel  : BYTE;
                days                : BYTE;
                forced    : BOOLEAN;
                lasttimerun : ra_date;
         END;

        eventrecordarray = ARRAY[1..20] OF eventrecord;

        messagerecord  = RECORD
                unused : ARRAY[1..4] OF BYTE;
                name            : STRING[40];
                typ             : msgtype;
                msgkinds        : msgkindstype;
                attribute : BYTE;
(*
         Bit 0 : Enable EchoInfo
                1 : Combined access
                2 : File attaches
                3 : Allow aliases
                4 : Use SoftCRs as characters
                5 : Force handle
                6 : Allow deletes
                7 : Is a JAM area
*)

                dayskill,  (* Kill older than 'x' days *)
                recvkill                : BYTE;
(*
        Kill recv msgs, recv for more than 'x' days
*)
                countkill       : WORD;
                readsecurity    : WORD;
                readflags,
                readnotflags    : flagtype;
                writesecurity  : WORD;
                writeflags,
                writenotflags  : flagtype;
                sysopsecurity  : WORD;
                sysopslags,
                sysopnotflags  : flagtype;
                originline      : STRING[60];
```

```
akaaddress      : BYTE;
age             : BYTE;
jambase         : STRING[60];
group           : WORD;
altgroup        : ARRAY[1..3] OF WORD;
```

```
                attribute2      : BYTE;
(*
        Bit 0 : Include in all groups
*)
                freespace               : ARRAY[1..9] OF BYTE;
        END;

        exitinforecord = RECORD
                baud                    : WORD;
                sysinfo                 : sysinforecord;
                timeloginfo     : timelogrecord;
                userinfo                : usersrecord;
                eventinfo       : eventrecord;
                netmailentered,
                echomailentered  : BOOLEAN;
                logintime       : ra_time;
                logindate       : ra_date;
                timelimit       : WORD;
                loginsec                : LONGINT;
                userrecord      : INTEGER;
                readthru,
                numberpages,
                downloadlimit           : WORD;
                timeofcreation  : ra_time;
                logonpasswordcrc : LONGINT;
                wantchat                                : BOOLEAN;
                deductedtime    : INTEGER;
                menustack       : ARRAY[1..50] OF STRING[8];
                menustackpointer : BYTE;
                userxiinfo      : usersxirecord;
                errorfreeconnect,
                sysopnext       : BOOLEAN;

(* These  next fields hold
  data related to an
  EMSI session
*)
                emsi_session    : BOOLEAN;
                emsi_crtdef,
                emsi_protocols,
                emsi_capabilities,
                emsi_requests,
                emsi_software           : STRING[40];

                hold_attr1,
                hold_attr2,
                hold_len                : BYTE;
                pagereason      : STRING[80];
                statusline      : BYTE;
                lastcostmenu    : STRING[8];
```

```
        menucostpermin          : WORD;
        doesavt,
        ripmode                 : BOOLEAN;
        extraspace      : ARRAY[1..86] OF BYTE;
END;
```

```
(* QBBS STRUCTURES *)

        qbbs275_flagtype = ARRAY[1..4] OF BYTE;

        qbbs275_userrecord = record
                name            : STRING[35];
                city            : STRING[25];
                pwd             : STRING[15];
                dataphone,
                homephone   : STRING[12];
                lasttime        : STRING[5];
                lastdate        : STRING[8];
                attrib      : BYTE;
                flags       : qbbs275_flagtype;
                credit,
                pending,
                timesposted,
                highmsgread,
                seclvl,
                times,
                ups,
                downs,
                upk,
                downk,
                todayk    : WORD;
                elapsed,
                len             : INTEGER;
                combinedptr : WORD; (* Record number in COMBINED.BBS *)
                aliasptr  : WORD; (* Record number in ALIAS.BBS        *)
                birthday        : LONGINT;
         END;

(*      Attrib:
         Bit 0: Deleted
         Bit 1: Screen Clear Codes
         Bit 2: More Prompt
         Bit 3: ANSI
         Bit 4: No-Kill
         Bit 5: Ignore Download Hours
         Bit 6: ANSI Full Screen Editor
         Bit 7: Sex (0=male, 1=female)
*)

        qbbs275_sysinforecord = RECORD
                callcount  : LONGINT;
                lastcaller : STRING[35];
                extraspace : ARRAY[1..128] OF BYTE;
         END;

        qbbs275_timelogrecord = RECORD
```

```
        startdate   : STRING[8];
        busyperhour : ARRAY[0..23] OF INTEGER;
        busyperday  : ARRAY[0..6] OF INTEGER;
END;

qbbs275_eventstat = (deleted, enabled, disabled);
```

```
qbbs275_eventrecord = RECORD (* EVENTCFG.DAT *)
        status     : qbbs275_eventstat;
        runtime          : LONGINT;
        errorlevel  : BYTE;
        days             : BYTE;
        forced    : BOOLEAN;
        lasttimerun : LONGINT;
        spare      : ARRAY[1..7] OF BYTE;
 END;

qbbs275_gosubdatatype = ARRAY[1..20] OF STRING[8];

qbbs275_exitinforecord = RECORD
        baudrate                : INTEGER;
        sysinfo                 : qbbs275_sysinforecord;
        timeloginfo     : qbbs275_timelogrecord;
        userinfo                : qbbs275_userrecord;
        eventinfo       : qbbs275_eventrecord;
        netmailentered: BOOLEAN;
        echomailentered: BOOLEAN;
        logintime       : STRING[5];
        logindate       : STRING[8];
        tmlimit                 : INTEGER;
        loginsec                : LONGINT;
        credit          : LONGINT;
        userrecnum      : INTEGER;
        readthru                : INTEGER;
        pagetimes       : INTEGER;
        downlimit       : INTEGER;
        wantchat                : BOOLEAN;
        gosublevel      : BYTE;
        gosubdata       : qbbs275_gosubdatatype;
        menu                    : STRING[8];
 END;
```