T R I U M P H S O F T W A R E

Presents

TriDoor

Completely revised and refined!

An On-Line Door Support Unit For use with Turbo Pascal 5.5, 6.0 and 7.0

TriDoor Written by Christopher M. Russo TriComm Communications Support Written by Jeremy H. DuBois

Voice : (508)263-4247 (Business) / (508)263-8420 (Home)

DISCLAIMER

Triumph Software and the employees/programmers in conjunction with and/or affiliated with Triumph Software can not be held responsible for the condition of any software received through any non-postal/non-parcel means.

Nor can said persons be held responsible for any damage caused to media or hardware as a direct or indirect result of the use of any of our products.

Triumph Software and it's employees and associates would like it to be known, however, that all of their products are thoroughly tested before leaving our offices. It is only in the deepest faith in our own product that any such software is released to the public.

Triumph Software reserves the right to change any documentation, disclaimer, licensing information or registration procedures and costs at any time for any reason with no prior warning or notice.

LIMITED LICENSING AGREEMENT

The holder of the unregistered TriDoor package is allowed a limited usage period of 30 (thirty) days, wherein he or she may incorporate the unit in any program that he or she develops as long as the program incorporating the TriDoor unit is not distributed in any manner at any time. After the thirty day period, the holder of the unregistered copy of the TriDoor package must register the software to be allowed further use of TriDoor.

A private registered copy may be obtained by filling out the TriDoor registration form and sending both the registration form and $15.00, in the form of a check or money order, to Triumph Software. The private copy will allow the registered owner to produce programs using the TriDoor unit and distribute them as long as the registered copy of TriDoor is not included in the package, no compensation of any kind is obtained in any way from the distribution of the software incorporating the TriDoor unit, and Triumph Software is given ample credit as depicted in the final chapter of this document. The registered copy of the TriDoor unit must not be distributed in any way other than as a compiled portion of a program or software produced by the registered copy holder.

A commercial registered copy may be obtained by filling out the TriDoor registration form and sending both the registration form and $30.00, in the form of check or money order, to Triumph Software. The commercial copy will allow the registered owner to produce programs using the TriDoor unit and distribute them. This copy also allows the holder of the registered copy to obtain legal compensation for the program incorporating the TriDoor package. The holder of the commercial registered copy must include acceptable credit in the documentation of his or her program as further explained in the final chapter of this document. The registered copy of the TriDoor unit must not be distributed in any way other than as a compiled portion of a program or software produced by the registered copy holder.

An individual copy must be registered for each individual machine and each individual company or person. The only exception is in the case of a company owned by a singular person in which case the copy may be registered for both.

Triumph Software reserves the right to revoke registration privileges either temporarily or permanently if the conditions of the registration are not properly met by the holder of the registered TriDoor package.

Triumph Software reserves the right to use the registered owners name, company and/or product created incorporating the TriDoor unit in any of Triumph Software's advertising of any type.

Registered copy holders will be entitled to minor upgrades of the TriDoor package at a minimal processing and materials fee of $5.00 per upgrade.

HOW TO USE THIS MANUAL

First and most importantly is that you read the entire manual from front to back. As easy as TriDoor is to use it will be absolutely useless to you if you don't at least read most of it, and in addition to that there are also many hints and refining tips in this document that will help you to truly take command of this software, so it is in your best interest to read *all* of it.

As far as the actual text goes, you need to keep a few things in mind about the format of this manual.

- The normal text in this document is in Times New Roman font, like this.
  The variables, functions and procedures in this manual are in Arial font, like this.

- Anything in bold arial letters is a function or procedure. i.e. **td_writeln**, **td_readln** and **read_dorinfo**.

- Anything in arial italics is an accessible variable. i.e. *disable_queue*, *stats.ansi* and *toggle.disable_screen*

The rest is simply in the text itself. Good luck and happy reading.

TABLE OF CONTENTS

1.0

Why Use TriDoor?

1.0 Why Use TriDoor?

TriDoor is the most advanced, efficient and easy-to-use utility of it's kind available on the software market today. It allows even the novice Borland Turbo Pascal programmer to create fully-functioning doors, on-line programs and utilities in a matter of minutes without ever having to delve into the hundreds of texts on communications programming.

In addition to having complete communications support TriDoor also comes equipped with many other features which are extremely useful when writing communications applications. Examples of these are color ANSI graphics controls, user time handling routines and many Pascal-like commands such as **td_writeln** which is the communications equivalent of **write** and **writeln**.

TriDoor is flexible too! With the exception of a few things here and there (like the tell-tale chat mode) your users may never even know that you did not write the entire program- communications routines and all. You will see no hard-coded error messages, forced configuration files or annoying restrictions that you did not develop yourself. This allows you to use all your programming resources the way YOU want as opposed to the way a tool kit may have set them up. A prime example of this is command line parameters; some tool kits will force you to enter in the name of a configuration file on the command line. What if you would rather enter in an IRQ setting or a user name? What if you do not want a configuration file? TriDoor allows you to do as you will.

The programming possibilities are endless- I have written on-line games, archive viewing utilities, text/ANSI graphics file viewers, "top ten list" generators, terminal programs, SysOp paging doors and many MANY more.

How many times have you said to yourself, "This is truly an excellent bulletin board system I have running if only I could add on a [blank] without changing to a new BBS software..."? Well, there are a few solutions to this problem: You could switch BBS software which may give you a few options you did not have before but you are also reducing yourself to rebuilding a system you have already put probably hundreds of hours of work into and potentially compromising the quality of your user's environment - or- you could download a door from your local BBS where you may find a program to do what you

need, but it may not be EXACTLY what you want -or- you could use TriDoor to write your own! You could have the door you want, make the changes you need at ANY time and have the pleasure of having made it yourself.

I frequently find myself saying, "If you cannot do it with TriDoor, then you probably cannot do it." I have only scraped the surface of it's capabilities and ease of use; the rest is up to you. You now hold the power to make the doors, on-line games and utilities that you have always wanted!

2.0

Features of TriDoor

TriDoor has many advanced and VERY easy-to-use features which are built right in. And although there are so many that I literally cannot remember ALL of them, we are still adding all the time so give us a call so we can get you a copy of the latest version.

- NOW VERY INEXPENSIVE! SEE LICENSING AGREEMENT!

- Automatic set-up and support of communications ports
- Automatic reading and processing of DORINFO1.DEF and PCBOARD.SYS
- Easy to use commands such as **td_writeln** and **td_readln** which function as communications supporting equivalents of the Turbo Pascal commands
- Built-in chat mode with word-wrap and ANSI color graphics
- Boolean variable halt_program which is set to TRUE if [F8] is hit from the local keyboard- allows for programmer-defined "hang-up" procedures
- Easy forced "caps-lock", and "password-entry" (see "***" instead of "hey" or "YOU" instead of "you")
- Length restriction available on **td_readln** input fields
- Built-in string capitalization, first letter capitalization and easy integer/longint to string conversion available
- Built-in direct screen write functions
- Automatic status bar displaying time, name, baud rate and program name
- Automatic on-line user time handling routines
- Built-in routines for clearing user's and local screen
- Built-in Boolean response "Are you sure? (Y/[N])" function
- Built-in user-page (>BEEP!< "The SysOp is paging you!" >BEEP!<)
- Automatic "Last Printed" memory
- Command list on local screen in status bar
- Auto-sensing of color/monochrome local screen
- Borland Turbo Pascal 5.5, 6.0 and 7.0 support
- Full color ANSI graphics support

- On-line user time modification from local console
- Automatic user drop out monitoring (halt_program := TRUE) at time<1
- System messages in status bar ("SysOp : User is being paged.")
- Ability to EASILY add support for ANY BBS system
- Ability to be run in multi-tasking environments such as DESQVIEW and WINDOWS.
- Local disable/enable of remote user keyboard
- Ability to fake line-noise generation from local console
- Ability to use non-standard IRQ/COMM port configurations
- Ability to change all communications parameters
- Locked communications port support
- Complete flow control support (XON/XOFF and CTS/RTS)
- Configurable/disable able user keyboard time-out feature
- Messages such as "Entering chat mode." can now be changed
- Configurable return to normal status bar after nn seconds
- Procedure naming conventions similar to Turbo Pascal's
- Efficient, fast, operation
- ANSI capable **display_file** procedure
- Ability to disable local screen from program or local [F9] keypress
- PCBoard convention local command keys ([F5]=dos shell, [F10]=chat mode)
- A built-in DOS shell

... And more!

3.0

What's New This Version

3.0 What's New This Version

While being rather cliché, I think an easier question in this case would be "What ISN'T new on this version?" The answer being "Not a heck of a lot."

I have been meaning to do a lot of these revisions and improvements for quite some time now, but higher responsibilities within the company and my own life often drag me behind. After some back-breaking effort and a few sleepless nights, however, here it is.

First and foremost are the fees. I have reduced them to staggeringly low values of $15.00 for the private copy and $30.00 for the commercial copy. Economy? Fit of niceness? General insanity? Who knows? Just enjoy it... and register!

The worst and best news is that all of the procedures, functions and variable names have been changed; **getinput** has become **td_readln**, **print** has become **td_writeln** and so on. While this is going to take a little elbow grease and a good global-replace session for some of you, for those of you just starting, it will make life a lot easier.

I have also added a configurable and disable able user keyboard inactivity time-out feature. A very nice addition since otherwise the user could sit there for forty-five minutes and never hit a key and simply tie up the line.

I have also made a minor modification that simply restores your status-bar at the bottom of the local screen after a number of seconds which you can set. While this is a minor thing, it is nice since no-one really wants to know that you have to hit Escape to exit chat mode thirty minutes after you have left the session.

The SysOp keys have also been modified to comply with PCBoard conventions. There are two reasons for this : one is that I like it better and I think PCBoard is excellent, and two is that more and more doors are accepting this convention.

All of the procedures and functions have been streamlined. I have removed, blended, combined and added certain processes and functions in order to make the program faster and more compact. Most will not notice a difference until they get up into the higher speeds.

Also new are the built-in local DOS shell and the ability to disable the local screen. This screen disable function is often handy for people running multiple nodes and can be done through [F9] or in the program itself.

Finally is the ability to change such messages as "Entering chat mode." In the future you may be able to use actual text and ANSI files for these, but at the moment you are limited to 255 characters.

While there are more slight changes and modifications here and there, writing them all would take the rest of this document, so I would simply advise reading the rest of the document since that is where you will find them.

4.0

Standard Programming Procedures

4.1 PROGRAMMING WITH TRIDOOR

Creating doors with TriDoor is an astoundingly simple process but a couple steps must be taken in each program you write in order for the program to function normally. The computer may physically "hang" if these few following procedures are not completed (but will usually just drop out with a complaint about not being able to set-up properly.) Once you have gone through these procedures you then may use any of the commands listed in the command summary just as you would normal Turbo Pascal commands.

A very basic example of how to set-up a simple program (EXAMPLE.PAS) comes with this software package. While it is very crude and simplistic it will be an excellent building block from which you can start. In the future I intend to release an actual functioning door which you may do with as you please. This will most likely not be anything more than a BBS phone number maintenance door, but it will be more in-depth than the example program included with this package. Until then I have put a great deal of effort into making this document more verbose so as to make it even easier to understand how to program with TriDoor.

Despite the examples and this manual, however, it is very important that you have a basic understanding of the following Turbo Pascal/Pascal features : records, global variables, constants, functions, procedures, loops, units, and general syntax. Without these and the other basic fundamentals of Turbo Pascal, you will have a more difficult time using this unit. This is *not* a "let's learn Turbo Pascal" document! This *is* a "let's learn TriDoor" document. This is also *not* a "let's learn the fundamentals of BBS operation" document, so you need to know how your system is set-up and it does help to have a little experience with setting up doors as well.

Do not be discouraged if you feel under-knowleged. The likelihood is that you know more than you realize, and if you even know a small amount about these things, you should be all set. Press on! If something confuses you, simply refer to the appropriate manual and learn more about it.

The following is an actual step-by-step process of what you need to do to set up TriDoor with your program. It is a good idea to print out and refer to the program EXAMPLE.PAS as you go along in this section.

4.1 Standard Programming Procedures

The very first thing you must do is declare your TriDoor unit in your 'uses' declaration within your source code. TriDoor needs to redefine some of Turbo Pascal's CRT constants for it's own use, so it is important to declare your TriDoor unit after the CRT unit in all cases. These reassignments will not affect your program's operation in any adverse or noticeable way.

TriDoor supplies a global record variable, *stats*, to you. In the next portion of your program, you must fill in all of the fields in this record. This process is done automatically by two functions that are included in the TriDoor unit; these functions are **read_dorinfo** and **read_pcboard**. These functions read in the DORINFO1.DEF and PCBOARD.SYS door support information files, respectively, and fill them into the *stats* record for you.

If you are planning to support a BBS software that does not create either of these support files, you must then fill in these values by means of a like procedure of your own. To make it easier to understand just what we have done, a copy of our **read_dorinfo** and **read_pcboard** functions have been included in the section of this document entitled **CREATING YOUR OWN DOOR SUPPORT**.

Remember that if you are using non-standard IRQ settings, you must make it so that your program can determine the appropriate address from another means, such as a command-line parameter or a configuration file of your own.

A list of TriDoor-defined global constants such as COM1, IRQ3 and others is listed in the section entitled **GLOBAL CONSTANTS**. These constants have addresses assigned to them to make programming easier.

There are also many variables accessible to you which will have a great deal of affect on the amount of control you have over TriDoor. These are listed in the chapter **ACCESSIBLE GLOBAL VARIABLES**.

TriDoor also supplies you with a string variable; *support_path*. The functions **read_pcboard** and **read_dorinfo** will look in the path specified in this variable for the on-line door support information files. In other words, if you fill in this variable with the path where these files are available, these functions will look in that directory for those files. This is very useful because it allows you to supply a path to the files instead of copying them into the directory where the door is executed.

The *support_path* global variable, like all others, can be altered, and incorporated into your own code to suit your needs. It is recommended that if you write procedures for other door support files that you use this variable in your function.

Once the *stats* record has been successfully filled, all you need then do is call the Boolean **setup_tridoor** function. If this function returns a TRUE then your communications link has been setup properly and you can then use any of the other TriDoor functions and procedures as you would a normal Turbo Pascal command. If it returns a FALSE then you have most likely supplied TriDoor with some inaccurate information somewhere along the way and the TriDoor functions and procedures will not work until you fix the problem and call the procedure again.

Don't forget the locked communications support! Because of today's high-speed modems it is sometimes necessary to "lock" the speed from your computer to your modem at a higher rate than the modems actual connection to the remote caller. This is done so that optimum performance from your high-speed modem can be obtained.

This version of TriDoor has two new variables to allow for "locked" communications ports. The old *stats.baud* has been changed to *stats.real_baud* which is the actual connection speed. The *stats.lock_baud* variable is your locked communications port setting, which is actually the speed at which the local computer and modem are transmitting to each other.

To support locked communications ports you need only to fill in these two variables before running the **setup_tridoor** function. This topic will be discussed at length in the section entitled **LOCKED COMMUNICATIONS SUPPORT**. Remember that any door support information file reading function supplied with TriDoor automatically checks for locked communications support. The only exception being on how each given QuickBBS and QuickBBS clone creates the DORINFO1.DEF file. If it supplies the locked baud rate like it is supposed to do, you will be all set. Otherwise, you will have to supply the baud rate through another means such as the aforementioned command line parameter or configuration file.

Another very important thing is to keep the stack and heap sizes at a size large or small enough to allow the built-in TriDoor DOS shell to function properly. This is done through the $ M compiler directive which is explained in your Turbo Pascal manual. Keep in mind that in many cases, the appropriate values can be gotten by a little trial and error. Also keep in mind that as your program grows you may need to change the compiler directives to accommodate for size and variable memory consumption.

4.2 PROGRAMMING TIPS

Here are several ideas and concepts to keep in mind as you program. Remember that TriDoor is very flexible to allow you to do pretty much what you like with the commands and make it as personalized as possible but due to this flexibility some things are not so readily obvious (or forced, for that matter) as they might be in a more structured programming environment.

- Remember to check the **exit_door** function in every loop. This function will check every possible condition that would indicate that the program needed to halt. If you do not do this, you will wind up with a door that will simply hang when a user logs off or runs out of time.

- If you create support for another BBS support file it is often a good idea to make the procedure a Boolean returning function so that way you can immediately tell if there was a problem when the program tried to read the data. TriDoor's **read_dorinfo** function does this. See the example program (EXAMPLE.PAS) provided with this package to see how it is used.

- Again, if you are creating support for another BBS, try to use the *support_path* variable to allow your program to be present in a directory other than the one that your support file is in. Refer, once again, to the **read_dorinfo** function in the chapter PROGRAMMING WITH TRIDOOR.

- If you want the local user (SysOp) to see something that the remoter user (on-line user) does not, use a **writeln** instead of a **td_writeln** or perhaps use TriDoor's **statusbar_message** routine.

- Remember to check to see if the local screen is disabled before using any non-TriDoor screen writing function such as **write** or **writeln**.

- Try not to exclude non-ANSI graphics users from your door. Many people do not have ANSI graphics support. Don't limit yourself and your program.

- Remember that many bulletin board systems run better if they stay in memory while a door is run, and some HAVE to stay in memory at all times. Keep this in mind when writing your doors and keep them small, or break them into modules. Don't put once-a-day-maintenance programs as part of the main door, etc.

- THINK IT OUT! If there's something you want to do, but are not sure if you can with TriDoor then take a moment to think it through. I can almost guarantee you can do it. If you cannot think of a way, give us a call and we will give you a hand or remedy the problem.

5.0

Advanced Programming Procedures

5.1 ADVANCED PROGRAMMING WITH TRIDOOR

When you begin going beyond the realms of the basic TriDoor features made available to you, you then begin having the need to access some of the global variables and records.

You also need to be familiar with certain terms and concepts which will not only allow you to make excellent communications procedures of your own, but are also necessary to understand the next few sections of this document. You should read this section whether or not you plan to attempt this because it will also allow you a better understanding of how TriDoor works.

The first thing you should understand is the concept of a halting condition. A halting condition is a defined number of possibilities that form together in various combinations that indicate that the program should be stopped. An example of this is when a carrier is dropped. Because the program was not in local operation (which means that the variable *toggle.user_local* was set to FALSE at some point during startup) and the carrier had been dropped (which means that the TriDoor function **carrier_detect** returned a FALSE) the program is then expected to come to a halt. It is important to also keep in mind that there is one Boolean variable, *toggle.halt_program*, which has priority over all other halting conditions. That is, if this variable is set to true, no matter what other conditions say, the program is expected to drop out immediately. TriDoor's **exit_door** function automatically checks these variables, in order of priority, at all times. It is important to check this procedure in all of your customized door procedures and functions. Another thing that is important to understand is that, at some times, the Boolean variable *toggle.disable_screen* will be set by either a local command-keypress ([F9]) or by your own program. This means that the SysOp does not want the screen to be updated until the screen is re-enabled. It is important to monitor this toggle whenever you plan on writing to the local screen. All TriDoor output functions do this automatically, but Turbo Pascal's **write** and **writeln** do not!

5.2 LOCKED COMMUNICATIONS SUPPORT

It is very important to support locked communications support in any software that you plan to release to the public. If you do not a fair amount of people with high-speed modems will not be able to use your package or will simply not bother to try.

If you are running PCBoard 14.x or any other BBS that supplies a compatible PCBOARD.SYS file then you should not have to concern yourself with this issue because **read_pcboard** will automatically handle any values necessary to set this support up. If you run another BBS, however, different scenarios may apply. Some of these are as follows:

If you are running QuickBBS or another BBS that supplies a DORINFOx.DEF file you MAY have to supply TriDoor with the locked baud rate through some other means such as a configuration file or a parameter. Never hard code the locked baud rate because others may want to change it. All other information will be filled in automatically by **read_dorinfo**. SOME QuickBBS clones provide the locked baud rate in the normal baud rate field.

If you are running any other BBS there are two possible scenarios. If your BBS supplies you with the locked port information in it's door support file then you may simply read it in like TriDoor's **read_pcboard** does. If your BBS does not supply you with the necessary information you must obtain the information from a configuration file or command line parameter as in the case of a DORINFOx.DEF support file.

5.3 CREATING YOUR OWN DOOR SUPPORT

This is a copy of the **read_dorinfo** function which is incorporated directly into the TriDoor package. It is supplied to you as a reference for creating your own BBS support functions.

```
{* * * * * * * * *}

function read_dorinfo : Boolean;

{ reads a users stats from the file DORINFO1.DEF and returns false if the file was not
found }
{ in the path stored in support_path }

var
    deffile : text;              {* pointer to the file DORINFO1.DEF *}
    i,
    loop1,
    loop2   : integer;           {* local indexing loops *}
    comst,
    filler  : string;            {* temporary work strings *}
    work1,
    work2   : string[40];        {* short temporary work strings *}

begin { read_dorinfo }

    if fsearch(mainpath+'DORINFO1.DEF','') <> '' then {* if the file was found... *}
        begin

            assign(deffile,mainpath+'DORINFO1.DEF');     {* assign & reset file *}
            reset(deffile);

            for loop := 1 to 4 do        {* get rid of some un-needed strings *}
                readln(deffile,filler);
```

```
case filler[4] of                       {* determine comm port *}
    '0' : stats.comport := 0;
    '1' : stats.comport := COM1;
    '2' : stats.comport := COM2;
    '3' : stats.comport := COM3;
    '4' : stats.comport := COM4;
end;

readln(deffile,stats.comstr);           {* read in '1200 N,8,1' type string *}

readln(deffile,filler);                 {* ignore un-needed string *}

readln(deffile,work1);                  {* read in user's first and last name *}
readln(deffile,work2);

readln(deffile,filler);                 {* ignore un-needed string *}

readln(deffile,i);                      {* read in ANSI/non-ANSI *}

readln(deffile,filler);                 {* ignore un-needed string *}

readln(deffile,stats.time);             {* read in user time *}

close(deffile);                         {* close file, done reading *}

stats.name := work1 + ' ' + work2;      {* make a name into a full name *}

work1 := '';
loop2 := 0;
loop1 := length(stats.comstr);

    {* search the '1200 N,8,1' style string up to the first space *}

while (loop2<loop1) and (ord(stats.comstr[loop2+1]) <> 32) do
    begin
        loop2 := loop2 + 1;
        work1 := work1 + stats.comstr[loop2];
    end;

val(work1,stats.real_baud,loop);        {* get the value of the string and *}
                                        {* store it into the baud rate value    *}
```

```
            if i = 1 then stats.ansi := TRUE        {* assign stats.ansi toggle *}
                else stats.ansi := FALSE;

            i := pos('BAUD',stats.comstr);          {* get position of the word 'BAUD' *}
            i := i + 5;                             {* for reference to next procedure   *}

            work1 := copy(stats.comstr,i,1);        {* get parity *}
            case work1[1] of
                'N' : stats.comp_parity := NONE;
                'E' : stats.comp_parity := EVEN;
                'O' : stats.comp_parity := ODD;
                'M' : stats.comp_parity := MARK;
                'S' : stats.comp_parity := SPACE;
            end;
```

```
                                                                                val(copy(st
                                                                                {* get data


                                                                                stats.comp_
```

```
            val(copy(stats.comstr,i + 5,1),loop1,loop2);  {* get stop bits *}
            stats.comp_sbits := loop1;

            read_dorinfo := true;                   {* we were successful, return a true *}
        end
    else read_dorinfo := false;                         {* file not found, return a false *}

end;  { read_dorinfo }

{* * * * * * * * *}
```

6.0

Standard Command Summary

6.0 Standard Command Summary

The following pages are a list of main commands made accessible to the programmer by TriDoor. These commands are those that are needed primarily and should accommodate to your every communications need.

Additional, more advanced, commands are also available to the programmer for use when programming alternates or substitutes to some of TriDoor's main commands. These commands are in the section entitled **ADVANCED COMMAND SUMMARY**.

Synopsis

procedure direct_gotoxy(x,y : integer);

Description

This procedure will position the screen memory cursor position in the place in memory relative to the position (*x,y*) on the actual screen. This procedure only affects the local screen.

If the program does not set the direct screen memory position by calling on the **direct_gotoxy** procedure, the **direct_char** and **direct_strg** functions will resume where the last execution of a direct output function left off.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    direct_gotoxy( 0, 24 );
    direct_strg( '[F1] Help 1 [F2] Help 2 [F3] Normal Status Bar [F5] DOS',
112);
end;
```

Variables

direct_xypos
toggle.disable_screen

See Also

direct_strg, direct_char
ansi_gotoxy

Synopsis

procedure direct_char(d_char : char; d_attr : integer);

Description

This procedure will place the character *d_char* directly into screen memory, and therefore, on the screen in the position relative to the memory location stored in the variable *direct_xypos*. This position can be set by using the procedure **direct_gotoxy.**

The *d_attr* variable is the color setting of the character your are about to display. It is a very useful parameter since it will allow you to create any color combination of background and foreground colors with just one number.

For a color-coded listing of these colors, as well as an example of how to determine what number corresponds with what color, run the program DISPATTR.EXE, included with this package.

If the program does not set the direct screen memory position by calling on the **direct_gotoxy** procedure, the **direct_char** and **direct_strg** functions will resume where the last execution of a direct output function left off.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    direct_gotoxy( 1, 10 );
    direct_char( 'A', 7 );        {* write a character at 1,10 *}
    direct_char( 'B', 7 );        {* write another character at 2, 10 *}
end;
```

Variables

direct_xypos
toggle.disable_screen

See Also

direct_gotoxy, direct_strg
ansi_gotoxy

Synopsis

procedure direct_string(d_strg : string; d_attr : integer);

Description

This procedure is identical to **direct_char** except that it will allow you to display an entire string instead of just a single character.

The *d_attr* variable is the color setting of the character your are about to display. It is a very useful parameter since it will allow you to create any color combination of background and foreground colors with just one number.

For a color-coded listing of these colors, as well as an example of how to determine what number corresponds with what color, run the program DISPATTR.EXE, included with this package.

If the program does not set the direct screen memory position by calling on the **direct_gotoxy** procedure, the **direct_char** and **direct_strg** functions will resume where the last execution of a direct output function left off.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    direct_gotoxy( 1, 5 );
    direct_strg( 'This is an exciting example : ', 7 );  {* write at 1,5 *}
    direct_strg( 'SEE?!', 7 );                  {* continue right after prev. *}
end;
```

Variables

direct_xypos
toggle.disable_screen

See Also

direct_gotoxy, direct_char
ansi_gotoxy

Synopsis

```
procedure set_cts( cts_on : Boolean );
```

Description

This procedure enables and disables the active modem's hardware flow control. Hardware flow control is very important for proper modem operation, especially in the case of modems operating at speeds over 2400 baud.

This procedure is automatically handled and executed by TriDoor.

The default setting for the flow control is on. (TRUE)

Example

```
begin
    set_cts( TRUE );
end;
```

Variables

none

See Also

cts_true

cts_true

Synopsis

function cts_true : Boolean;

Description

This function polls the modem and returns a Boolean TRUE if it is clear to send.

This function is automatically monitored by all of TriDoor's output functions and procedures.

Example

```
begin
    while ( not cts_true ) do;      (* Loop while the modem is not ready *)
    writeln( 'Modem is now ready for character.' );
end;
```

Variables

none

See Also

set_cts

carrier_detect

Synopsis

function carrier_detect(comm_port : integer) : Boolean;

Description

This function will return a Boolean TRUE if a carrier is detected at the communications port *comm_port*.

Global constants for communications ports are made available by this TriDoor. They are COM1, COM2, COM3 and COM4; representing the four main-stream ports used by PCs.

This function is constantly checked within TriDoor, but if you create a new procedure, you may find it necessary to monitor this function. The optimum alternative is to monitor the **exit_door** function instead, however, since it also checks the *halt_program* Boolean variable, and also takes into account whether or not the user is in local mode. (There may not be a carrier, but if the user on-line is in local mode, there is no reason to drop them out of the program, and thus **exit_door** would return a FALSE.)

Example

```
begin
    if ( carrier_detect( stats.comport ) ) then
        writeln( 'There is a carrier present.' )
    else writeln( 'There is no carrier present.' );
end;
```

Variables

stats.comport
COM1, COM2, COM3, COM4 (constants)

See Also

exit_door

Synopsis

function setup_tridoor : Boolean;

Description

This function performs all the necessary setup routines for operation of your door. It returns a Boolean TRUE if it did so without any problems, and a FALSE if there was some error in setup. Errors are usually caused by trying to set up the wrong communications port, baud rate, or other critical information.

Before this routine is called, your program must fill in the *stats* record with all appropriate settings. This is where **setup_tridoor** will look for baud rate, communications port, IRQ settings, and more. For further information regarding the *stats* record see the chapters entitled **ACCESSIBLE GLOBAL VARIABLES** and **PROGRAMMING WITH TRIDOOR**.

TriDoor also provides you with two functions which fill in this record, automatically, from two different bulletin board system door support files. These functions are **read_dorinfo** and **read_pcboard**, and they read in the necessary data from the files DORINFO1.DEF and PCBOARD.SYS, respectively.

** If you do not call the **setup_tridoor** procedure, all other TriDoor functions and procedures will not operate properly, and may cause your system to lock up! **

Example

```
begin
    if ( read_pcboard ) then
        begin
            if ( setup_tridoor ) then
                begin
                    (* main door operation here *)
                end
            else writeln( 'ERROR : TriDoor was not able to set up the comm
port.' );
        end
    else writeln( 'ERROR : PCBOARD.SYS not found.' );
```

Variables

stats (record)

See Also

read_pcboard, read_dorinfo

Synopsis

```
function exit_door : Boolean;
```

Description

This function will check the carrier on the current communications port, the value of the *toggle.halt_program* and *toggle.user_local* Boolean variables, and the on-line user time statistics. From all of these values, the **exit_door** function automatically determines whether or not the on-line user should be dropped out of the program.

The order of halting condition precedence is : halt, local, carrier.

In other words, no matter what conditions prevail, if *toggle.halt_program* is TRUE, **exit_door** will return a TRUE. If the function **carrier_detect** returns a FALSE, but *toggle.user_local* is TRUE then **exit_door** will return a FALSE. etc.

The following is a table that shows the results of a call to the **exit_door** function under all possible circumstances.

Example

```
var
    name_strg : string[35];

begin
    td_writeln( 'Please enter your name after the prompt.  Your name must be||' );
    td_writeln( 'longer than three characters.||' );

    while ( length( name_strg ) < 3 ) and ( not exit_door ) do
        begin
            td_writeln( '||Enter your name : ' );
            td_readln( name_strg, 35, CAPS_ON,  CODE_OFF );
            if ( length( name_strg ) < 3 ) then
                td_writeln( '||Your name must be at least three characters long.||' );
        end;
end;
```

Variables

toggle.halt_program, toggle.user_local, toggle.inactivity_timeout

See Also

carrier_detect
time_until_timeout, time_remaining
display_time

Synopsis

```
function  strg( in_value : longint ) : string;
```

Description

This function will take any value given to it as *in_value* and will return a string representation of that value.

Example

```
var
    int_val : integer;
    long_val : longint;
    temp_strg : string;

begin
    temp_strg := strg( long_val );
    td_writeln( 'The value of the long integer is : ' + temp_strg + '||'  );
    temp_strg := strg( int_val );
    temp_strg := temp_strg + ' is the value of the standard integer.';
    td_writeln( temp_strg + '||' );
    td_writeln( 'The value of the sum of ' + strg( long_val ) + ' and ' +
        strg( int_val ) + ' is ' + strg( long_val + int_val ) + '.||' );
end;
```

Variables

none

See Also

td_upcase, capitalize

td_upcase

Synopsis

function td_upcase(in_strg : string) : string;

Description

This function is TriDoor's replacement for/supplement to Turbo Pascal's **upcase** command. The difference being that this function will capitalize an entire string and not just one character.

Example

```
var
    bbs_name : string;

begin
    write( 'Please enter the name of this BBS : ' );
    readln( bbs_name );
    bbs_name := td_upcase( bbs_name );
    writeln('You have entered ', bbs_name, ' as the name of this BBS.' );
end;
```

Variables

none

See Also

strg, capitalize

capitalize

Synopsis

function capitalize(strg : string) : string;

Description

Don't let the name of this function fool you; it does a lot more than just convert everything into upper-case. This function will return a string with every first letter of every word capitalized, and all others lower-cased.

i.e. 'HURON CAROL' would become 'Huron Carol'.
'chris russo' would become 'Chris Russo'.
'HoWarD MCgEe' would become 'Howard Mcgee'.

(notice that this function does not notice prefixes on names like McGee and DuBois.)

Example

```
var
    name_strg : string[35];

begin
    td_writeln( 'Please enter your name : ' );
    td_readln(  name_strg, 35, CAPS_ON, CODE_OFF );
    name_strg := capitalize( name_strg );
    td_writeln( 'Your name will appear as ' + name_strg + ' on this system.||' );
end;
```

Variables

none

See Also

td_upcase, strg

Synopsis

procedure td_clrscr;

Description

This procedure is the TriDoor communications equivalent of Turbo Pascal's **clrscr** procedure. It will clear both the local and remote screen, if applicable. If ANSI is enabled by setting *stats.ansi* to TRUE this procedure will automatically take advantage of ANSI screen clearing by calling TriDoor's **ansi_clrscr** procedure.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    td_clrscr;
end;
```

Variables

toggle.disable_screen

See Also

ansi_clrscr

td_outstrg

Synopsis

procedure td_outstrg(comm_port : integer; out_strg : string; echo_strg : Boolean);

Description

This procedure is a very basic, un-filtered, **writeln** equivalent. It will output the string *out_strg* to the communications port *comm_port*. If *echo_strg* is TRUE it will also display the output on the local console.

The major differences between TriDoor's **td_outstrg** and Turbo Pascal's **writeln** are as follows :

- Inserting variables requires +'s not ,'s.

```
Turbo Pascal : write( 'TriDoor is a great ' , adj_strg );
TriDoor      : td_outstrg( stats.comport, 'TriDoor is a great ' + adj_strg,
               ECHO_ON );
```

- **td_outstrg** requires that all values be turned to strings.

```
Turbo Pascal : write( 'TriDoor Ver ' , TDVER );
TriDoor      : td_outstrg( stats.comport, 'TriDoor Ver ' + strg(TDVER),
               ECHO_ON );
```

- **td_outstrg** will not do a carriage return unless given one. (This reduces the number of commands by a factor of two, and sometimes reduces the number of actual instructions.)

```
Turbo Pascal : writeln( 'TriDoor is truly amazing!' );
               writeln;
TriDoor      : td_outstrg( 'TriDoor is truly amazing!'+ CR + CR );
```

(CR is a global constant supplied by TriDoor)

- **td_oustrg** allows your door program to display a text message to only the remote screen, or to both screens.

```
Turbo Pascal   : writeln( 'This will only be displayed on the local screen.' );
TriDoor        : td_outstrg( stats.comport, '...only on remote!',
ECHO_OFF );
               : td_outstrg( stats.comport, '...on both screens!',
ECHO_ON );
```

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
const
    TDVER  = 3.00;
    CR     = chr(13) + chr(10);

var
    adj_strg : string;

begin
    td_outstrg( stats.comport, 'TriDoor Ver ' + strg(TDVER), ECHO_ON );
    td_outstrg( stats.comport, 'TriDoor is a great ' + adj_strg, ECHO_ON );
    td_outstrg( stats.comport, 'TriDoor is truly amazing!'+ CR + CR,
ECHO_ON );
end;
```

Variables

stats.comport
toggle.disable_screen
CR, BS, ECHO_ON, ECHO_OFF (constants)

See Also

td_outchar, td_writeln, statusbar_message
direct_char, direct_gotoxy

Synopsis

procedure td_outchar(comm_port : integer; outchar : char; echo_char : Boolean);

Description

This procedure is identical to **td_outstrg** except that it is modified to accept only one character at a time. When the programmer is sure that only one character will be output, this is a *slightly* faster procedure to use.

I have given the programmer access to this procedure mainly so that if they would like to create an alternate or substitute string output procedure they can do so without sacrificing a great deal of speed. However, the **td_outstrg** procedure *is* the fastest string output procedure that you can possibly have in TriDoor since it accesses the appropriate interrupts directly within the procedure. In conclusion, unless you will be making a function or procedure that has significant differences from TriDoor's, we advise you just use the ones given to you.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    td_outchar( stats.comport, 'T', ECHO_ON );
end;
```

Variables

ECHO_ON, ECHO_OFF (constants)

See Also

td_writeln, td_outstrg, statusbar_message
direct_char, direct_gotoxy

td_writeln

Synopsis

procedure td_writeln(strg : string);

Description

This is the more refined version of **td_outstrg**. It is much easier to use since it refers to the variable *toggle.td_writeln_echo* to determine whether or not the text should be echoed to the local console. It also automatically assumes that you will be using *stats.comport* as your default communications port.

In addition to these benefits, it also allows you to replace a '+CR+' with a '||' to represent a carriage return.

- For example :

```
td_outstrg( stats.comport, 'This manual is long.'+CR+'I enjoy reading'+CR,
            ECHO_ON );
```

can be replaced by...

```
td_writeln( 'This manual is long.||I enjoy reading.||' );
```

- You can also link together carriage returns :

```
td_writeln( 'Another example.||||How riveting.||' );
```

This would produce 'Another example.'
"
'How riveting.'

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    td_writeln( 'This manual is long.||I enjoy reading.||' );
    td_writeln( 'Another example.||||How riveting.||' );
end;
```

Variables

toggle.td_writeln_echo, toggle.disable_screen
CR

See Also

td_outchar, td_outstrg, statusbar_message
direct_char, direct_gotoxy

Synopsis

procedure statusbar_message(message_strg : string);

Description

This procedure will display the message passed to it in the variable *message_strg* in the status bar window. It is an excellent way for your program to communicate with the SysOp without allowing the user see the message as well.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    statusbar_message( 'Press [ESC] at any time to exit chat mode.' );
end;
```

Variables

toggle.disable_screen

See Also

display_statusbar, display_helpbar_1, display_helpbar_2

Synopsis

function td_keypressed : Boolean;

Description

This is the TriDoor communications equivalent of Turbo Pascal's **keypressed** function. It will check the input buffer to determine whether or not a key has been pressed and sent from the remote user and return a Boolean TRUE if there is one waiting.

This function does not acknowledge local key presses.

Example

```
var
    in_key : char;

begin
    if ( td_keypressed ) then       {* if a remote key was pressed *}
        in_key := td_readkey

    else if ( keypressed ) then     {* if a local key was pressed *}
        in_key := readkey;
end;
```

Variables

buffer_top, buffer_head, buffer_tail

See Also

confirm
td_readkey, get_keypress
td_readln
get_phone, get_date

Synopsis

function td_readkey : char;

Description

This function is the TriDoor communications equivalent of Turbo Pascal's **readkey** function. It will remove and return the next waiting character in the input buffer.

This function does not acknowledge local key presses.

Example

```
var
    in_key : char;

begin
    if ( td_keypressed ) then
        in_key := td_readkey        {* read a key from comm port *}

    else if ( keypressed ) then
        in_key := readkey;      {* read a key from local keyboard *}
end;
```

Variables

buffer_top, buffer_head, buffer_tail

See Also

confirm
td_keypressed, get_keypress
td_readln
get_phone, get_date

Synopsis

function get_keypress : char;

Description

This function is an enhancement on the previous **td_keypressed** and **td_readkey** functions. It will wait for a character to be pressed, either locally or remotely, and then return it. If *toggle.disable_user_keyboard* is set to TRUE it will *not* acknowledge remote key presses.

Example

```
var
    in_char : char;

begin
    in_char := get_keypress;
end;
```

Variables

toggle.disable_user_keyboard, toggle.disable_screen

See Also

confirm
get_phone, get_date
td_readln
td_readkey, td_keypressed

Synopsis

```
procedure td_readln( input_strg : string; max_length : word;
                     caps_lock, coded_input : Boolean ) : string;
```

Description

This function is the TriDoor *near* equivalent of the Turbo Pascal **readln** procedure. It acknowledges both the local and remote keyboards, *unless* the global variable *toggle.disable_user_keyboard* is set to TRUE. In such a case, remote keystrokes will be ignored. *toggle.disable_user_keyboard* is automatically set by TriDoor's special remote function keystroke [ALT]-[F4].

There are also some additional features mad available in TriDoor's **td_readln** that are not available in Turbo Pascal's **readln**.

- You can set the maximum length of the string in the field *max_length*, or enter the global constant MAXIMUM for no relative maximum. (Absolute maximum is 255 characters.)

- If *caps_lock* is set to TRUE, all input will be instantly forced into upper-case letters. So if the user types "Lloyd Alexander" they will see, and the procedure will record "LLOYD ALEXANDER". TriDoor supplies the global constants CAPS_ON and CAPS_OFF to use in the place of *caps_lock* when calling the function.

- If *coded_input* is set to TRUE, all input will be instantly forced into "*"s. (asterixes) So if the user types "TERRY BROOKS" they will see "**********" but the procedure will record "TERRY BROOKS" as the input. TriDoor supplies the global constants CODE_ON and CODE_OFF to use in the place of *coded_input* when calling the function.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
var
    instring_long  : string[80];
    instring_short : string[30];

begin
    td_writeln( 'Please enter a message to the SysOp : (max 80 characters)||' );
    td_readln( instring_long, 80, CAPS_OFF, CODE_OFF );

    td_writeln( 'Please enter your name : (max 30 characters)||' );
    td_readln( instring_short, 30, CAPS_ON, CODE_OFF );

    td_writeln( 'Please enter your password : (max 15 characters)||' );
    td_readln( instring_short, 15, CAPS_ON, CODE_ON );
end;
```

Variables

CAPS_ON, CAPS_OFF, CODE_ON, CODE_OFF (constants)
toggle.disable_user_keyboard, toggle.disable_screen

See Also

confirm
get_keypress
td_readkey, td_keypressed
get_phone, get_date

Synopsis

function get_phone : string;

Description

This function is a hybrid of the **td_readln** function that will read in a formatted phone number of the convention (xxx)xxx-xxxx.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
var
    phone_strg : string;

begin
    phone_strg := get_phone;
end;
```

Variables

toggle.disable_screen

See Also

confirm
get_date
td_readln
get_keypress
td_readkey, td_keypressed

Synopsis

function get_date : string;

Description

This function is a hybrid of the **td_readln** function that will read in a formatted date of the convention MM/DD/YY.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
var
    date_strg : string;

begin
    date_strg := get_date;
end;
```

Variables

toggle.disable_screen

See Also

confirm
get_phone
td_readln
get_keypress
td_readkey, td_keypressed

Synopsis

function confirm(question_strg : string; default : char   ) : Boolean;

Description

Confirm is one of my favorite and most often used extraneous functions. It takes *question_strg* and appends a '? (Y/[N])' to it if the *default* is 'N' (global constant NO), or a '? ([Y]/N)' if the *default* is a 'Y' (global constant YES). Then the user must enter a 'Y', 'N', or [RETURN] for the default.

If the global variable *toggle.disable_user_keyboard* is set to TRUE, the remote keystrokes will be ignored. Otherwise, this function will acknowledge both local and remote keystrokes.

The function then returns a Boolean TRUE if the user selected [Y]es, and FALSE if they picked [N]o.

The global constants YES and NO are provided by TriDoor.

i.e. if confirm('Are you sure you want to explode',YES) then...

would produce...

'Are you sure you want to explode? ([Y]/N) '

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    if confirm( 'Are you sure you want to blow up the earth', NO ) then
       td_writeln( '||||BOOM!||||' )
    else td_writeln( '||||Billions of people say "Thank you."||||' );
end;
```

Variables

toggle.disable_screen, toggle.disable_user_keyboard
YES, NO (constants)

See Also

get_phone, get_date
get_keypress
td_readln
td_readkey, td_keypressed

Synopsis

procedure ansi_clrscr;

Description

This procedure will utilize ANSI graphics screen controls clear both the local and remote screens.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_clrscr;
end;
```

Variables

stats.ansi
toggle.disable_screen

See Also

ansi_left, ansi_right, ansi_gotoxy, ansi_up, ansi_down,
ansi_erase_line, ansi_color

Synopsis

procedure ansi_erase_line;

Description

This procedure will utilize ANSI graphics screen controls to erase to the end of the current line from the current cursor position on both the local and remote screens.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_erase_line;
end;
```

Variables

stats.ansi
toggle.disable_screen

See Also

ansi_left, ansi_right, ansi_gotoxy, ansi_up, ansi_down,
ansi_clrscr, ansi_color

Synopsis

procedure ansi_gotoxy(x, y : integer);

Description

This procedure will utilize ANSI graphics screen controls to place the cursor at the position *x*, *y* on the local and remote screens.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_gotoxy( 10, 12 );
end;
```

Variables

stats.ansi
toggle.disable_screen

See Also

ansi_left, ansi_right, ansi_up, ansi_down, ansi_erase_line,
ansi_clrscr, ansi_color

Synopsis

procedure ansi_left(move : integer);

Description

This procedure will utilize ANSI graphics screen controls to move the cursor *move* spaces left on the local and remote screens.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_left( 10 );
end;
```

Variables

stats.ansi
toggle.disable_screen

See Also

ansi_gotoxy, ansi_right, ansi_up, ansi_down, ansi_erase_line, ansi_clrscr, ansi_color

Synopsis

procedure ansi_right(move : integer);

Description

This procedure utilize ANSI graphics screen controls to move the cursor *move* spaces right on the local and remote screens.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_right( 10 );
end;
```

Variables

stats.ansi
toggle.disable_screen

See Also

ansi_left, ansi_gotoxy, ansi_up, ansi_down, ansi_erase_line,
ansi_clrscr, ansi_color

Synopsis

procedure ansi_up(move : integer);

Description

This procedure will utilize ANSI graphics screen controls to move the cursor *move* spaces upward on the screen.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_up( 12 );
end;
```

Variables

stats.ansi
toggle.disable_screen

See Also

ansi_left, ansi_right, ansi_gotoxy, ansi_down, ansi_erase_line,
ansi_clrscr, ansi_color

Synopsis

procedure ansi_down(move : integer);

Description

This procedure utilizes ANSI graphics screen controls to move the cursor *move* spaces down on the local and remote screens.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_down( 5 );
end;
```

Variables

stats.ansi
toggle.disable_screen

See Also

ansi_left, ansi_right, ansi_gotoxy, ansi_up, ansi_erase_line,
ansi_clrscr, ansi_color

Synopsis

procedure ansi_color(blink, intensity, foreground, background : integer);

Description

This procedure will utilize ANSI graphics screen controls to change the current color on both the local and remote screens. A group of constants has been made available to the programmer to fill in the necessary variables when calling this function. These constants are as follows :

blink	BLINK_ON, BLINK_OFF
intensity	INTENSITY_ON, INTENSITY_OFF
foreground / *background*	BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    ansi_color( BLINK_OFF, INTENSITY_ON, YELLOW, BLACK );
    td_writeln( 'This is solid bright yellow on black.||||' );
    ansi_color( BLINK_ON, INTENSITY_ON, RED, BLACK );
    td_writeln( 'This is blinking bright red on black.||||' );
    ansi_color( BLINK_OFF, INTENSITY_OFF, BLUE, GREEN );
    td_writeln( 'This is solid dark blue on green.||||' );
end;
```

Variables

stats.ansi
toggle.disable_screen
BLINK_ON, BLINK_OFF, INTENSITY_ON, INTENSITY_OFF (constants)
BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA,
CYAN, WHITE (constants)

See Also

ansi_left, ansi_right, ansi_gotoxy, ansi_up, ansi_down,
ansi_erase_line, ansi_clrscr

Synopsis

function read_dorinfo : Boolean;

Description

This function is one of the automatic functions, included with TriDoor, that reads in a on-line door support file, in this case DORINFO1.DEF, and fills in the *stats* record with the needed information for door operation. If the file exists and it successfully read in and converted, the program will return a Boolean TRUE. If the file is missing, the function will return a FALSE.

If your program fills in the *support_path* variable with the path to the on-line door support file, this function will automatically search for it in that directory. Otherwise it will search the current directory.

To see a copy of these procedures and an explanation on how to create your own functions for different door support files see the sections entitled **PROGRAMMING WITH TRIDOOR** and **CREATING YOUR OWN DOOR SUPPORT**.

Example

```
begin
    if ( read_dorinfo ) then
        begin

            if ( setup_tridoor ) then
                begin

                    {* main program here *}

                end
            else writeln( 'TriDoor failed to setup communications port!' );;

        end
    else writeln( 'TriDoor could not find DORINFO1.DEF!' );
end;
```

Variables

stats (record)

See Also

read_pcboard

Synopsis

function read_pcboard : Boolean;

Description

This function is one of the automatic functions, included with TriDoor, that reads in a on-line door support file, in this case PCBOARD.SYS, and fills in the *stats* record with the needed information for door operation. If the file exists and it successfully read in and converted, the program will return a Boolean TRUE. If the file is missing, the function will return a FALSE.

If your program fills in the *support_path* variable with the path to the on-line door support file, this function will automatically search for it in that directory. Otherwise it will search the current directory.

Example

```
begin
    if ( read_dorinfo ) then
        begin

            if ( setup_tridoor ) then
                begin

                    {* main program here *}

                end
            else writeln( 'TriDoor failed to setup communications port!' );;

        end
    else writeln( 'TriDoor could not find DORINFO1.DEF!' );
end;
```

Variables

stats (record)

See Also

read_dorinfo

Synopsis

procedure display_file(file_name : string; line_pause : word);

Description

This procedure will display the file *file_name* and pause every *line_pause* lines will prompt the user for a keypress before continuing. This procedure displays text on both the local and remote screens.

This procedure now acknowledges ANSI screen controls.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    display_file( 'WELCOME.TXT', 23 );
end;
```

Variables

toggle.disable_screen

See Also

none

7.0

Advanced Command Summary

7.0 Advanced Command Summary

The following is a group of commands that are made available, by TriDoor, to the programmer.

These functions and procedures are automatically executed and handled by TriDoor and would only be needed by the programmer if he or she was intending to write an alternate or substitute function or procedure to one of TriDoor's.

record_logon_time

Synopsis

procedure record_logon_time;

Description

This function will take the current time and record that as the user's log-on time and store it in the global variable *logon_time*.

This function is automatically executed by TriDoor in the **setup_tridoor** function.

Example

```
begin
    record_logon_time;
end;
```

Variables

logon_time

See Also

time_remaining, display_time, time_until_timeout
statusbar_refresh, reset_activity_timer, reset_statusbar_timer

Synopsis

```
function  time_remaining : integer;
```

Description

This function will return the number of minutes the user has remaining in their on-line session. In addition to it's primary function, this function also monitors all secondary time-related functions and handles them as is necessary.

This function monitors user inactivity, and if the user exceeds their inactivity time limit, and *toggle.inactivity_timeout* is set to TRUE, the **time_remaining** function will set *toggle.halt_program* to TRUE and display the message stored in *toggle.inactivty_timeout*.

This function will also monitor the user's remaining on-line time. If the user's time has run out and *toggle.automatic_dropout* has been set to TRUE, the **time_remaining** function will set *toggle.halt_program* to TRUE and display the message stored in *message.out_of_time*.

Example

```
var
    time_rem : integer;

begin
    time_rem := time_remaining;
    td_writeln( 'You have' + strg( time_rem ) + ' minutes left online.||' );
end;
```

Variables

logon_time, inactivity_time
message.out_of_time, message.inactivity_timeout
stats.time
toggle.halt_program, toggle.automatic_dropout, toggle.inactivity_timeout

See Also

record_login_time, display_time, time_until_timeout
statusbar_refresh, reset_activity_timer, reset_statusbar_timer

display_time

Synopsis

procedure display_time;

Description

This procedure will display the time the user on-line has remaining in their session. It displays it in the upper-right hand corner of the status bar.

This function can be deactivated by setting the global variable *toggle.disable_time_display* to TRUE.

This function is automatically executed by TriDoor.

Example

```
begin
    display_time;
end;
```

Variables

logon_time, inactivity_time
message.out_of_time, message.inactivity_timeout
stats.time
toggle.halt_program, toggle.automatic_dropout, toggle.inactivity_timeout,
toggle.disable_time_display, toggle.disable_screen

See Also

record_login_time, time_remaining, time_until_timeout
statusbar_refresh, reset_activity_timer, reset_statusbar_timer

Synopsis

function time_until_timeout : longint;

Description

This function determines and returns the number of seconds until the user has exceeded their inactivity time-out limit.

If you ever decide to write your own replacements or alternates for internal TriDoor functions, such as **td_readln** which give the user an opportunity to sit for long periods of time without pressing any keys, it is good to check this frequently.

The easiest way to do this is to simply call on the **time_remaining** function at the end of every wait cycle that the user does not press a key. This function handles *all* time related functions including inactivity and user time depletion.

The number of seconds of inactivity the user is allowed before time-out is stored in the global variable *inactivity_timeout*. The last occurrence of a user keypress is stored, in seconds, in the global variable *activity_log* as the number of seconds from the preceding midnight.

This function is automatically monitored and executed by TriDoor.

Example

```
var
    task_complete : Boolean;

begin
    task_complete := FALSE;   (* assume task is not completed *)
    while ( time_until_timeout > 0 ) and ( not task_complete ) do
        begin
            (* Task that requires user input here.*)
            (* Assigns task_complete := TRUE if user has completed task *)
        end;
    if ( time_until_timeout > 0 ) then
        td_writeln( 'User is out of time.' );
end;
```

Variables

inactivty_time, activity_log

See Also

display_time, record_login_time, time_remaining
statusbar_refresh, reset_activity_timer, reset_statusbar_timer

Synopsis

function statusbar_refresh : Boolean;

Description

This function is very similar to the **time_until_timeout** with two exceptions. Firstly, its purpose is to monitor the time since the status bar had last been changed, and secondly, it only returns a TRUE if it is time to refresh the status bar or a FALSE if it is not.

It is also important to note that after the point at which it returns a true, it automatically resets the counter, so it will return a TRUE every nn seconds. This interval value is stored in the global variable *statusbar_refresh_time*.

This function is automatically monitored and executed by TriDoor.

Example

```
begin
    if ( statusbar_refresh ) then
        display_statusbar;
end;
```

Variables

statusbar_log, statusbar_refresh_time

See Also

record_logon_time, time_remaining, display_time, time_until_timeout
reset_activity_timer, reset_statusbar_timer

reset_activity_timer

Synopsis

procedure reset_activity_timer;

Description

This procedure simply resets the user keyboard inactivity timer. This function is automatically called by TriDoor ANY time a remote user presses a key. TriDoor does NOT monitor the local keyboard, but all higher-level TriDoor functions do.

In other words, if you write your own function that monitors Turbo Pascal's **keypressed** function and then executes a **readkey** the inactivity timer will not be reset, *unless* you execute **reset_activity_timer**. (It is possible that you may *want* to have it set up this way, depending on the circumstances.)

The number of seconds of inactivity the user is allowed before time-out is stored in the global variable *inactivity_timeout*. The last occurrence of a user keypress is stored, in seconds, in the global variable *activity_log* as the number of seconds from the preceding midnight.

This function is automatically executed by TriDoor.

Example

```
begin
    if ( buffer_head <> buffer_tail ) then    (* if ring buffer has an entry waiting*)
        reset_activity_timer;
end;
```

Variables

inactivty_time, activity_log

See Also

record_logon_time, time_remaining, display_time, time_until_timeout
statusbar_refresh, reset_statusbar_timer

reset_statusbar_timer

Synopsis

procedure reset_statusbar_timer;

Description

This procedure is the duplicate of **reset_activity_timer** except that it handles the status bar timer and not the user inactivity timer.

The last occurrence of a status bar reset is stored, in seconds, in the global variable *statusbar_log* as the number of seconds from the preceding midnight.

This procedure is automatically executed by TriDoor.

Example

```
begin
    reset_statusbar_timer
end;
```

Variables

statusbar_log

See Also

record_logon_time, time_remaining, display_time, time_until_timeout
statusbar_refresh, reset_activity_timer

display_statusbar

Synopsis

procedure display_statusbar;

Description

This procedure will display the normal status bar at the bottom of the local screen. This status bar will display the name of your program, stored in *door_name* as well as the name of the user on-line (*stats.name*), baud rate (*stats.real_baud*) of connection and the user's **time_remaining** in the door.

This procedure is maintained by TriDoor and will always be restored after a number of seconds set by the programmer in the variable *statusbar_refresh_time*. It can also be brought up by a local keypress of [F3].

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    display_statusbar;
end;
```

Variables

statusbar_refresh_time,
toggle.disable_screen, toggle.disable_time_display

See Also

display_helpbar_1, display_helpbar_2
statusbar_message

Synopsis

procedure display_helpbar_1;

Description

This procedure will display the first of two help bars at the bottom of the screen with a list of some of the command-keys available to the local SysOp. This procedure is automatically monitored by TriDoor and activated by a local keypress of [F1].

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    display_helpbar_1;
end;
```

Variables

toggle.disable_screen

See Also

display_helpbar_2, display_statusbar
statusbar_message

Synopsis

procedure display_helpbar_2;

Description

This procedure will display the second of two help bars at the bottom of the screen with a list of some of the command-keys available to the local SysOp. This procedure is automatically monitored by TriDoor and activated by a local keypress of [F2].

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    display_helpbar_2;
end;
```

Variables

toggle.disable_screen

See Also

display_helpbar_1, display_statusbar
statusbar_message
, the program will return a Boolean TRUE. If the file is missing, the function will return a FALSE.

If your program fills in the *support_path* variable with the path to the on-line door support file, this function will automatically search for it in that directory. Otherwise it will search the current directory.

To see a copy of these procedures and an explanation on how to create your own functions for different door support files see the section entitled "PROGRAMMING WITH TRIDOOR".

Example

```
begin
    if ( read_dorinfo ) then
        begin

            if ( setup_tridoor ) then
                begin

                    {* main program here *}

                end
            else writeln( 'TriDoor failed to setup communications port!' );;

        end
    else writeln( 'TriDoor could not find DORINFO1.DEF!' );
end;
```

Synopsis

procedure special_key_check;

Description

This procedure is executed when a local function key or special key is pressed. These keys ([F1], [ALT-F4], [PgDn], etc.) actually generate TWO keystrokes, the first one having a value of zero. If you create a procedure or function as a substitute or alternate to one of the TriDoor functions that require a fair amount of time or waits for keystrokes from the local or remote user, you must do the following in order to make sure that these special local keystrokes are still acknowledged :

1> If a key is pressed, check to see if it is a NULL character. (chr(0), **ord** value of 0)

2> If the key that was pressed was, in fact, a null character, call the **special_key_check** procedure. This will automatically handle the keypress and then return to the calling procedure.

3> If it is not a null character, treat it as a normal one.

Example

```
var
    done      : Boolean;
    kb_hit    : character;    {* keyboard character *}

begin                       {* do_some_task *}

    while ( not done ) and ( not exit_door ) do
       begin

          { *** procedure body here *** }

          if keypressed then
             begin

                kb_hit := readkey;
                if ( kb_hit = chr(0) ) then
                   special_key_check
                else

             { ** handle keypress as normal **}

             end;

    end;

end;  { do_some_task }
```

If you do not do this special functions, like status bar help menus, chat mode, and DOS shell will not function within your procedure.

Variables

none

See Also

none

Synopsis

procedure chat_mode;

Description

This is a a built-in, full-feature, multi-color (if the user's ANSI is enabled by having set the global variable *stats.ansi* to TRUE), word-wrapping, chat mode.

Every door you write incorporating TriDoor will automatically have this nice chat mode built right in to it, thus allowing the SysOp to communicate with the on-line user easily and efficiently.

This procedure is automatically handled by TriDoor and is activated by pressing [F10] on the local keyboard.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, **this function will not execute!** **

Example

```
begin
    chat_mode;
end;
```

Variables

stats.ansi
toggle.disable_screen, toggle.disable_user_keyboard

See Also

none

Synopsis

procedure user_page;

Description

This procedure will allow the SysOp to page the on-line user. Yes, you read correctly. Oftentimes the user will walk away from the computer for one reason or another and if you feel the need to get his or her attention, this procedure will alert the user to that fact.

When called, **user_page** will display the messages stored in *message.user_being_paged_1* and *message.user_being_paged_2*, and then beeps until the user answers the page or the SysOp aborts it.

It is not advisable to call this procedure from within the program since it will then bring the user into chat, from which only the SysOp can release them.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, **this function will not execute!** **

Example

```
begin
    user_page;      {* remember, this is not advisable *}
end;
```

Variables

toggle.disable_screen,

See Also

Synopsis

procedure artificial_line_noise;

Description

This procedure will generate a brief spurt of feigned line noise. While this is a bit silly, it is unfortunately sometimes a nice thing to have when you need to get rid of an annoying user without making yourself look bad.

The **artificial_line_noise** procedure is automatically handled by TriDoor and is activated by pressing [ALT]-[F4] on the local keyboard.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, no output will be displayed on the local console! **

Example

```
begin
    artificial_line_noise;
end;
```

Variables

toggle.disable_screen

See Also

none

Synopsis

procedure dos_shell;

Description

This procedure will drop the SysOp to DOS while leaving the user on-line. This is an extremely useful function for any SysOp, but will require the programmer to do some clever memory management.

You must use the $M compiler directive at the beginning of each of your door programs to allow space in memory for both your door and the DOS shell to function. For further information on the $M compiler directive, see the EXAMPLE.PAS program included with TriDoor, and most importantly, consult your Turbo Pascal manual.

TriDoor will automatically determine which shell you are using (COMMAND.COM, 4DOS.COM, etc.) and run that whenever [F5] is pressed on the local keyboard.

The **dos_shell** function is automatically handled by TriDoor and is activated by pressing [F5] on the local keyboard.

** Remember that if the variable *toggle.disable_screen* is set to TRUE, **this function will not execute!** **

Example

```
begin
    dos_shell;
end;
```

Variables

toggle.disable_screen

See Also

none

8.0

Accessible Global Variables and Constants

8.1 ACCESSIBLE GLOBAL VARIABLES

Here is a list of global variables that may be accessed at any time during normal program operation. Remember that the *stats* record must be filled in before you run the **setup_tridoor** function.

For information on how to handle records, consult your Turbo Pascal manual.

All variables with a "*" before them are already defined upon boot-up but may be/may need to be changed to suit your needs. In most cases this is done to certain norms and you may not need or want to alter them during your program. In other cases, it is done as a default so you will at least have something there if you decide to write a quick and simple program.

All variables with a "#" before them are automatically maintained by TriDoor and certain TriDoor functions which you will only have to call upon if you write your own communications functions or alternates to TriDoor functions.

The following function is the *stats* record that must be filled in before calling the **setup_tridoor** function. The functions read_pcboard and read_dorinfo automatically fill these values in from the door support files, DORINFO1.DEF and PCBOARD.SYS, respectively.

```
stats_record =      { ** This is the actual {stats} record type ** }
    record
        name            : string[50];    { name of user currently on-line }
        real_baud,                       { actual baud rate (as seen by user) }
        lock_baud       : longint;       { locked baud rate (comp. to
modm)}
        time : integer;                  { time user has left in door today }
        comstr          : string[20];    { comstring - format 1200,N,8,1' }
        comport         : integer;   { communications port- COM1,COM2, etc...  }
        comp_interrupt,              { communications port interrupt (IRQ) }
        comp_dbits,                  { data bits (7,8) }
        comp_sbits,                  { stop bits (normally 1) }
        comp_parity     : integer;   { parity (ODD,EVEN,NONE) }
        ansi            : Boolean; { ANSI graphics commands enabled if TRUE }
    end;
```

8.1 Accessible Global Variables and Constants

The following is a record of messages that are displayed on both the local and remote screens at certain times during program operation. All of these messages have default settings, but can be changed at any time.

```
message_record =      { ** This is the actual {message} record type ** }
    record                        { Message displayed when...  }
*       enter_chat_mode,          { entering chat mode }
*       exit_chat_mode,           { exiting chat mode }
*       user_being_paged_1,       { user being paged (1 of 2) }
*       user_being_paged_2,       { user being paged (2 of 2) }
*       sysop_aborted_page,       { SysOp aborts user pager }
*       user_answered_page,       { user answers user page }
*       inactivity_timeout,       { user is inactive and dropped out }
*       out_of_time,              { user runs out of time for session }
*       forced_dropout : string;  { user is forced out by SysOp }
    end;
```

The following is a record of Boolean (TRUE/FALSE) toggles which help you to control TriDoor in some case, and help TriDoor keep track of itself in others.

```
toggle_record =    { ** This is the actual {toggle} record type ** }
    record                    { When set to TRUE... }                    {def}
*       td_writeln_echo,      { td_writeln will echo to local screen }   { T }
        user_local,           { on-line user is local }
*#      disable_screen,       { TriDoor will not write to local scrn }   { F }
*       clear_disabled_screen, { clears local screen when disabled }     { T }
*#      disable_queue,        { disables the print queue }               { F }
*#      disable_time_display,     { will not show time in statusbar }        { F
}
*#      disable_user_keyboard, { remote user keyboard strokes ignored }  { F }
*       inactivity_timeout,   { drops out if inactivity limit reached }  { T }
*       automatic_dropout,    { drops out if halting condition is met }   { T }
         XOFF,                { XON/XOFF flow control status register }
*#      halt_program,         { top priority halting condition }         { F }
*#      in_chat_mode : Boolean; { whether or not in chat mode }          { F }
    end;
```

Most of these toggles are all pre-assigned by TriDoor, and some of them (like *toggle.disable_time_display* and *toggle.disable_user_keyboard*) are changed by TriDoor from time to time within the normal operation of your door. The def column are representations of the default values of these toggles when a program using TriDoor is started; T=TRUE, F=FALSE.

The rest of the global variables are just standard variables, not record variables, that may be accessed at any time by the programmer.

```
var
    stats         : stats_record;          { main stat_record r ecord }
*   message       : message_record;          { main message_record record }
*   toggle        : toggle_record;         { main toggle_record record }

*#  direct_xypos : word;                   { screen memory position aft. }
                                           { direct_gotoxy has been called.  }

*#  logon_time      : word;                { logon time of user (in minutes) }
*#  screen_mode     : word;                { screen memory position }
*#  statusbar_refresh_time,                { time until status bar is re-drawn }
*#  inactivity_time          : integer;    { time until user is timed out }
                                            { ( ^both of these are in seconds^ ) }

*#  statusbar_log,                         { record of time statusbar was changed }
*#  activity_log     : longint;            { keeps record of last user keypresses }
                                            { ( ^both of these are in seconds^ ) }

    support_path,                          { path to door support files }
*#  queue_cache    : string;               { queue string for td_writeln procedure }

     door_name     : string[80];           { name of program running }
```

8.2 ACCESSIBLE GLOBAL CONSTANTS

The following is a list of global constants available to the programmer in order to make some tasks a little easier. They cannot be re-assigned like accessible global variables can; they have been pre-assigned to actual addresses and values.

- EVEN, ODD, MARK, SPACE, NONE (parity settings)
- COM1, COM2, COM3, COM4 (comport settings)
- IRQ2, IRQ3, IRQ4 IRQ (settings)
- BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE (ANSI graphics color settings)
- INTENSITY_ON, INTENSITY_OFF, BLINK_ON, BLINK_OFF (additional ANSI graphics settings)
- TDVER, TDDATE (active TriDoor version and date)
- CR, BS (carriage return and destructive backspace)
- YES, NO (used for **confirm** function)
- CAPS_ON, CAPS_OFF, CODE_ON, CODE_OFF (used for **td_readln** procedure)
- ECHO_ON, ECHO_OFF (used for **td_outstrg, td_outchar**)

All of these constants can be used in normal programming. For example:

```
td_writeln('This program is using V'+TDVER+' of TriDoor');
stats.comport := COM1;
stats.comp_interrupt := IRQ4;
```

9.0

Reserved Words

9.0 Reserved Words

The following list of words are reserved (used by TriDoor) and not to be re-defined by any program incorporating TriDoor as a conflict will arise that will result in non-functionality, to some degree, or possibly even a complete lock-up of the final resulting program or software.

tridr55?	tridr60?	TDVER	TDDATE	CR	YES
NO	BLACK	YELLOW	RED	MAGENTA	CYAN
WHITE	GREEN	BLUE	BLINK_ON	BLINK_OFF	CAPS_ON
CAPS_OFF	CODE_ON	CODE_OFF	MAXIMUM	ECHO_ON	ENABLE1
ECHO_OFF	IER	IIR	LCR	MCR	LSR
MSR	RLS	RDA	TRE	MSI	PIC
EOI	PICMASK	ENABLE2	DISABLE1	DISABLE2	BIOSSEG
KEYHEAD	KEYTAIL	EVEN	ODD	MARK	SPACE
NONE	COM1	COM2	COM3	COM4	IRQ2
IRQ3	IRQ4	buffptr	buffrec		

INTENSITY_ON	INTENSITY_OFF	stats_record	pcboard_record
message_record	toggle_record stats	message toggle	direct_xypos
logon_time	comintvec	screen_mode	user_local
XOFF	halt_program	in_chat_mode	inactivity_time
statusbar_log	activity_log	support_path	queue_cache
buffer_top	buffer_head	buffer_tail	oldint exitsave
door_name	comint	setup_comport	setup_parameters
setup_buffer	clear_buffer	tridoor_shutdown	get_display_type
cts_true	set_cts	strg	td_upcase
capitalize	record_logon_time	time_remaining	display_time
statusbar_refresh	carrier_detect	exit_door	setup_tridoor
td_outstrg	td_outchar	td_keypressed	td_readkey
td_readln	td_writeln	td_clrscr	get_phone
get_date	get_keypress	confirm	ansi_gotoxy
ansi_left	ansi_right	ansi_up	ansi_down
ansi_erase_line	ansi_color	ansi_clrscr	direct_char

9.0 Reserved Words

direct_strg	display_statusbar	display_helpbar_1	read_dorinfo
read_pcboard	chat_mode	display_helpbar_2	user_page
dos_shell	display_file	command_file	

statusbar_refresh_time	shut_down_comport	time_until_timeout
reset_activity_timer	reset_statusbar_timer	statusbar_message
artificial_line_noise		

10.0

Acceptable Credit in Programs and Documentation

10.0 Acceptable Credit in Programs and Documentation

Any program compiled using the TriDoor package must give proper credit to Triumph Software and TriDoor in the documentation. This credit must appear on the title page of your documentation, or if such a page does not exist, it must appear with the credits of your program. Credits are such things as "Program written by...", "Program design..." etc. If neither of these things appear in your documentation, then credit must appear within the first three pages of the document. If no document is to be included with your package, a separate file, called README.DOC must be included with the package and within that file the appropriate credit to Triumph Software and TriDoor must be placed.

A disclaimer must also be included in the documentation *immediately* following this credit stating that Triumph Software is not affiliated with you or your company in any way and that Triumph Software can not be held responsible for any programs created using the TriDoor package.

Acceptable minimum credit to Triumph Software and TriDoor is as follows :

TriDoor Communications and On-line Door Driver
(c)1992,1993 Triumph Software, All Rights Reserved.
(508)263-4247 / (508)263-8420

Note that this is the minimal credit required for TriDoor, but we would like to encourage a more verbose credit in your documentation in order to broadcast further about Triumph Software and TriDoor. Remember, the more copies we sell, the cheaper TriDoor becomes and the more revisions and improvements we will make. Help to continue to support the shareware concept!

Triumph Software reserves the right to temporarily revoke registration privileges until proper credit for Triumph Software is given in any software/programs created using the TriDoor unit.

10.0 Acceptable Credit in Programs and Documentation

The following lines will AUTOMATICALLY appear VERY BRIEFLY upon bootup of any program using a registered copy of TriDoor :

> TriDoor Vx.xx - On-line Door and Communications Support
> (c)1993 by Triumph Software, All Rights Reserved.
> Reg No : xxxxxxxxxx Reg To : John Q. Programmer

A special copy of the package without this notice may be obtained by special arrangements with Triumph Software. Different licensing agreements and disclaimers may apply.

Removal of this notice or any of it's information without prior and valid written approval by Triumph Software voids the registration of the TriDoor package and any privileges that go along with said registration.

11.0

Who to Contact

11.0 Who to Contact

Did your machine curse aloud in protest of our software? Are files missing from the release you obtained? Did raging demons erupt from the center of the earth, bursting through your floor, and take away your dog? Did Valkaries cart your brother off to Valhalla to serve as their hero and love slave, never to return?

Have you obtained the latest version of this software?

Well, for some of those problems you may have to contact someone else- (perhaps Ghostbusters™, we can't do EVERYTHING you know!) *However*, we can help solve your Triumph Software problems. If you have difficulties, questions, need a new release of our software, want an upgrade or even just have a bit of feedback for us, please feel free to give us a ring or send us an Internet message.

Triumph Software **Voice Business Line** : (508)263-4247

If we are not at work, please try us at home.

Christopher M. Russo Voice : (508)263-8420
Jeremy H. DuBois Voice : (508)263-7004
(716)274-0227

You may also contact us on the Internet at the following address :

Jeremy DuBois : jer@blaise.cif.rochester.edu

We are also looking out for potential support boards and beta-testing systems, so if you have a system and are interested, make sure you inform us of your desire and eligibility.

12.0

History of TriDoor

12.0 History of TriDoor

TriDoor and it's communications support is the end result of almost four years of effort by both Jeremy DuBois and myself. (Christopher M. Russo) We started one rainy day when Jeremy had informed me that he had managed to write some basic polling Pascal communications routines. As Jeremy already had a bulletin board system running at that time, I suggested that we write an on-line game for it.

And that is when it all began. We went from writing a semi-complete and not-too-awful on-line game to writing DoorBase, our first version of an on-line door supporting unit. Then I wrote a game on my own- Monopolistic Competition, followed shortly thereafter by Monopolistic Competition II which was running off of a newer, but still bedraggled version of DoorBase.

Then one day while trying to write a terminal program, we realized that the communications routines that Jeremy had made were simply not fast enough to handle all the tasking necessary, and at anything above 2400 baud, was a miserable failure. Thus, Jeremy set out once again to create newer, faster, interrupt driven routines.

I had quite a lot of trouble dealing with the newer routines for reasons which, I admit, are beyond my knowledge. After time and perseverance, however, I mastered the new routines and re-wrote the newest version of DoorBase, now called TriDoor in the interest of our newly-founded company.

And thus, I present to you said software and complete documentation for the easy usage and incorporation of communication and door support in your Pascal programs for QuickBBS and clones. I strongly believe that using TriDoor is the easiest, most user-friendly way of writing doors and on-line games available today.