Dec. 01,1993

RsKey Keyboard Emulator

User's Guide

by RiBiG

Table of Contents

1. General

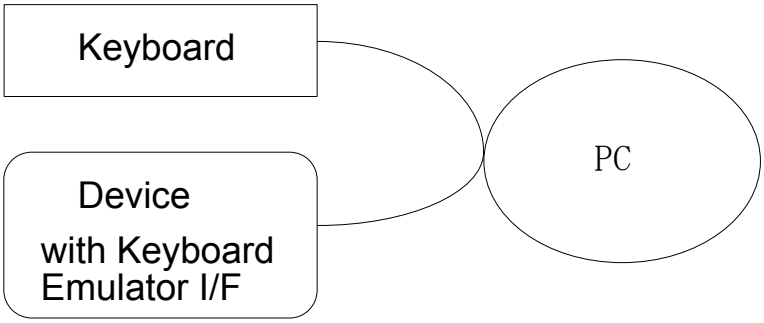
RsKey is a software keyboard emulator in the form of MS-DOS TSR. Once it resides in memory, RsKey enables DOS application programs to receive data directly from an external device connected to PC's RS232c port. It does this by emulating the PC's keyboard firmware in such a way that DOS programs think that the data comes from the keyboard.

With no modification, the existing commercial package programs will be able to support RS232c devices; handy terminal, barcode readers and electric measurement instruments can upload data directly into spreadsheets(Lotus1-2-3, etc) or database(dBase, RBase, Paradox, etc) package application programs. For programs to be newly developed, there is no need to build in RS232c I/O related routines. They can assume that all data come from the keyboard.

Hardware vs Software Keyboard Emulator

RsKey is not just another software keyboard emulator utility. It is designed as a new solution to PC data entry problems.

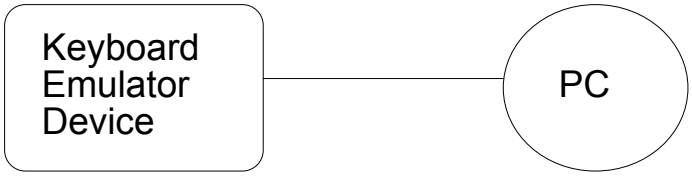
For last several years, the hardware keyboard emulator has been looked upon as a new method of data entry to PC. It uses the keyboard port in transferring data to PC from external devices. The technique has proved advantageous; it eliminated various difficulties traditionally considered inevitable in sending external data to PC through RS232c. Where data length is relatively short, the method is replacing RS232c interface. Today, most barcode readers are equipped with the hardware keyboard emulator interface. The trend of using the keyboard port for the data transfer to PC is still making way.



* From a PC, the keyboard and the keyboard emulator device can not be distinguished.

But the keyboard port of PC is not designed in view to communicating with external devices other than the keyboards. Unless this hardware limitation is recognized, an attempt to undertake serious data communication through the port will be troubled with various unexpected setbacks.

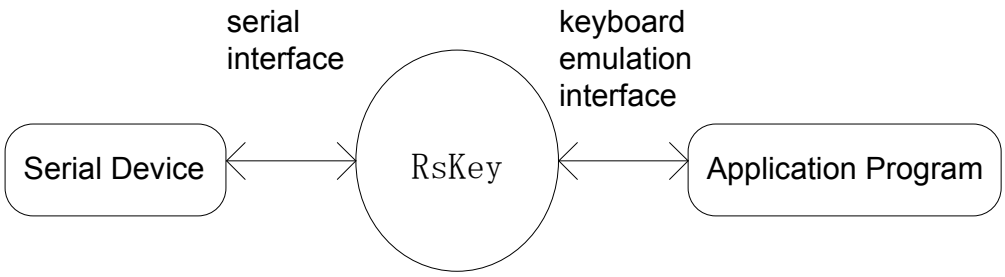
Furthermore, most existing application programs are so designed that they are to accept fixed format data. They can not receive data from a device that can not format its data in application specific ways. This suggests another major limitation of the hardware keyboard emulator as the solution of PC data entry; the mere keyboard emulation of external data is inadequate to carry out general-purpose data communication using commercial packages as a host.



*Data on the device will appear on a PC as they are. It is the application programs that have to re-format / translate data coming in from the device.

*The hardware keyboard emulator is transparent to the PC, but not to application programs running on it. It is the software that users interact with and a device should be transparent to PC software, not to PC.

A new solution is on order; one with flexibility of RS232c without sacrificing the advantages of the hardware keyboard emulator. RsKey is offered as the answer. On one hand, it emulates the keyboard in delivering RS232 data to a foreground application program, transparently. No modification to a host program is required. On the other hand, it is able to perform full-fledged RS232c communication.



*Serial devices communicate with RsKey. It does not see application programs.

*Application programs interact with RsKey. It does not see serial devices.

*Anything can happen within RsKey.

An external device can assume that RsKey is a communication program. It can send data in whatever speed it wishes, using whatever protocol it thinks suitable. A host program only has to assume that data will be entered on the keyboard.

RsKey as Data Entry Front End

In addition, RsKey, in bridging between RS232c data and a host program, can act as a front-end processor to a host program. When RS232c data passes through RsKey, it can process data in ways various application contexts require.

RsKey has no built-in codes to perform any particular data processing. What it offers is the basic keyboard emulator features that are required by all application contexts where RsKey is possibly used. What it does not deliver is a task specific data processing codes. Instead of building in a task unique code, RsKey provides a mechanism to incorporate external programs written for a given task into its own code. Such an external program is called a submodule.

When a submodule is run, it will link with RsKey. Everytime RsKey has received data, it will make a request to a submodule for data process. The submodule will returns processed data to RsKey. Different application requirements are met by preparing different submodules while the basic and important communication and keyboard emulator capabilities are untouched.

Despite many new enhancement features, RsKey is a low cost solution. It requires no hardware. Use RsKey; what else is left is to connect a device to a PC with a cable. It incurs no additional investment.

2. Features

2.1 Basic Features

2.1.1. Full control of RS232c

RsKey provides easy control over all aspects of RS232c, with the on-line help on each parameters.

2.1.2. Keyboard Emulation

RsKey works with many MS/PC-DOS programs, even with those which other software keyboard emulators do not.

2.1.3. Easy setup

No re-booting & turning-off of PC, no cable shuffling... Execute RsKey, anytime before or after connecting a device.

2.2 RsKey's Advantages

RsKey is superior to the other software keyboard emulators on the following points;

2.2.1 Small Resident Memory Size

Its resident size is only 8k(in the standard configuration)

2.2.2 Many Functions

It offers more functions(detailed below)

2.2.3 Application Compatibility

It works with more DOS applications than others

2.2.4 Many Parameters

It has many parameters

2.3 Demerit of Software Keyboard Emulator

The de-merits of the software keyboard emulator as against hardware keyboard emulator are;

2.3.1. OS Dependency

Every software keyboard emulator is OS dependent(for this matter, every program is). For each different OS, one that works under the environment must be written. On the other hand, the hardware keyboard emulators are PC dependent. On the same PC, it works no matter which OS is used.

2.3.2. Application Program Dependency

On MS-DOS, RsKey do not work with those programs which directly peek the keyboard I/O port.

(However, on MS-Windows where no program interacts with the PC hardware and the program style is standardized, the software keyboard emulator for MS-Windows is NOT application program dependent).

2.3.3 MS-Windows Compatibility

RsKey does not work under MS-Windows. A Windows program, wRsKey, is available for the environment. RsKey and wRsKey share the same features. For the details on wRsKey, refer to its manual.

What follows is the description of features unique to RsKey.

2.4 Input Source Management

With RsKey resident, an application program has the 2 input sources, viz., the keyboard and a device connected to RS232c port. If both the keyboard and an external device are active at the same time and they send data to the program at the same time, data from them will be mixed up in a receiving application program.

Since RsKey is forcing the foreground application to receive data from RS232c port where it does not have native support to discriminate the input sources, it is RsKey's responsibility to shoulder the burden to eliminate this possibility.

RsKey circumvents this problem by sitting between the data traffic from the keyboard and from RS232c. When the keyboard entry is under way, RsKey will hold data from RS232c and permit the keyboard to be the sole input source to the foreground application program. Once it starts receiving data from RS232c, the keyboard is disabled.

By setting an option, however, RsKey can give the keyboard a permanent priority over RS232c. Thus, even when RS232c is the current input source, a key press can occasion an input source switch to the keyboard.

2.5 Line Data Processing

RsKey can find line data out of RS232c character stream.

Data transmitted by an external device through RS232c are received by RsKey as characters. Independently, each has no meaning. However, at the sending end, that is, on an external device, before data is sent out, characters are often interrelated and a group of characters as a string usually comprises a significant unit.

For example, on a measurement instrument, a measurement result such as "12.345" bears its meaning only as a string. If it is broken apart into each independent characters "1", "2", ".", "3", "4" & "5", as when sent to a PC through RS232c, it loses its original meaning.

The receiving end has to re-construct an original unit of data out of incoming independent characters. RsKey provides a way to implement it and this is what is meant by "RsKey can find line data out of RS232c character stream". RsKey re-constructs an original string data by locating an agreed-upon data terminator. RsKey temporarily buffers received characters until spotting data terminator character(s), whereupon the buffered characters as a whole are considered an original line (string) data.

The ability to construct a line data(string) is critically related with another feature; data pre-processing function.

2.6 Data Preprocessing by External Submodules

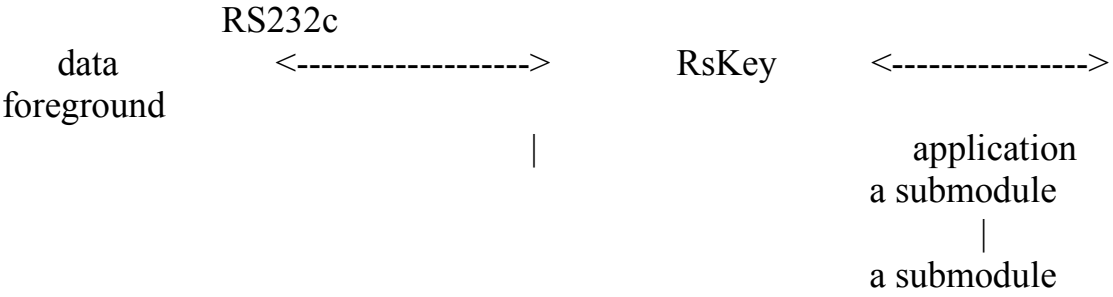
RsKey is a programmable front-end processor. RsKey has a built-in mechanism by which users can configure it to pre-process RS232c data.

RsKey per se does not have any built-in data processing codes; instead, it is equipped with a mechanism whereby users can incorporate processing codes to complete a particular job. This is accomplished by writing sub-

modules (a kind of sub-routine to be called by RsKey to process character / line data) and by dynamically linking them with RsKey after it is in memory.

A submodules takes a form of stand-alone MS-DOS executable file or pure machine code program with an pre-defined program structure like MS-DOS device drivers. The latter type is loaded into RsKey by the machine code loader supplied with RsKey.

All the data that go to a host program pass through RsKey and its sub-modules. Unlike the hardware keyboard emulator, raw RS232c data from an external device are never directly seen by the host program. It only sees RsKey's post-processed RS232c data.



Multiple submodules may be used simultaneously. When more than one submodule are used, each submodule is stacked on the previous one. A top module can be popped(removed) from the module stack. The stacked submodules are chained together to form the "module chain" through which data is passed around.

Use of submodule may be thought of as similar to filter programs in the pipe chain on MS-DOS or UNIX. Each submodule is independent and re-usable. By combining submodules, more complex process is attainable.

Submodules are written in C or assembler. They executes as nearly fast as if they were built-in functions. There is no drastic execution speed deterioration due to use of submodules. In a spreadsheet application program, the input can be formatted by its built-in functions but it visibly slows down the program execution.

It is not necessary to modify a host application program. Often, it is not modifiable as with commercial packages. If modifiable at all, writing a submodule, rather than modifying the host, eliminates the need to go over the host program's source codes(which can be very complex and long) to find out relevant parts and to modify(which often involves in more than one portion) , not to mention the maintenance problems.

A submodule is a re-usable and context-free program that deals with a specific and well-defined task. In writing a module, no attention to the application context is necessary. It permits total concentration on data manipulations in developing a module. Debugging is easy, mainly because most of them are single functional.

The line processing function of RsKey is built in so that a submodule can easily handle the line data. It is the line data that a submodule is often expected to process. Since RsKey re-constructs the line data and submits it to a module, it only has to perform a string operation.

....

To illustrate, suppose that RsKey would receive the line data, from a measurement instrument, like "+ 123.3456" and that you had to,

a. remove the character '+' at the beginning
b. remove all the spaces in the string
c. limit the data to have 2 decimal places.

For this, 3 separate submodules to implement a., b. & c. should be written, rather than preparing one to discharge the 3 jobs. One submodule is to remove '+' character at the head of a string. This will receive a string and check if the first digit of the string is '+' and, if so, remove it. In writing this submodule, it is not necessary to think about how and where it is used. The other 2 modules can be written in the same way. Each of these submodules may be utilized for other purposes in totally different application contexts. Each alone is context-free and not very useful, but when they are combined in a certain way, they will perform a practical application-specific task.

A submodule can be written to filter data, to furnish special software protocols or to set specific characters to emulate certain function keys. RsKey comes with example submodules. Refer to the section on them for what submodules can do.

A submodule can be developed, using the standard MS-DOS software development tools. No special hardware or software is required. Often, having TurboC++ (Borland) is enough. To write a machine code module, only an assembler like MASM(Microsoft) is required.

A submodule development kit is also included with RsKey. Refer to the last section for the details.

[NOTE]

A submodule may be regarded as a binary "object". A submodule "encapsulates" the specific data processing function and it provides an external interface. Because every submodule has the same interface, they can be chained in any order.

Loading a submodule means adding a function to RsKey. All the previous submodules' functions are alive as well as one by a new submodule. This compares well with "inheritance" feature of object programming.

3. Usage

RsKey has many optional parameters and can be configured to accommodate many different application requirements.

3.1 Running RsKey

To run RsKey, type "rskey" at DOS prompt.

```
>rskey[ENTER]
```

When the configuration file for RsKey already exists, the values in the file are read and RsKey will run with its parameters accordingly set. If RsKey can not locate a configuration file, all the parameters will be set to the default.

RsKey will run and stay resident.

3.1.1 Configuration File

The configuration file contains all the optional parameter values. RsKey has many optional parameters. If you have to set values to those parameters each time you run RsKey, it will be tedious. To avoid this, RsKey saves the parameter values in the configuration file and loads the value from the file when run.

3.2 Terminating(Removing) RsKey

RsKey is a TSR. When executed, it will reside in memory. To remove it, use "-r" (remove) option.

>rskey -r[ENTER]

If RsKey is resident, this will remove it. If not, an error message will be displayed.

3.3 Setting Options -- Command Line Option

In running RsKey, optional parameters may be set a value from the command line.

If this is done, RsKey will first read the configuration file, if any, and will set the parameters based on the read values. Then, the command line options are checked. If valid options are specified, RsKey will use the command line values to re-set and override the current parameter values. After the initialization process completes, RsKey stores the new parameter values in the configuration file.

a. If no command line option is used, all the parameter values will remain unchanged.

b. Any parameter value may be changed, using valid command line options.

Here is how to use the command line option.

3.2.1 Option List

All the available command line options may be listed out by displaying the help screen. For this, use "-h"(help) option. Option characters like H must be pre-fixed by "-" (hyphen) or "/" (slash).

>rskey -H[ENTER]

This will displays all the options available. The displayed list of all the options is called the help screen.

RsKey will NOT be loaded when H option is used.

3.2.2 Types of Option

In the help screen, you find 2 types of option;

1. options with "*" asterisk and a value(s).
2. options with "+"(plus) or "-" (minus)

For the former type of options, you need to supply a value along with the option character. The acceptable ranges of values for each option are indicated. The asterisked value is the defaults for a option.

The latter type calls for ON/OFF switch. "+" will turn on the option."-" will turn OFF. In setting these options, either "+" or "-" must be supplied along with option characters.

3.2.3. -O Options vs Non -O Options

Non O options are stand-alone. For example, to set communication parameters and the buffer sizes, a command line option spec. may look like,

```
>rskey  -b9600 -pn -d7 -m1024 -f256[ENTER]

baud rate:9600, parity:none, data bit:7, machine code
```

buffer:1024 bytes, communication buffer:256 bytes

No space between option characters and option values are allowed.

(X) -b 9600 -p 7
(O) -b9600 -p7

O options are those options which are specified with O option character. The command line spec. format is;

>rskey -O[space] options[space] option ...

Example:

>rskey -o line+ beep-
==

They can not be specified without the preceding O option character

>rskey -line -beep (X) NO GOOD.

The following 4 command lines are all valid.

>rskey -b9600 -o line+ beep- -pn -d7 -m1024 -f256[ENTER]

>rskey -b9600 -o line+ -pn -d7 -o beep- -m1024 -f256[ENTER]

>rskey -b9600 -o beep- -pn -d7 -o line+ -m1024 -f256[ENTER]

>rskey -b9600 -pn -d7 -m1024 -f256 -o line+ beep-[ENTER]

3.3.4 Command Line Option Spec Rules

a. Options can be used in any order.

b. -O & non -O option may be used together. A command line below is acceptable.

```
>rskey -b9600 -o line+ -pn -o beep-
```

c. Those options that take + / - are enabled when "+" is suffixed and disabled when "-". "+" can be omitted. RsKey will automatically append "+", if it does not find "-" at the end of a option character.

```
>rskey -o line -i -n GOOD

( the same as >rskey -o line+ -i+ -n+ )
```

3.4 Configuration File

RsKey creates the default configuration file named "rskey.cfg" in the same directory as it is placed, if it does not find one. If it finds one, it will always read the file.

Once any number of parameters are changed from the command line, the changes will be reflected in the configuration file. Next time RsKey is executed, the new parameters will take effect.

The command line options will always override the values read from the configuration file. To change a parameter value, specify a new value from the command line when running RsKey.

You can control how RsKey manages the configuration file:

a. You can prevent RsKey from reading the default configuration file.

b. You can create a configuration file with a given name, so that different parameter setting can be saved in different configuration files.

For these purposes, the option "Q" is provided. In specifying Q option in the command line, you must take care that the option must proceed all the other options.

```
>rskey -Q -b9600        ...        OK

>rskey -b9600    -Q      ...        NG
( -Q must be placed before any other options )
```

3.4.1 -Q -- Ignoring the Default Configuration File

Use -Q option with no value.

```
>rskey -Q
```

This will prevent RsKey from reading the default configuration file. RsKey sets all the parameters to their default values.

The other options can be still set together with Q, as long as they are placed after -Q.

```
>rskey -Q -b4800 -d7 -pe -s2

baud rate : 4800, data bit : 7 bits, parity : even,    stop bit : 2
the other parameters set to RsKey's defaults
```

The default values are found in the help screen.

The command line options specified together with -Q will not be saved in the configuration file. They are effective only as long as the executed RsKey remains in memory.

3.4.2 -Q -- Alternative Configuration File

An alternative configuration file name may be given with -Q option.

```
>rskey -Qnewfile    <other options>
```

- If a configuration file named "newfile" exists in the current directory, RsKey will read the values from it.

- If it does not, RsKey creates a configuration file by that name and sets all the parameters to the default.

- If any other command line options are used, the corresponding parameters will be set accordingly.

If a configuration file is in another directory / drive, supply -Q with a full filespec.

```
>rskey -qb:¥rskey¥newfile -b4800 ...
```

* create "newfile" in rskey directory on B: drive

3.5 Parameter Setup Menu

Instead of entering the options from the DOS command line, the use of the setup menu will greatly simplify the parameter setup.

To display the setup menu, use the option "-X"

>rskey -X+ (or > rskey -X)

From the menu, select the option values. Then, run Rskey from the menu.

Although using the setup menu to configure RsKey looks much different from entering the command line options, they are the same internally. The setup menu is just another way to specifying the options.

Thus, everything mentioned for the command line option is applicable even when you use the setup menu.

a. When -Q is used, RsKey will not read configuration file nor will not save any parameter changes made from the setup menu.

>rskey -q -x

All the parameters will be set to the defaults. Every item in the setup menu will show the default value.

>rskey -q -x -b9600

Each option item in the menu will be set to the default, except for the baud rate which will be set to 9600.

b. Every command line option will be reflected in the menu. When run, RsKey reads the configuration file. If any command line options are specified, corresponding parameters are re-set. Finally, it displays the setup menu with the current parameters.

c. -X option is saved in the configuration file. Once -X is used, the option will remain effective next time RsKey is executed.

>rskey

If X option is previously specified, the menu will appear.

To prevent the setup menu from showing up, -X- must be explicitly specified from the command line. When executing RsKey in a batch file, for example, use -X-.

>rskey -x-

This will guarantee that the setup menu option will never appear, regardless of the configuration file value for X option.

3.6 Help on Options

The explanation of each option in the menu is available on-line. Move the cursor to the item for which a help is wanted and press F1 key.

When a help on an option is on the screen, pressing UP or DOWN arrow key will bring about the help on the previous or the next option.

3.7 Using Setup Menus

The setup menu is easy to use; move the cursor to the option items you wish to modify and press ENTER.

If the option is an ON/OFF type, a press of ENTER key will toggle the option state. If the option takes a value, another prompt will appear to accept it.

a. When an option requires a numeric value, a window with the current value will appear. Use UP / DOWN keys to change the numeric values. Use of SHIFT with UP/DOWN increment the value by 100, instead of 1.

b. If an option takes a value from several alternatives, as with baud rate or data bit, a window menu will appear. Use UP / DOWN to select an item and press ENTER.

c. POSTAMBLE / PREAMBLE options will take up to about 30 characters. An edit box will pop up. Enter a string and press ENTER. If the string contains characters unacceptable to RsKey, the input window remains on the screen even after pressing ENTER. Pressing ESC will discard any changes to the original string and remove the edit window from the screen.

d. Data Terminator option will display 2 input boxes. To move the cursor between the boxes, use RIGHT/LEFT keys. Data Terminator is always required when the line processing option is ON. At least one character must be assigned as data terminator. This means that Box 1 must have a character.

To select a character, use UP / DOWN keys. If data terminator is 1 character long, Box 2 must show "--". Press DEL or keep pressing UP key until "--" will appear. Set a value in Box 2 only when 2 data terminator characters are required.

4. Sub-Module

RsKey's mechanism to incorporate independent sub-modules into its processing routine makes RsKey a flexible PC data entry solution.

4.1 General

The general idea has already been discussed in Section 2.

To re-cap, a sub-module is an external program totally independent of RsKey until it is loaded into memory. Once it is loaded, it is linked with RsKey. After a submodule is linked with (incorporated into) RsKey, it becomes a part of RsKey; it turns into a data process routine within RsKey.

When a submodule is executed, it asks RsKey to register it in a table. When RsKey receives RS232c data, if a submodule is registered, it will request the module to process the data. If no sub-module is used, RsKey will by-pass this step.

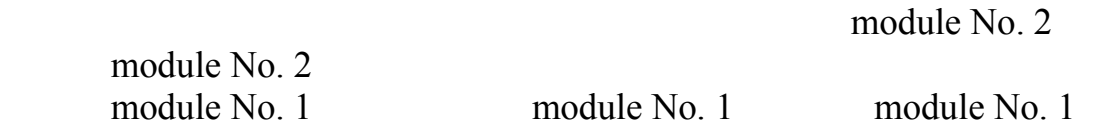
RsKey can run without installing any sub-modules. However, a sub-module can not be executed when RsKey is not resident.

4.2 Submodule Stack

When multiple sub-modules are loaded at the same time, RsKey will form a chain of submodules, pushing one on top of the previous one. In order to remove a sub-module, it must be popped from the top of the chain. A sub-module can not be removed unless it is on the top of the submodule stack.

Submodule Stack

module No. 3



If the submodule No.1 is loaded, RsKey will submit the data to it for processing. When the next module No. 2 is loaded, it is pushed on top of No.1. RsKey asks the top module No.2 to process the data and it is No. 2 that passes the data on to No.1. In the same fashion, No. 3 is stacked on top of No.2. RsKey gives the data to No 3.

To release / remove a submodule, it must be at the top of the module stack. In order to remove No. 2, it is necessary to remove No. 3, first.

4.3 Data Flow among

RsKey submits data for processing to the top submodule which then passes processed data on to the one below it. The last one returns the data to RsKey. A module receives data from RsKey or a previous module. Data it receives may have already been processed by the previous module, but there is no way to know what the original data is. The way the data is processed by the submodules has an important implications; a different order of loading the same set of submodules could produce a different output.

1. original data ------> A ------> B ------> output
"abc" "ABC" Not Processed "ABC"

2. original data ------> B ------> A ------> output
"abc" "RsKey" "RSKEY" "RSKEY"

Assume that;

"A" is a filter submodule to convert the lower-case alphabets into their upper-case.

"B" is a submodule to convert the string "abc"(note that each character is in lower case) to "RsKey".

In passing through 2 different submodule chains, the original data "abc" ends up with totally different strings. In order to effect an intended processing, be sure to use the correct submodule load order.

4.4 Submodule Program

.is an utility to list the submodule names in use in the order they are loaded. If you are not sure of the order the submodules have been loaded, use .. This program is non-resident. Further details will be found later.

4.5 Types Of Module

Modules may be categorized in a few different ways. Two important classifications are based on;

a. the program structure of a submodule

b. the way a submodule is incorporated into RsKey

4.5.1 Program Structure Classification

The modules may be divided into 2 groups by the way they are written and compiled. It is easy to tell one from the other by examining the program file of a submodule.

4.5.1.1. Stand-Alone Module

A module may be a stand-alone MS-DOS executable program(.). It can be run by typing in its file name from MS-DOS command line. This module is a standard MS-DOS TSR

4.5.1.2. Machine Code Module

A submodule may be a pure machine code program whose structure is in accordance with that defined by RsKey's technical manual. It usually has the extension of .BIN. The machine code loader program supplied with RsKey loads this type of module into RsKey's internal machine code area.

The stand-alone module is a MS-DOS TSR. It is usually written in one of high level languages(usually C). This type is suited for writing a module to perform more complex processing tasks. A submodule of this type consumes more memory than the other type.

The machine code submodule is written in assembler. It must have a pre-defined program structure so that the machine code loader can correctly load and release it. The program structure is very simple; writing a submodule of this type is much easier and less time consuming than writing a DOS TSR. For a simple task such as converting a string or filtering characters, this type is best suited. The memory it occupies is minimal. With less than 500 bytes of additional memory, RsKey can have a string filter function.

4.5.1.3. Executing

These two modules are executed, differently.

1. Stand-along Type:

Since stand-alone module are MS-DOS .file, it can be run in the same way as the other MS-DOS programs. Most submodules of this type will require the multiple command line options. To remove a module, use -R option.

2. Machine Code Type

Machine code modules must be run (loaded) by the machine code loader, LOADBIN..

When running(loading) the sub-modules, they must link with RsKey. No manual operation is involved. The stand-alone submodules will link themselves with RsKey. The machine code modules will be linked to RsKey by the machine code loader.

While a module is resident, do not remove RsKey. It could be released when only machine code modules with no cleanup routines are in use. Depending on submodule in use, this could cause unexpected critical errors(which usually ends up with the system hang-up).

3.Loading The Machine Code Module

To load a machine code module, RsKey must be executed with the machine code buffer option set. Unless RsKey reserves the machine code memory area, an attempt to load them will fail. Every machine code module is loaded into the area RsKey reserves. The area must be bigger than the total sizes of the submodules to be loaded.

>rskey -m1024 allocate 1k of the machine code buffer

To find out how much of memory is required by each machine code submodules, run LOADBIN.machine code loader with L(LIST) option.

> loadbin -F<filename> -L

It will display the details on <filename> submodule. The actually required space could be larger than the displayed value by up to 16 bytes.

Execute LOADBIN.with -F(file) option to load a machine code module.

>loadbin -F<machine code module name>

The use of -L(list) option will display the details of the specified module before prompting to proceed with the module load;

>loadbin -F<module name> -L

A machine code submodule can take the command line options. Use -O option character.

>loadbin -F<module name> -O"-n -t"

In this case, the string "-n -t" will be passed to the submodule as the command line option.

To un-load a machine, use -R(remove) option.

>loadbin -R -F<module name>

A submodule name may be omitted, if the module is in the current directory or those set in PATH environment variable. LOADBIN.is able to get a submodule name(to save the memory area, not a full file spec., but only the 8 character long file name) from RsKey and tries to locate the file by the name. In case a submodule is not in the current or in any of those set by PATH, be sure to supply its full file spec. with -F option.

At the time of loading a submodule, LOADBIN will not load any unnecessary parts of the submodule such as initialization and clean-up routines. In unloading a module, LOADBIN must load the cleanup routine, if any. If LOADBIN can not locate a file supplied by RsKey or from the command line, an error will be reported.

4.5.2 How a Submodule is Incorporated into RsKey

The modules may be classified by the way they are incorporated into RsKey.

In this classification, the modules are numbered. There are 6 different types(0 to 5).

To know the type of a machine code module, run LOADBIN.with L(list) option. One of the listed item is the submodule type. For stand-alone submodules, there is no way to know their types.

The users do not have to know this module classification. For those who have to maintain submodules, the type is a useful key to knowing what functions they have. For the module developers, the type is of critical importance because the choice of a submodule type determines the functions it can have.

4.6 .& Unloading of Modules

The stand-alone and the machine code modules can co-exist. Once they are loaded, they look the same to RsKey. It can not distinguish one type from the other.

To list out the names of the loaded sub-modules in the order they are loaded, use ..

```
>listsub
```

The program will display the module names as follows;

1. AAA
2. BIN1*
3. BBB
4. BIN2*

The machine code names have the asterisk appended.

Only the last module(one on the top of the module chain)can be released. In this case, only BIN2* can be removed. Since BIN2 is the machine code module, use LOADBIN with -R option.

>loadbin -R <-Fbin2.bin> (-F is optional)

Then release BBB. All stand-alone submodules can be released with R option.

>bbb -R

Then,

>loadbin -R <-Fbin1.bin>

A submodule may be loaded at any time you wish to do so.

[WARNING]

When a machine code module is in use, be careful not to unload one when the other modules are on top of it.

1. AAA
2. BIN1*
3. BBB

BIN1 could be removed even when BB is on it. As with the load order, the unload order is important.

5. Line Processing

Four (4) associated options must be set correctly for RsKey to perform the line processing.

a. Line Processing Option

b. Data Terminator

c. Line Buffer

d. Inter-character Input Timing

These options determine how RsKey treats RS232c data. If RsKey exhibits unexpected behaviors while the line process is enabled, check a. - d. option settings. One of them is often their cause.

There are 3 other options which are related with the line processing; SUFFIX, POSTAMBLE and PREAMBLE. They serve useful auxiliary line process functions as described below.

5.1 Line Processing Option

Unless this option is enabled, RsKey will not attempt to construct a line data out of RS232c data stream.

If it is turned on, RsKey will assume that each received character belongs to a line data(string) and will try to find data terminator character(s) set by Data Terminator option.

Note that enabling this option alone does not guarantee the proper line processing operation. The other 3 options must be correctly set.

5.2 Data Terminator Option

The data terminator character(s) signifies the end of a line data.

An external device appends one or two suffix characters to the end of every line data. The device may have the option to set the suffix or it may have the fixed suffix. Consult its manual. On many devices, CR or CR/LF is used as the suffix.

< line data > + <suffix >

"abcdefg" + CR or "abcdefg" + CR/LF

When RsKey is able to locate the character(s) used as the suffix by the sender, it can take out the line data out of the RS232c character stream.

In order for RsKey to do it, Data Terminator option must be set to be identical with the sender's suffix. If it is CR, set the option to CR.

Up to two characters may be set as the data terminator. No device is expected to use a suffix of more than two characters. In setting the data terminator option, there is no choice involved; it just has to match the sender's suffix.

If the data terminator option is set incorrectly, RsKey exhibits some very strange behaviors. Since the data terminator is not generally thought causing those problems and it is not associated with a particular type of misbehavior, it is overlooked in searching a cause of error. This is especially so when various options are set and multiple submodules are used. When the line option is turned on and strange behavior is encountered, be sure to check this option first.

Upon spotting the data terminator character, RsKey removes it from the line data

< line data > + <sender's suffix> -----> <line data>

"12345" + CR/LF "12345"

Then, the line data is appended the character set by Suffix option.

<line data> -----> <line data> + <RsKey's suffix>

"12345" "12345"+ENTER

No matter what suffix characters a sender uses, they will not be taken in by host application programs.

Suffix vs Data Terminator

Both terms refer to the characters appended to the end of line data.

The appended character is called SUFFIX from the view point of those which actually append characters. From the other side which extracts the appended data, the characters are called DATA TERMINATOR

Sender Side Receiver Side

<line data > + CR/LF ----------> <line data> + CR/LF

CR/LF = Suffix CR/LF = Data Terminator

For instance, the external serial device appends SUFFIX characters and RsKey tries to locate DATA TERMINATOR (SUFFIX character(s) is identical with DATA TERMINATOR).

external serial device RsKey

<line data > + CR/LF ----------> <line data> + CR/LF

Or RsKey appends ENTER key as SUFFIX.

RsKey		application program
<data>+ENTER	---------->	<data>+ENTER

RsKey's DATA TERMINATOR characters must be the same as external devices SUFFIX character. RsKey's SUFFIX character is what an application program expects to know the end of line input(sometimes, referred to as buffered input); it is usually ENTER, but some programs use TAB or cursor keys.

5.3 Line Buffer

RsKey uses the secondary buffer called the line buffer in constructing a line data. When the line option is on and a character is received, RsKey places it in the line buffer. When RsKey finds the data terminator, the buffered characters are taken out as a line data.

	line buffer	
A ------->		A
B ------->	AB	
C ------->	ABC	
D ------->	ABCD	
E ------->	ABCDE	
F ------->	ABCDEF	
CR ------>		-----> ABCDEF

*CR = DATA TERMINATOR

The size of the line buffer is adjustable. If RsKey expects to receive long line data, the buffer must be large enough to hold the longest line data. If it expects to receive short ones, reduce the size of the buffer. This will make RsKey's resident size smaller.

For most data collection & instrument devices, the default of 128 bytes is long enough.

When the line option is OFF, the size of the line buffer can be set to 0.

5.4 Inter-Character Input Timing

The inter-character input timing option serves 2 purposes;

1. Recovery from a deadlock state
2. Parallel use of character-based and line-based processing

5.4.1 Recovery from DeadLock

A program which works in the background must provide for a method or another to escape from potential deadlock states. Once a background program comes to deadlock, the whole PC system will stop. On a multi-task environment, such a background program can be easily killed. OS supports it. On a single task MS-DOS, the only remedy is to reset the PC unless the background program has a way to escape from the state.

RsKey is a background program and it could cause deadlock. RsKey's features are such that it is inevitable.

- When the line option is enabled, RsKey buffers the RS232c characters to construct a line data until the data terminator is received.

- In managing the input sources, RsKey permits either the keyboard or RS232c port to be the sole input source to a foreground program. While RsKey is buffering RS232c data to form a line data, the keyboard is disabled.

Once RsKey begins to receive RS232c data, it disables the keyboard and waits for the data terminator. What would happen if no data terminator character is received in this condition -- deadlock. The PC system is alive, but RsKey prevents the foreground program from accepting the keyboard entry.

a. One solution to escaping from this state is to send the data terminator manually. But this is too crude and, in practice, it will not be an acceptable one.

b. A more sophisticated solution is to permit a press of a key combination to reset RsKey's state so that it enables the keyboard again. This is better than the above one but still requires a manual operation.

RsKey adopted another method. It automatically recovers RsKey from the deadlock state.

c. The method uses the timer to measure the time interval between an input of a character and the next while RsKey is composing a line data. Given a maximum time interval, if the measured time interval is less than that maximum, the latest character is placed in the line data buffer. If no character comes before the maximum interval expires and the line data buffer holds data, every single character in the buffer will be flushed as a character data. RsKey will not endlessly wait for DATA TERMINATOR.

More specifically, here's how RsKey forms a line data. Let's assume that the max. time interval was 0.5 sec. Note that the line option must be ON.

1. Initially, the line buffer = ""

If the line buffer is empty, the next incoming character is unconditionally put in the line buffer. Here suppose that "a" is received.

2. line buffer = "a"

If the next character "b" comes within 0.5 sec., it will be placed in the line buffer.

3. line buffer = "ab"

Assume that "c","d", and "e" come within 0.5sec after the previous character.

4. line buffer = "abcde"

If the data terminator CR is received, RsKey will perform the step 5.

If the max. interval expires before the data terminator is received, RsKey will perform step 6.

5. line buffer = "abcde"+CR

RsKey will take out the line data from the line buffer and submit the line data to the top submodule. RsKey also empties the buffer. RsKey's state will become the same as 1.

6. RsKey will take out "abcde" and break them apart to "a","b","c","d" and "e". RsKey gives each character to the top submodule as the character data, sequentially, not line data. The submodule will process each character as the character data. After every character is output, the buffer is emptied and the next line data construction starts(that is, go back to 1).

From this, it is seen that,

a. The 4 options, Line, Data Terminator, Line Buffer and Inter-Character Input Timing, are related with one another.

b. Each option is critical for RsKey to perform the line processing correctly.

c. A submodule could receive character-based data, even if the line option is turned on. (A submodule can decide whether it processes the character-based data when the line option is ON).

5.4.2 Parallel Use of Character-based and Line-based Processing

The use of the timer in forming the line data, originally provided to avoid deadlock, serves another function. It is not an important one, but it is still worthwhile to take a note of it.

As noted in the previous section, even when the line option is enabled, RsKey can still handle the character-based data when the inter-character interval is greater than the given maximum time.

This fact can be taken advantages of; when the line option is ON, RsKey can still be made to handle received data as the character-based data by sending each character in such a way that the input interval between characters is longer than the maximum time interval.

But,

a. The parallel use of the character-based and the line-based processing is difficult to put into practice unless an external device can measure time.

b. Unless carefully planned, it is advised not to use the character-base and the line-based in parallel. Stick to one mode and make sure that the other mode will not become effective.

If RsKey's working is understood and there is a good reason to use this feature, do it but only wisely. Do not forget that it will be difficult to support such an application if you develop one.

5.5 Suffix

Upon composing a line data < LINE DATA>, RsKey will

a. Send it to the top module in the module chain.

<LINE DATA> ----> submodule ----> <PROCESSED DATA>

b. Append the preamble and the postamble strings to the processed line data

<PREAMBLE>+<PROCESSED DATA>+<POSTAMBLE>

c. Finally, append the suffix character.

<PREAMBLE>+<PROCESSED DATA>+<POSTAMBLE>+<SUFFIX>

Note that it is the processed data(one that the submodule chain returns) that the preamble, the postamble and the suffix are appended to, never to the raw data.

The final data format will become

<preamble> + processed line data + <postamble> + <suffix>

SUFFIX usually refers to a character(s) that is appended to the end of each line data. In RsKey, Suffix is the keyboard key to be emulated at the end of each line data.

< line data > + suffix

"12345" + CR(= ENTER)
"12345" + Right Arrow

Every program defines a key to exit from the edit job. In case of a spread-sheet program, after inputting the data in a cell, TAB or an arrow key will move the cursor to another cell while ENTER will only "enter" the data to the cell. In case of most other application programs, pressing ENTER will advance the cursor to the next field as in a database application. Unless a key to exit from a edit job is pressed, the data is not really "entered" to the program.

Suffix is used to provide such an exit key emulation. RsKey offers 6 default suffix keys; CR(=ENTER), TAB, UP, DOWN, RIGHT, LEFT. If no such key is required, select NONE.

Internally, these function keys are represented by the respective ASCII characters.

BS	=	chr$(8)
TAB	=	chr$(9)
ENTER	=	chr$(13) (= chr$(&h0d))
UP	=	chr$(28)
DOWN	=	chr$(29)
LEFT	=	chr$(30)
RIGHT	=	chr$(31)

5.6 Preamble / Postamble

The preamble and the postamble are optional strings that may serve various purposes. The preamble string is prefixed to a line data while the postamble is appended to it. By default, the null string is assigned to them.

The preamble/postamble strings can consist of any ASCII characters , except for chr$(0). It must be less than 32 characters. The unprintable characters are specified by the use of the backslash character in the format of "¥xx" where xx is the hex number of an ASCII code. For instance, "¥10" is the same as chr$(&h10).

The postamble is sometimes used in lieu of the suffix. If, for example, the postamble string is set to "¥0d¥0d¥09" and the suffix is set to NONE, every line data will become

<line data> + "¥0d¥0d¥09"

Since chr$(&h0d) emulates ENTER and chr$(&h09) TAB, the postamble string will have the same effect as pressing ENTER twice and then TAB after inputting the data. If the default suffix keys are not adequate, use the postamble.

If the postamble string is "¥0d" and the suffix is set to NONE, it will be the same as setting the suffix to CR.

6. Input Source Management

RsKey controls which device, the keyboard and the RS232c port device, is to be the sole input source to the foreground application program. Because of this mechanism, the data from one source will not mingle with the data from another.

RsKey manages the input sources as follows;

1. when no data is coming from the RS232c port, the keyboard is always the input source to the foreground application.

2. even when the data from an external device is received, the keyboard remains the input source if the key entry is being made on it.

3. when the RS232c device becomes the input source, it will be the sole input source until there is no data coming from the device.

RsKey disables the keyboard or the RS232c device but this does not mean the disabling of the hardwares. The keyboard and the RS232c hardware are alive. "Disable" means that RsKey prevents their data from being passed on to the foreground application program. To avoid conflicts with other programs, RsKey tries not to use hardware/firmware features where software solutions are available.

6.1 Switch from Keyboard to RS232c

The input source is switched from the keyboard to RS232c if no key entry is being made when the data arrives from the RS232c.

In order to determine whether a key entry is being made or not, RsKey is measuring the time of each key press. This allows RsKey to know when the last key press has taken place and how long ago.

When RS232c data comes, the keyboard state can be one of the followings;

a. A key is being pressed

b. The last key press was made within a given time interval

c. Since the last key press, a given time interval may have expired already.

If a. or b. is the case, RsKey will permit the keyboard to remain the input source. In case of b., RsKey holds the RS232c data until the keyboard state shifts to c.

When there is RS232c data in RsKey, the moment the keyboard state becomes c., the input source switch to the RS232c takes place.

The time interval during which RsKey waits for a key press after the last one is adjustable. It is one of RsKey's parameters. For this, use -E option from the command line. Since this parameter is not expected to be frequently changed, no menu item for it is provided. The default for E option is 9(0.5 sec). 1 unit is equal to 1/18 sec.

```
>rskey -E18

RsKey will waits 1 sec for the next key press
```

6.2 Switch from RS232c to Keyboard

The keyboard can be given a permanent priority over the RS232c by turning on "Keyboard Always On" option. When this option is enabled, an key

press will IMMEDIATELY trigger the input source switch to the keyboard, regardless of the RS232c status.

If the option is OFF as often is the case, the switch to the keyboard does not immediately take place. When a key is pressed, RS232c status can be one of the followings;

a. A character is being received

b. The last character arrived within a given time interval

c. Since the last character, a given time interval may have expired already.

Just like the switch from the keyboard to the RS232c, the switch to the keyboard will happen only if c. state appears. Any key press under a. or b. states will be ignored(not saved anywhere).

How long RsKey waits for the next character is a parameter to be set by -J option. The default for J option is 18(1 sec). 1 unit = 1/18 sec.

```
>rskey -J32
```

RsKey will waits 2 sec for the next RS232c character with the keyboard disabled.

No menu item is provided for this option.

7. Submodule Examples

Several submodules are supplied together with RsKey. They are called the standard modules. They are released in the hope that they demonstrate the scope of the submodule's capability to extend RsKey's functions.

The standard submodules range from a very simple string filter to a more complex pre-processor. A few such as "SETKEY.BIN" and "XMODEM." are general purpose submodules. A few others such as "FILESUB." may be modified to meet a specific application requirement. The rest such as "CONV.BIN" are provided for demonstration purpose only.

Depending on how a submodule is incorporated into RsKey, it is classified into several types. In writing a submodule, choosing the submodule type(this is the same as how it is to be incorporated into RsKey) determines what functional extension it can deliver. The standard submodules are written in various types. In using a submodule, it is not necessary to know a module type. It is not indicated anywhere. But note that, although all of them are called submodules, their supportive features could be quite different, according to their types.

7.1 Machine Code Module

Before loading a machine code module, be sure to execute RsKey with the machine code buffer option set.

7.1.1 Toupper.bin

Description:

This is a module which converts the alphabetic characters into their upper case. While this module is resident, all the alphabets will be turned into the upper case.

Load:

>loadbin -ftoupper.bin

To check if it has been successfully loaded, run ..

Release:

>loadbin -r [-ftoupper.bin]

7.1.2 Conv.bin

Description:

This is a demo program with no practical use in mind. It maps the numeric characters into another according to the following rule;

```
0  1  2  3  4  5  6  7  8  9
         |  |  |  |  |  |  |  |  |  |
9  8  7  6  5  4  3  2  1  0
```

"12345" ------> "45678"

Load:

>loadbin -fconv.bin

Release:

>loadbin -r [-fconv.bin]

7.1.3 Setkey.bin

Description:

This is a general purpose submodule that provides RsKey with user-definable function key emulation capability.

By default, RsKey supports the emulation of BS, TAB, ENTER, ESC and 4 arrow keys. This submodule supplements RsKey's function key emulation.

The module affords users to set up an unique ASCII code for a particular function key emulation. Users are free to assign any ASCII code they see fit to a function key. For instance, F1 key may be emulated by chr$(1), chr$(22) or any other codes.

After this module is installed, a code assigned to a function key is used to simulate the function key press.

Load:

>loadbin -fsetkey.bin

Release:

>loadbin -r [-fsetkey.bin]

7.1.3.1 Mapping Table File

The link between an ASCII codes and a function key is established in the text file named "KEYTABLE.DEF". This file serves as the mapping table. In it are the headings of various function keys. By placing an ASCII code in a line for a function key to emulate, the code is mapped to the function key. If a line is not filled in with a code, the corresponding function key is not emulated by any codes.

- If chr$(1) is to emulate F1 key, place "01"(2 digits in hex) in the blank line under F1 heading.

- To map chr$(&h0f) to PageUp key, put "0f" in the blank line under PageUp heading.

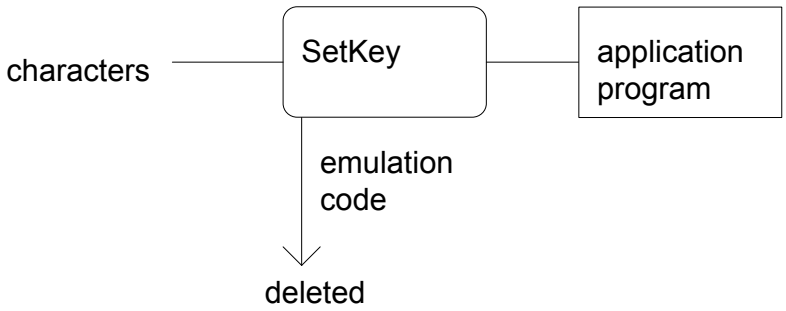
- An ASCII code must be specified in two digit hex.

- Do not assign any codes to those function keys that do not have to be emulated.

- To place a comment, use a semi-colon at the start of a line.

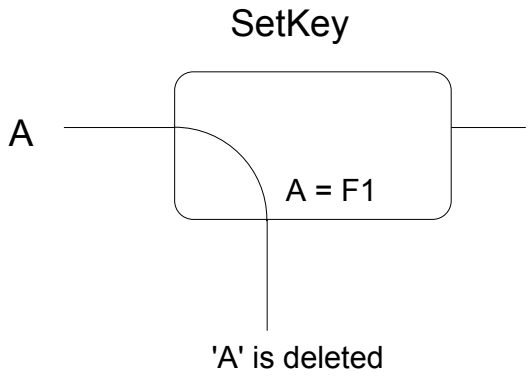
When the module is loaded, it will look for the mapping table file in the current directory. If it does not find one, it will report an error. If it does, it will read the file.

Every character that goes to a foreground application always passes through SETKEY.BIN after the line-based or the character-base processing have been performed. Each is checked out by this submodule to be an assigned emulation code. It it is, the designated function key is emulated and the code is discarded.



7.1.3.2 Function Key Emulation Code

A code set up to emulate a function key is not passed to the foreground application as a character. It is used only to simulate the press of a function key. For this reason, avoid setting up a printable character as an emulation code.



* 'A' is used to emulate F1. 'A' as a character will never be output and the foreground application will never receive 'A'

Use ASCII codes in the ranges between 0 and 32 and between chr$(&H80) and chr$(&H9F) for the function key emulation. Any other characters may be used if you are sure that your application program do not expect them.

RsKey will filter out(discard)any ASCII codes below 32, except 8, 9, 13, 27, 28, 29, 30, 31, just before they are given to the foreground program.

When a software protocol is used, be sure not to assign those characters used by the protocol. For example, if XON/XOFF flow control is effective, do not use the codes for XON/XOFF characters.

Suppose 'a' is used to emulate F10. This submodule will simulate F10 everytime 'a' is received and then the character is discarded. This means that 'a' will never goes to the foreground application programs.

The rule is - if a given ASCII code is used to serve any other function, try not to use it to avoid conflicts.

7.1.3.3 Emulatable Function Keys

This submodule supports the emulation of the following keys;

F1 - F12, Insert, Delete, Home, End, PageUp, PageDown
BS, TAB, ENTER, ESC, 4 Arrow keys

7.1.3.4 Other Notes_

- The submodule will scan the codes used in the mapping table from the top, in the order they are used. If a same code is set up for different function keys, say, F1 and F10, the code only emulates the first matched function key; in this case, F1.

SetKey will sequentially check a match from the top

```
----->
F1     ...   F10     ...
01               01
```

If it finds that 01(= chr$(01)) is assigned for F1, it will stop the check there. 01 for F10 will never be checked by SetKey.

- The submodule provides a way to re-assign the codes for the function keys RsKey supports by default(BS, TAB, ENTER, ESC, 4 Arrow keys). If, for instance, F1 is assigned chr$(27), it will always emulate F1, although chr$(27) is reserved to ESC by default. In a case like this, give another code to ESC, say, 20 in the mapping table file.

7.1.4 Setstr.bin

Description:

This is a general purpose submodule. It is the only machine code example submodule that accepts the command line options.

This is what this module does;

RsKey ---> the line data ---> this module ---> the processed line data

|
|

a. Using the line data passed from RsKey, it searches a database file to find a matching record.

b. If it finds one, the original line data is replaced by the string in the database file.

c. Each record in the database file has 2 fields, one for a key and another for a string.

"ABC" "ABC is replaced"

If there is a record like this and RsKey receives the line data "ABC", the data is replaced by "ABC is replaced" and the foreground application receives the latter string.

In short, the submodule is a simple database front-end processor.

7.1.4.1 Database File - "SETSTR.DBS"

Before running this module, a database file must be prepared. The file has to have the name "SETSTR.DBS" and be placed in the same directory as this submodule.

Any number of a pair of strings may be written in the database file. The first string is used as the key field. The second string is the substitution string.

The file can be created by a text editor.

<key string><more than one tab or space><substitution string>

key1 string1[ENTER]
key2 string2[ENTER]
...
...

This submodule will replace every occurrence of "key1" with "string1".

7.1.4.2 Loading the Database File into EMS

When the module is executed, it will read the database file "SETSTR.DBS" in the current directory and load it into EMS memory. If no EMS memory exists or necessary EMS pages can not be allocated, an error will be reported.

It is this EMS file that this submodule searches through, not the disk file. The maximum size of the database file is determined by that of the EMS memory available.

7.1.4.3 Searching Through the File in EMS

This submodule performs a scan through the EMS file from the beginning to the end. When it finds a match, it modifies the original input line data and passes the processed data on to the next module or RsKey.

If a match is located toward the top of the file, the time required for the search is relatively shorter than if toward the end of the file. This is particularly so when the file contains many records.

The submodule scans the entire file, in vain, when there is no matching key. Since every line data passes through this module, RsKey's overall performance dramatically declines when the line data with no match in the database file is read from an external device. To avoid this, various options to control the searching capability are provided; for instance, the submodule can be made to conduct a search only when the input line data meets one condition or another.

7.1.4.4 Warning

If the database file contains many entries and the data comes in continuously from the RS232c, RsKey may fail to receive the line data correctly. It can lose a character or so in a string.

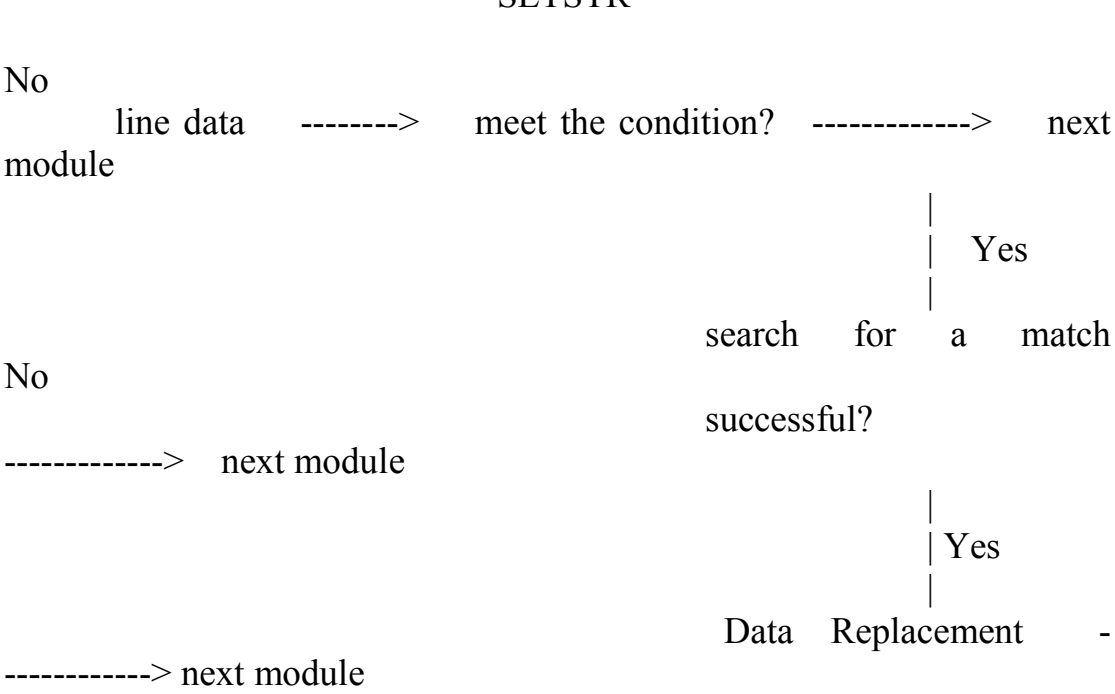
Before using this module, make sure that RsKey is not losing a character in a string data and check if it performs the way an application context demands.

If your application is such that an external device does not send out data until the foreground program receives the data completely, this submodule may be safely used. However, if a device sends out data while this submodule is searching the database, please check carefully before use.

7.1.4.5 Options:

By default, the module will conduct the linear scan over the database file for every line data Rskey submits to it. But this unconditional search is inefficient when the line data is known to have no match in the database file before the search.

The submodule's searching mode may be controlled by supplying it with the command line options. The options compel the module's search to be conditional upon the received line data. Only those line data that meets a specific condition will be looked up in the database file. The others will be passed over.



-S Option

SETSTR.BIN will conduct the search only if the line data begins with one of the characters identified by this option.

-S1234567890

* The submodule will accept the line data starting with a numeric character for the search.

The non-printable characters can be set in hex in the format of "¥xx".

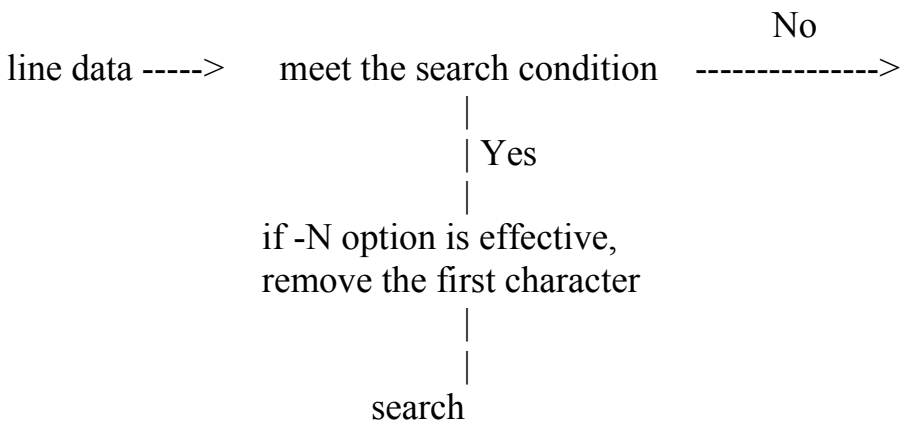
-S¥01¥05

* The line data must begin with either chr$(&h01) or chr$(&h05) to be processed by this submodule

-N Option

The first character will be removed from the line data just before the linear search starts.

The line data must meet the search condition for this option to be effective. The submodule first checks whether to search the database file using the line data. Only if the data is a target data of this submodule, will it remove the first character from the line data.



Example:

-S0123456789 -N

If a line data begins with a numeric character, this module will perform the search. But, just before it tries to search, the first

character will be stripped off.

0ABCDEFG ----> ABCDEFG

* If "ABCDEFG" is in the database file, the corresponding string will replace it.

123456 -----> 23456

* "23456" is used as the search key.

-N option is always used with -S option. Every often, -S option is accompanied with control characters.

-S¥01 -N

* Any line data starting with chr$(1) will be searched.

chr$(1)+"ABCDEFG"

The actual string to be searched will be "ABCDEFG". Prefixing the line data with chr$(01) will make it the target data of this submodule.

In the database file, there is no need to place chr$(1) at the beginning of every key field data.

We advise not to use -N option without using -S option.

-D Option

If SETSTR.BIN finds no corresponding record in the database file, the string specified with -D option will replace the original line data.

Example:

-S¥01 -N -DNG

* If the incoming data is CHR$(1)+"ABCD", the submodule will perform the search. Just before initiating searching, it removes the first character CHR$(1). A matching data is found in the file, the line data is replaced by the file data. If not, the string "NG" will replace the original data.
If -D option is not used, the module's output will be "ABCD" with the first character CHR$(01) stripped off.

-S¥01 -N -D

If no character is supplied to -D option, the line data with no matching record in the database file will be eliminated.

chr$(1)+"ABCDE" ----> no matching data -----> ""

***Option to Machine Code Module**:*

A machine code module must be loaded by LOADBIN.. Thus, any options to the submodule must be supplied through the loader LOADBIN..

To tell LOADBIN.to give a machine code module an option string, use -O option.

Example:

>loadbin -Fsetstr.bin -O"-S0123456789 -N"

* LOADBIN will give the string "-S0123456789 -N" to the module SETSTR.BIN. It is always a good practice to quote the string

7.2 Stand-Alone

7.2.1 Roundup.exe

Description:

As its name suggests, the submodule rounds up a number(actually, a numeric string - a string consisting only of the numeric characters, a period, spaces and +/- characters) at a specific digit place.

input	ROUNDUP	output
"264.235"	-------->	-D-2
-------> "264.24"		
		-D0
-------> "264"		
		-D1
-------> "260"		
		-D2
-------> "300"		

The digit place to round up a number at is a parameter to be set from the command line.

This submodule ignores all the non-numeric line data. When it receives the line data, it checks if the data is a numeric string or not. Only if it is, the submodule will operate on it or else it will be passed on to the next module.

Note:

This submodule operates only on the line data. Be sure that RsKey passes the line data to the submodule.

The latest version of this submodule will refuse to load if RsKey's line option is OFF.

Install:

> roundup -Dxx

To run the module, -D option is a mandatory option. It sets the digit place to round up the incoming numeric data.

Release:

> roundup -R

Options:

-D option (Mandatory)

This option sets at which digit place the incoming numeric strings are to be rounded up.

> roundup -Dx

... xxxxxx.xxxxxx ...
654321 012345
| |
-d5 |
-d-2

-S option

-S(Space Remove) option will eliminate any leading space characters in the incoming numeric line data.

```
"         1.2345"    ---> "1.2345"
 ~~~
spaces
```

-P option

-P(Plus)option will remove '+' character at the beginning of the numeric line data, if any.

```
>roundup -D-3 -P

"+        0.1234" --->    "         0.123"
 ~
```

* "+" character is removed from the data

```
> roundup -D-3 -S -P

"+        0.1234" --->    "0.123"
 ~~~~
```

* "+" and the spaces are removed.

-M option

-M(Minus)option will remove "-" character at the beginning of the numeric line data, if any.

```
>roundup -D-3 -M
```

"- 0.1234" ---> " 0.123"

~

* "-" character is removed.

-B option

The submodule will give off beep sound each time the line of data is processed. This option can accompany a numeric value for the beep duration. If no value is supplied, the beep duration will be set to 1 second.

> roundup -D0 -B

This will set the beep duration to 1 second(default).

> round up -D0 -B3

The beep will sound for 3 seconds.

7.2.2 Xmodem.exe

Description:

An installation of this submodule will enable RsKey to receive the data from an external device with Xmodem protocol(128 Check Sum).

Install:

> xmodem

This module accepts no options.

Release:

> xmodem -R

Note:

In XMODEM, some of the communication parameters are fixed:

data bit	:	8 bits
stop bit	:	1 bit
parity	:	none parity

When executed, this submodule will automatically reset RsKey's parameters to these fixed values.

When removed, it will restore the original parameter setting of RsKey.

Do not install this module unless the sender supports the protocol.

7.2.3 Pollsub.exe

Description:

Given a string, this submodule keeps transmitting the string, character by character, each character appended suffix characters, through the port on which RsKey is active.

Given String	"ABCD"
Transmitted	'A'+suffix

'B'+suffix
'C'+suffix
'D'+suffix

The target application of this submodule is the polling of devices in a network configuration.

In many network configuration where multiple devices are connected with one another, each device is assigned unique ID and the network's host computer uses the ID to collect the data from it. The transmitted characters by this submodule are usually the ID characters assigned to each device. This communication method is usually referred to as POLLING.

When this submodule is installed, it will keep broadcasting ID characters in the background.

Install:

> pollsub -C[poll character]

-C option is mandatory. For the option's detail, see below.

Right after the submodule is run, the polling function is disabled. To enable it, press CTRL + RIGHT SHIFT. To disable it, press the same key combination.

Release:

> pollsub -R

Options:

-C Option [Mandatory]

Supply this option with poll characters to be transmitted sequentially.

> pollsub -cABC

* The module keeps sending out

'A'+CR/LF
'B'+CR/LF
'C'+CR/LF

Since no suffix is specified, the default suffix, CR/LF, is appended to the end of each character.

The submodule dose not have any default poll characters. This option is always required when running this submodule. If it is not used, an error will be reported.

-S Option

-S option sets the characters to be appended to the end of the polling characters.

> pollsub -cABC -s¥0d

* In transmitting a poll character, the submodule suffixes CR.

'A'+CR
'B'+CR
'C'+CR

More than one character may be supplied to -S option.

> pollsub -cABC -s0¥09¥0d¥0a

A+0+TAB+CR/LF
B+0+TAB+CR/LF
C+0+TAB+CR/LF

-I Option

-I option controls the polling speed.

> pollsub -cABC -i2

* To slow the polling speed, give a bigger number than 1. The bigger the number, the slower the speed.

Note:

1. The character transmission is based on RsKey's communication parameter.

2. The polling function is turned off right after it is executed. Be sure to turn it on with CTRL + RIGHT SHIFT. The press of the same key combination will turn on and off the polling function.

7.2.4 Biossub.exe

Warning:

1. BIOSSUB is intended for use only by those who are interested in developing programs using RS232c ports. To make full use of the submodule, the knowledge on BIOS and interrupts is required. The casual users should skip the material in this section.

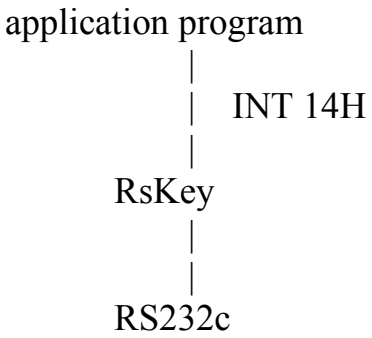
2. BIOSSUB is a preliminary version. No official support on this submodule is available.

3. BIOSSUB and its related files, when used with RsKey, comprise one complete sub-system which stands on its own. In future, a finished version may be released as a separate product. The purpose of supplying them now as the standard submodule is to hint at the potential use of RsKey.

Description :

As the submodule name suggests, BIOSSUB replaces PC's RS232c BIOS(INT 14h) for the port RsKey is active on. It has no effect on the other RS232c port for which ROM BIOS remains effective.

When this submodule is installed, it overrides the existing ROM BIOS routine for the port RsKey is running on and INT 14H becomes the interface with RsKey for foreground programs.



This submodule demonstrates that RsKey can be configured to be a general purpose RS232c sub-routine.

While this module is resident, RsKey's keyboard emulation function is disabled by default.

Install :

To execute, type at MS-DOS prompt, simply type,

> biossub

Release:

To remove the module from the memory,

> biossub -r

While this module is resident, the call to 14H for the port RsKey is running should use BIOS interface library provided along with this submodule. The BIOS library is found in BIOSLIB directory under RSKEY directory.

For the details, refer to BIOSLIB.C and its header file. Sample programs are also provided.

Although the functions in BIOSSUB works as they are, they should be re-written in -line assembler or assembler(in-line assembler codes are also included in the latest version).

Note:

1. Data Transmission

Data transmission functions in the interface module merely set a character or a string in the send buffer within BIOSSUB. The program control will come back immediately from these functions; it does not wait till transmissions finish. BIOSSUB will transmit data in the background.

The size of the send buffer is 512 bytes. If the buffer is full, the call to transmission functions will result in an error with the error code being -1. Other than this buffer full error, there is no other cause of send error.

2. AH Value

Any call to 14H must be called with AH equal to 1 or 2. AH will not take any other value.

3. Finding out the port RsKey is active on.

Call 14H with DX set to 0xff.

4. Call Back Function

BIOSLIB has a function "rs_set_callback_func()". This function sets up a function to be called by BIOSSUB when the send buffer within BIOSSUB becomes empty. SAMPLE.C is a sample program to use this function.

5. Initialization Function

A program that uses BIOSLIB must call "rs_rskey_bios_init()" before calling any other BIOSLIB functions. The codes in BIOSLIB.C shows why this must be so.

6. Turning on RsKey's Keyboard Emulation Function

Right after BIOSSUB is executed, RsKey's keyboard emulation function is disabled. To enable it, call "rs_enable_key_emulator(1)". To disable, use 0 for the argument to the function.

7. Controlling POLLSUB

When POLLSUB is in use, polling function of the submodule can be controlled from within BIOSSUB. For this, BIOSSUB must be executed with the option -P.

```
> biossub -p
```

This option permits BIOSSUB to communicate with POLLLSUB through RsKey. "rs_enable_polling(int sw)" is supported by this feature.

To illustrate the use of this function, suppose that POLLSUB is resident and it is sending out poll characters continuously. While it is taking place, you want to send out a string through the port RsKey is running on(meaning POLLSUB is active on). For this, the polling should be stopped temporarily, the string is sent out and the polling must be re-started. This can be done manually. BIOSSUB's support to control polling will enable the program to manage it without operator's intervention.

7.2.5 Filesub.exe

Description:

When this submodule receives line data, it accesses a data file(disk file) to search a record and replace a part of each line data with string read from the file.

Before accessing a specified data file, the submodule checks to see if a line data has the pre-defined format. Only those line data which confirm to the prescribed format will be processed. The other line data and character-based data are immediately passed on to the next submodule

This submodule is provided for a demonstration purpose to show that a submodule can read a disk file while pre-processing data.

Database File on a Disk:

To run this submodule, prepare a data file on which this module will perform searches. The file must have the format of each record being 10 bytes long. The submodule will access the file, using a record number, assuming that each record is 10 bytes long.

data0 data1 data2 .....

The file can contain up to 100 records with the first record number being 0 and the last 99.

The default database file name for this module is "DATA.DBS". Refer to the file by that name on the diskette.

At the time the module is executed, it looks for "DATA.DBS" in the current directory. If it is found, the submodule constructs the absolute path name of the file. If not found, an error will be reported. The absolute path is required to ensure that, even when the current directory is changed, the submodule still searches the same file.

A different database file can be used. For that, use -F option.

```
>filesub -fc:¥rskey¥rskey.dbs
```

Line Data Format:

For a line data to be processed by this submodule, it must have a specific format.

a. a line data must contain a record number.

b. a record number must be prefixed by '+' character

c. the range of the record number is from 0 to 99(2 digits)

d. the record number must be placed at the beginning of line data.

From b, c and d, the first digit must be "+" and the next 2 digits must be numeric characters.

Example:

```
"+01ABCDEFG"
```

"+99TestString"

Line Data Processing:

When the submodule receives a line data, it will take the following steps.

Assume that "+01ABCDEFG" has been received.

1. take out the first 3 digits("+01")

2. check if the first digit is "+".

3. if 2. is true, check if the second and the third digits are numeric

4. if 3. is true, use the number obtained in 3 as the record number to access the file. In this example, "+01" is converted to 1.

5. access the file and read the record. In this example, it reads in the 1st record and obtains "data1".

6. after obtaining a record data, replace the first 3 digits with the record data.

"+01ABCDEFG" ----> "data1ABCDEFG"

6. pass the processed data on to the next submodule or to RsKey

If the record number is bigger than the last record number, the module will not search the file. Instead, the first 3 digits are replaced by "??". If a file contains 30 records and the submodule receives "+31ABCDEFG", it will be converted to

"??ABCDEFG"

Options:

-F Option

This option is used to specify a database file name other than the default "DATA.DBS" file.

>filesub -f¥newfile

* the file "newfile" in the root directory will be the database file.

Be sure to use the absolute path name of a file.

Avoiding giving a file name like

>filesub -fdatabase.dbs

The submodule can be loaded without an error. But when the current directory is changed(this often happens) and the new directory contains no file under this name, the submodule will be out of service.

-B option

This option will enable the resizing of the internal buffer of this module. The default buffer size is 1K.

> filesub -b512

* Set the buffer size to 512 bytes.

Do not use this option unless a technical release of this submodule is at hand.

-S option

-S option will set the size of stack this module uses. Do not use this option unless you understand the consequence.

Notes:

In loading FILESUB., make sure to enable RsKey's line option. This module can process line data only(the latest version will refuse to run unless RsKey's line option is ON).

It is recommended that this submodule be the first one to be loaded so that it is at the bottom of the module chain.

This submodule uses MS-DOS functions to access a disk file. If the foreground program is using MS-DOS function, this submodule will not be able to access the specified database file.

For instance, if you issue the command like,

> copy con nul

this submodule fails to function. It will only wait till it is able to use MS-DOS function which never takes place.

A few limitations though the submodule has, it works with many programs very usefully.

8. Writing a Sub-Module

8.1 General

Before reading this document, you need to be familiar with RsKey's submodules. This section describes how to develop your own submodules for RsKey.

Rskey comes with a source code file USERSUB.C and 2 objects files PROTO.OBJ and CCHAIN.OBJ. From these files, a user defined submodule may be built, quite easily.

This tool is intended for non-professional RsKey users who wish to extend RsKey's capability to suit their requirements.

8.2 Necessary Software

Borland International's TurboC or BorlandC is required to build executable
programs from the provides files.

8.3 Generic User Submodule

To build a generic user submodule,

1. Compile USERSUB.C in small model. Small model is the default of TurboC / Borland C. Be sure to compile in C mode, not in C++ .

2. Link USERSUB.OBJ and PROTO.OBJ (In case TurboC is used, CCHAIN.OBJ must be also linked together).

These can be done in a single command line as follows;

```
>TCC usersub.c proto.obj cchain.obj

>BCC usersub.c proto.obj
```

The created USERSUB.is a TSR that can dynamically link with RsKey. To run it, just enter

```
>usersub
```

Then run to find out if "usersub" is listed or not.

To release the submodule, use -R option

```
>usersub -r
```

The submodule does nothing, except that it links with RsKey and stays resident.

It may be used as the base code for writing your submodule by filling out the empty functions.

Note:

1. These files are mostly for filter submodules. A complex submodule may not be built from them.

2. Created TSR will consume more memory than the submodules that come with RsKey. RsKey's standard submodules as well as RsKey are based on different program technique to reduce the resident memory requirement to the minimum.

The advantages of using the provided files are that many standard C non-I/O related library functions can be used in the resident portion and that a submodule may be written as if a standard C program is written. The submodule development is greatly simplified and a filter submodule can be written in a flash.

Most user required submodules comes within the range covered by the provided files. RsKey users with C knowledge do not have to rely on an external supplier to extend RsKey's functions.

There are areas that can not be covered by this tool. In that case, contact us or your supplier.

8.4 Writing User

In USERSUB.C, you find the following functions; some of them are already defined and the other empty.

```
void    set_user_title( void );
void    get_argument( int , char ** );
void    disp_help(int status);
void    init_data( void );
void    set_user_vector( void );
void    restore_user_vector( void );
void    process_line( void );
void    process_char( void );
```

Writing a user submodule is to fill out these functions.

They are called from the main program in PROTO.OBJ and they provide necessary information to it. If each of these functions responds to the main program correctly, the created submodule would work properly.

8.4.1 void set_user_title(void);

This function sets 3 variables -- Unique_Prog_ID, user_title and module_option. Take a look at USERSUB.C how to do them.

As the name suggests, Unique_Prog_ID must be UNIQUE program name.

user_title is a string that is displayed when the submodule is executed. Place the name of the submodule and the copyright info.

Where multiple submodules are used, line or character data is passed around the submodules. In some cases, after this submodule, you want to prevent data from being passed on to the one below it. If this is the case, set the variable to 0. By default, it is set to 1.

8.4.2 void get_argument(int , char **);

The function analyzes the command line argument when the submodule is run. For the program release option, use -R to be consistent with all the other programs in RsKey package. Providing -H option is also recommended.

8.4.3 void disp_help(int status);

USERSUB.C is so implemented that, if any option error occurs or -H option is specified, this function is called. This function exits the program. No cleanup is required before exiting.

8.4.4 void init_data(void);

This function is called before calling "set_user_vector(void)". Global variables may be initialed here rather than where they are declared.

8.4.5 void set_user_vector(void);

If there is any vector you want to trap, do it here.

8.4.5 void restore_user_vector(void);

This function is called when -R(remove)option is used. Restore all the vectors hooked in "set_user_vector(void)".

8.4.6 void process_line(void);

When RsKey receives input from an external device and LINE option is enabled, the line data will be passed to this function. The line data is in the variable "line_data[]" and its length is in "line_data_len".

Processing the line data is to modify the content of "line_data[]". If the data length is changed, do not forget to set "line_data_len" to the new data length.

8.4.7 void process_char(void);

When RsKey receives input from an external device and LINE option is OFF, the character will be passed to this function.

The character data is in "char_data". The character data may be

1. converted to another character
2. replaced by a string
3. deleted

To convert to anther character, just set the new character to "char_data".

To replace the character with a string, set the string to "line_data[]" and the data length to "line_data_len". Also, to notify the main program that the character is replaced by a line data, you must set "char_process_status" to 1.

To delete a character, set "char_process_status" to -1. The main program will know that the character has been deleted.

8.5 Example User

3 example user submodules are supplied. They illustrate how you can fill in the functions described above.

8.5.1. EXAMPLE1.C / EXAMPLE1.

This submodule will convert cases of characters. The function void process_line(void) is called only when RsKey's line option is enabled. If the option is OFF, void process_char(void); is called. This program works regardless of the option setting, because both functions are implemented.

Notice how it is obtaining the command line options and saving the option values to a global variable. The global variable is used later in process_line() and process_char() functions to determine how the data is to
be processed.

To run the program, just type,

```
>example1
```

To remove,

```
>example1 -r
```

To display the help,

```
>example1 -h
```

This would be a typical user submodule.

8.5.2. EXAMPLE2.C / EXAMPLE2.

This submodule adds several new options to EXAMPLE1.C. Everything in EXAMPLE1.C is in this file. The new options -S(space filter), -P (plus filter), -M(minus filter) and -D(decimal to hex) apply only to the line data. Therefore, no changes are made in the process_char() function.

Notice that the program is using the standard C library's functions. It is using "atol" and "ltoa". A submodule can use a function "sprintf" to set a formatted string in a specified buffer.

[NOTE]

It is possible to use "double" or "float" data types in a submodule. But if it is done, the program size will increase significantly. Since a submodule is TSR, it is recommended that the data types be not used unless you do not care the memory consumption by your submodule. Instead, try to provide your own functions to manipulate on a string.

Using C's library functions will make the program size bigger. To reduce the size of your program, provide your own routines.

8.5 3. EXAMPLE3.C / EXAMPLE3.

This submodule shows how to convert a character or a line data into another. The data tables will reside in memory. For this reason, if the data table contains more entries, the submodule consumes more memory.

This program will convert "012345", "123456", "234567", "345678", "456789", "567890", "678901", "789012","890123" and "901234" to strings set in the conversion table.

Character data 'A', 'B' and 'C' are processed as follows;

'A' ----> deleted

'B' -----> converted to 'b'

'C' -----> converted to the string " "C" is pressed"

[NOTE]

RsKey's standard submodule SETSTR.BIN, instead of placing its database file in memory, puts the data file in EMS. This will solve the memory consumption problem, but it causes another new problem due to the use of EMS(for the details, refer to the manual on SETSTR.BIN).

For DOS TSR to read a disk file is difficult, particularly when the data is continuously coming in while reading a disk file. From the generic submodule files, you can not write a submodule that handles a disk file while it stays resident.

If you wish to handle large amount of data in your submodule, please think about using Windows and a Windows application together with wRsKey. Under Windows, wRsKey and its submodule can act as a front-end processing input program for your application.

8.6 Building Executable Programs from Example C Files

```
>TCC <example C file> proto.obj cchain.obj
>BCC <example C file> proto.obj
```

For examples, to create EXAMPLE1.

```
>TCC example1.c   proto.obj   cchain.obj
>BCC example1.c   proto.obj
```