Documentation
for
STUFFIT 3.21 / STUFFKEY 4.00

Although copyrighted, the described programs are a
'labor of love'
and completely free!

Stuffkey.doc Last updated by Juergen Geist 1992 November 16.

Stuffkey 4.00 has the following enhancements over 3.30

- {IfSucc}/{Else}/{Endif} logic using ~{Timeout} on F{Find} and P{Prompt} commands.
- A {Stop} command.
- Improved error message pointers.

Stuffkey.doc Last updated by Juergen Geist 1992 October 17.

Stuffit 3.21 was written by Terje Mathison.

Stuffkey 3.30, by Juergen Geist, is Terje's code with the following enhancements:

- Error messages including error location pointer.
- Improved background time handling in Windows, Desqview, etc.
- Timer resolution down to timer ticks, 1/18 sec.
- A time out feature for Prompt and Find commands; the first step to adding If/Then/Else login.
- A simplified Prompt command, i.e.: p"text".

It is hoped that Terje will incorporate these mods; to avoid confusion until/if he does, Stuffkey will exist. All Stuffkey changes are documented here and in the .asm file. The 'zip' file should contain both versions of the .asm and .com files.

StuffIt.Doc Last-1 update by Roedy Green 1992 June 21

Table of Contents

PURPOSE

Stuffit 3.21 is a utility that automates the keyboard. You can provide a script of keystrokes and Stuffit will automatically type them into your application programs. What makes Stuffit different from other similar keyboard macro programs is:

1. The keyboard remains fully functional the entire time for additional manual input or correction.

2. Stuffit works with intractable programs that throw away keystrokes, bypass DOS etc.

3. Stuffit watches the screen. It does not just type blindly.

4 Stuffkey 4.0 supports if/else/endif logic based on what it finds on the screen.

5. Stuffit can handle large scripts, up to 64K.

6. Stuffit lets you generate more possible keystroke chords than other similar products.

7. Assembler source code is provided, giving you the option of adding new features to the program.

COMMAND LINE PARAMETERS IN A NUTSHELL

+0 or +   Delay until the application has used up all the keystrokes in the type-ahead buffer. This is useful for applications that throw away keystrokes fed to it too quickly. This is also a handy way to flush any pending (uneaten) codes from a previous invocation of StuffIt. e.g.: StuffIt +0 empties StuffIt's internal buffer. Note + all by itself defaults to +0.

+hh:mm:ss   time delay. (hours and minutes optional) +4 means wait 4 seconds (must be 0..59). +2:4 means wait 2 minutes and 4 seconds. +5:: means wait 5 hours. Represented internally as {Wait}. Often you want a +5 or so at the start of your script to give the program you are stuffing 5 seconds to load and get ready for a keystroke. The timing starts relative to when STUFFIT runs, not from when the following application requests its first keystroke.

+.tt   time delay as above, but in timer ticks, or 1/18 second resolution. tt can range from 1 to 18. Note this can't be mixed with hh:mm:ss, etc. Useful when you need just a bit of delay between keystrokes. +.6 will delay 1/3 second. (**-Stuffkey-**)

=hh:mm:ss   delay until a specific time of day (24 hour format) (hours and minutes optional). =16:00:00 means wait till 4 PM. =00:00:00 means wait till midnight. =0 also means wait till midnight. Represented internally as {AtTime}. Be careful! =04:00 would mean 4 minutes after midnight!

~hh:mm:ss   {TimeOut} next function, i.e., {Prompt}, {Find} in supplied time. Timer tick format is ok. i.e. ~.15. This allows script to continue on a "not found" condition. Stuffkey V.4.00 adds the ability to test the results of P {Prompt} and F {Find} commands. See {IfSucc}, {Else} and {Endif} for details. (**-Stuffkey-**).

nn   stuff a given character code e.g. 27 for Escape, 13 for Enter Only ASCII decimal numerics are accepted. See ASCII code chart in your IBM AT tech Reference. You may also use the mnemonics e.g. {Esc} for 27. Note that you may use either [] or {} but you must use [{], [}], {[} and {]} for the braces and brackets themselves.

@nn stuff an extended code preceded by a null. Only numerics are accepted. You may use the numeric forms or the mnemonics e.g. @68 or {F10}, @73 or {PgUp} See table following for the common codes. If you cannot find an mnemonic, you will have to use the numeric form.

cc:ss specify both the decimal character and its scan code e.g. 43:74 to mean the Plus key on the numeric keypad, rather than the ordinary plus key. See table following for common pairs. You may use the numeric forms or the mnemonics e.g. 10:28 or {^Cr} for Control-Enter. If you cannot find a mnemonic, you will have to use the numeric pair form.

'xxxx' Stuff the characters between the quote marks. The string itself may not contain 's.

"xxxx" Stuff the characters between the double quote marks. The string itself may not contain "s.

F70,20,5,01,"Please enter your phone number"
The absolute FIND command. Wait patiently for text to appear at col, row, length, attribute. The attribute is optional. StuffIt searches starting at column 70, row 20 for the string "Please enter your phone number". It waits until the string appears. The 5 means allow some slop. The string may START anywhere in a window within 5 characters to the right of the specified starting point. It does not mean the whole string must FIT in a window five characters wide. The slop may wrap around from one line to the next. See the notes below on Stuffit's co-ordinate system, where 1,1 is the upper left corner. The 01 is a rarely used feature. It means, the match only counts if the string is displayed with decimal video attribute "01". If you leave out the 01, leave out its comma as well. Represented internally as {Find}.
(**-Stuffkey-**) Provides the ~ or {TimeOut} feature to enable escaping from an "unfound" condition.

P10,1,5,01,"Please enter your phone number"

The relative PROMPT command. Wait patiently for
text to appear at relative col, row, length,
attribute. The attribute is optional. Stuffit
searches for the string "Please enter your phone
number" starting left ten columns, and up one row
from where the cursor is. It waits until the
string appears. The 5 means allow some slop. The
string may START anywhere in a window within 5
characters to the right of the specified starting
point. It does not mean the whole string must FIT
in a window five characters wide. The slop may
wrap around from one line to the next. The 01 is
a rarely used feature. It means, the match only
counts if the string is displayed with decimal
video attribute "01". If you leave out the 01,
leave out its comma as well. Represented
internally as {Prompt}.

P"Password: "

(**--Stuffkey--**)
The 'easy to use' version of the PROMPT command. This works the same as above, but defaults to looking for text at 3 spaces plus the length of the supplied text to the left of the cursor on the current line. The default 'slop' window is 4, with no attributes. In the example above, the text "Password: " is 10 characters long, so this is equivalent to P13,0,4,"Password: " in the detailed Prompt command. If the cursor is on line 12, position 30, Stuffkey would search for the text on line 12 starting at positions 27 thru 30. These defaults usually work, and certainly make the Prompt command easier to use! Stuffkey provides the ~ or {TimeOut} feature to enable escaping from an "unfound" condition for both versions of {Prompt}.

! Reboot. (=0 ! will reboot at midnight.) Also
known as {^aDel} or {Boot}. The code used to
reboot is bare bones. It does not flush caches,
watch out for DESQview or Windows etc. If you
need a safer reboot, use the separate REBOOT.Com
utility posted on BIX in IBM.UTILS/LISTINGS.

(**--Stuffkey--**) changes this from a warm to cold boot; reset vs. Ctl-Alt-Del.

{IfTrue}
{IfT}
{IfSucc} (**--Stuffkey--** 4.00)

If the ~ {TimeOut} function is used before a F {Find} or P

{Prompt} command, said command can be tested for success or failure. If the string indicated is found, {IfSucc} will be true, and commands immediately following it will be executed. If the string is not found, execution will resume after the next {Else} or {Endif}, whichever is found first. No attempt is made to match up If's with else's and endif's, they are processed as they are found. Unexpected {Else} and {Endif} commands are simply ignored. Thus, one {Else} or {Endif} command will respond to two or more prior {IfSucc} commands.

{IfFalse}
{IfF}
{IfFail}   (**--Stuffkey--** 4.00)
                            The inverse of {IfSucc} above. Equivalent to If NOT Success.
                            All rules for {IfSucc} apply.

{Else}   (**--Stuffkey--** 4.00)
                            Stuffkey will resume execution at this point on a failed
                            {IfSucc}, {IfTrue}, {IfT}, {IfFail}, {IfFalse} or {IfF} command.
                            If the {Else} command is run into unexpectedly, it will simply
                            be ignored.

{Endif}   (**--Stuffkey--** 4.00)
                            Stuffkey searches for a and will resume execution here after
                            any If statement, unless it finds an Else statement first.
                            Unexpected {Endif}'s are simply ignored.

{Stop}   (**--Stuffkey--** 4.00)
                            This command allows you to halt a script. Useful for 'one way'
                            If commands. i.e., If 'something' Do 'something' else stop the
                            program.

/F:MyFile.Txt FILE. Read commands from MyFile.Txt instead of
            from the command line. This command may be only
            be used on the command line, not nested in a file.

/B:512   BUFFER. Allocate room for 512 bytes for the
            script. You must allocate enough to store the
            ENTIRE script once it has been converted to a
            compact internal form. On initial loading,
            StuffIt will use a minimum of 512 bytes, or
            automatically expand to whatever is necessary to
            needed to store the compact tokenized form of the
            input file. If you want to save a few bytes you
            can prune it back using the /B: command. Future
            versions may do this automatically.

/E   EXPANDED. Makes Stuffit compatible with keyboard
            look-ahead buffer expanders. Unfortunately this
            option is incompatible with very old BIOSes that
            do not support the BIOS table header entries at
            40:80 and 40:82.

/L   LIST. Displays a list of all the mnemonic codes
            for keystrokes supported -- in other words a

miniature version of this manual. Since that list
is generated directly from the parsing tables
inside the program, when there are discrepancies,
use that list in preference to this manual.

/R REMOVE. Will remove (Unload) Stuffit from RAM.
StuffIt is a TSR and so stays resident in RAM.
The overhead is minuscule, a mere 1232 bytes, half
of which is the buffer. You cannot combine /R
with a script on the same line. STUFFIT /R will
automatically terminate any script running.

/Sxx SIGNATURE. If want to have multiple copies of
Stuffit running at once, you need to give each one
its own two-letter signature. /SST is the
default. Why would you want more than one copy
running? Stuffit can only look for one string on
the screen at a time. If you needed to look for
two AT ONCE, you would need two copies of Stuffit
running.

CO-ORDINATE SYSTEM

The co-ordinate system Stuffit uses for the FIND command is very familiar to programmers who poke bytes into the REGEN buffer of display adapters, but it may seem a little strange to someone who has a mathematical background. Mathematicians have two systems:

1. X,Y co-ordinates where the origin 0,0 is in the lower left corner.

2. Matrices indexed by row and column. Usually rows and columns are numbered starting with 1. Rows are numbered starting at the top. Traditionally co-ordinates are given in the order row, then column.

Stuffit uses a hybrid of these two schemes. It uses row and column, and starts numbering at 1,1 in the upper left corner. Stuffit uses column, row order rather than the usual row, column. The co-ordinate system for the PROMPT command is strange until you consider that the text you want to look for will usually be to the left and above the cursor. For relative positioning in the PROMPT command, left and up are considered positive with the current cursor position considered as 0,0. This is the exact opposite convention to what the FIND command uses. Beware! (Stuffkey's EASY prompt command might help.)

EXAMPLES:

Format floppies alternately in A: and B:, without answering any prompts:

```
REM EXAMPLE 1 FLOPPY FORMATTER (Batch file calling Stuffit)
echo off
:loop
Stuffit ' ' +0 13 P25,0,5,"Format another" +1 'N' 13
format A:
Stuffit ' ' +0 13 P25,0,5,"Format another" +1 'N' 13
format B:
goto loop


REM EXAMPLE 2 AUTOEXEC.BAT
Rem Start backup program in batch mode:
Rem Use Stuffit, to reboot GW program at midnight
Stuffit =00:00:00 !
Rem Start GW program
LAN-GW


REM Example 3, running Stacker's Defragger unattended
C:\SYS\Stuffit.Com +5 {C} F21,12,1,"Optimization complete" +2
{F10} {Y}
C:\Stacker\SDEFRAG.Exe F: /SE


REM   Example 4, logging into a NOVELL LAN with Stuffkey 4.0.
rem   Assuming use of Novell's IPX and NETX, network drive is F:,
rem login name is poltergeist, and password is pword.
rem   Load stuffkey to provide password if the Password:
rem prompt is found within 30 seconds of loading ipx, else reboot.
rem   This loading of stuffkey before running ipx, netx, etc.,
rem allows recovering/rebooting from a 'hung machine' condition
rem if any of ipx, netx, or log commands fail; this is a real live
rem example, used daily - it works!

stuffkey + ~30 P"password: "{IfT}+ 'pword' 13 {Else} + +1 ! {Endif}
ipx
netx
f:
log poltergeist
```

NOTE: Extra examples are included in the archive as xxxxx.bat and/or xxxxx.key files. To use the .bat files, just run them. To use the .key files, type "stuffkey /f:xxxxx.key", without the quotes, where xxxxxx is the name of the file. One such is stufftim.key, which tests the time commands across midnight. Also included is stufferr.bat, which runs stuffkey with deliberate errors to produce the following error examples.

ERROR EXAMPLES

 The error handler in Stuffkey 4.0 first displays the text of the errored line.
 Second, it attempts to point to the beginning of the command in error, followed by dashes to a second pointer under the actual offending character or number. These pointers, while not perfect, are pretty close; certainly better than not having them!
 Third, it displays the errored line number.
 Fourth, it gives a textual description of the error.

Stuffkey error examples, generated by Stufferr.bat, follow.

EXAMPLE 1: THE COMMA FOLLOWING THE PROMPT PROBABLY SHOULDN'T BE THERE. THIS MAKES THE FIRST ARG, DX (COLUMN) 0, AND SHIFTS THE REMAINING ARGS LEFT. THUS, THE 0 CHANGES FROM A LEGAL DY (ROW) ARG TO AN ILLEGAL SLOP ARGUMENT. NOTE THE ARROW POINTS TO THE ARG AFTER THE ERROR; A QUIRK TO BE AWARE OF.

```
+0 +1 {Prompt},12,0,4,"Password: " 'Poltergeist' 13
            ^----^
** Error lin# 1 Invalid Prompt find cmd.  Bad number - out of range.
```

EXAMPLE 2: THE 128 DX (COLUMN) ARG. IS ILLEGAL. VALUES ARE +/- 127.

```
+0 +1 {Prompt}128,0,4,"Password: " 'Poltergeist' 13
            ^--^
** Error lin# 1 Invalid Prompt find cmd.  Bad number - out of range.
```

EXAMPLE 3: A SPACE IS A LEGAL TOKEN, LEADING SPACES ARE NOT.

```
+0 { } {F12} { F12}
            ^---^
** Error lin# 1 Invalid Token.
```

EXAMPLE 4: THE TRAILING SPACE CAUSES THIS TOKEN TO BE UNRECOGNIZED. NOTE THE DIFFERENCES IN THE ERROR LOCATION POINTER BETWEEN INVALID TOKENS AND UNRECOGNIZED TOKENS/COMMANDS.

```
+0 { } {F12} {F12 }
        ^
** Error lin# 1 Invalid or unknown command.
```

EXAMPLE 5: TIME FORMAT CAN BE HH:MM:SS OR .TT, BUT NOT BOTH

```
+ +.1 +1 +1.1 F0,1,1920,"Cshow version" + +1 13"*.gif"13
      ^-^
** Error lin# 1 Invalid Time.  Bad number.
```

EXAMPLE 6: THE FIND COMMAND COORDINATES START AT 1,1 NOT 0,0.

```
+ +.1 +1  F0,1,1920,"Cshow version" + +1 13"*.gif"13
       ^
** Error lin# 1 Invalid Find string cmd.  Bad number - out of range.
```

EXAMPLE 7: THE QUOTES AFTER THE 13 MAKE IT LOOK LIKE A BAD NUMBER. IT IS A GOOD IDEA TO LEAVE A SPACE BETWEEN ITEMS.

```
+ +.1 +1  F1,1,1920,"Cshow version" + +1 13"*.gif"13
                         ^-^
** Error lin# 1 Invalid or unknown command.  Bad number.
```

EXAMPLE 8: NOTE THE LOCATION OF THE ERROR POINTERS.

```
+ {TimeOut}.1 {TokenTooLong}
       ^------^
** Error lin# 1 Invalid Token.
```

TROUBLE SHOOTING

PROBLEM: Stuffit fails to send all the keystrokes. Only some keystrokes get through.

SOLUTION: Your program is clearing the type-ahead buffer from time to time. You can outfox it by using, say, +2 or +.6 commands to insert a short delay AFTER the program has cleared the type-ahead buffer before Stuffit starts poking characters into it again.

This can be combined with the F and P commands to hold off generating the keystrokes until an omen appears on the screen that the program is ready for input and is not about to throw away the keystrokes you feed it. Sometimes you must combine the techniques, an F or P command followed by a short delay.

PROBLEM: I finally got Stuffit to work by putting in a huge great time delay at the start to handle the worst case, but most of the time such a long time delay is not needed, and just wastes time.

SOLUTION: Stuffit starts sending keystrokes right away. It does not wait until the succeeding program is loaded. You can stuff a dummy character followed by a +0 time delay, e.g. Stuffit 'z' +0. Stuffit will stuff the z, then process the +0 which causes it to wait for the keystroke buffer to empty. The application program then starts up. The application goes about its initialization then finally it throws the 'z' away by clearing the keystroke buffer. Stuffit notices the buffer is now empty and continues with the script. The net effect is Stuffit waited for the precise moment the program was ready before stuffing the keystrokes for the app.

Another solution is to use the FIND feature to look for the initial prompt.

PROBLEM: I tried the find command but it just won't work.

SOLUTION: First read up again on Stuffit's two co-ordinate systems. You may not be numbering your rows and columns the way Stuffit does. Next, note the slop only works to the right. If your guess at the co-ordinate were off by one to the left, you would never get a hit, no matter how much slop you allowed. Note that slop only works on columns, not rows. You generally must

get the row bang on. If you had 160 COLUMNS of slop however,
this would be treated as two ROWS of slop.

PROBLEM: Stuffit just echoes its command syntax summary no
matter what I do.

SOLUTION: Stuffit is laconic in its error messages, but it is
still better than it used to be. It does not tell you what you
did wrong. You have to guess. In a worst case, the way you do
this is to take out all your commands. Then add them one by one.
The one you last added at the time of failure is the one with the
syntax error.

Note that Stuffit completely analyzes the script before starting
to send any keystrokes to your application. If there is even one
tiny error in it, it will not do anything.

ALTERNATE SOLUTION: Use (**--Stuffkey--**). It has reasonably robust error
handling. It will still abort on the first error it finds, but it will describe the nature
of the error, display the offending line, and point to the error. See the ERROR
EXAMPLES section for more detail.

PROBLEM: My program insists I use keypad - minus. Where do I
find the magic colon pair for such keystrokes?

SOLUTION: There is a list of the most commonly used pairs at the
end of this document. See the IBM AT Technical reference manual
for more information on system scan codes.

PROBLEM: You referred me to the IBM AT Technical Reference
Manual for scan codes and ASCII codes. I don't have this
expensive book. What can I do?

SOLUTION: Call Falk Data Systems at (915) 684-7670, 5322 Rockwood
Court, El Paso Texas, 79932. They make an inexpensive wall chart
that shows the ASCII codes.

PROBLEM: I want to generate Shift-PrtSc. There does not appear
to be a scan code for this combination.

SOLUTION: Use the {PrintScreen} mnemonic. It will simulate the
special handling of PrtSc or Shift-PrtSc.

PROBLEM: the F command does not work to fetch a script out of a
file.

SOLUTION: The command is /F:, not F, for fetching from a file.
This is confusing. The F command is for FINDING a magic string

on the screen. Perhaps you forgot the colon or inserted spaces.

PROBLEM: My program does not seem to recognize keystrokes.

SOLUTION: It is unlikely, but one possible cause is this: when you DON'T use the colon form to give Stuffit both the ASCII and scan code, then Stuffit fakes it by using 2 as the scan code. Most programs do not check the scan code, but a few might be confused. Stuffit automatically generates the proper scan code for most keys. If your application needs a specific scan code/char combination, you night need the explicit CC:SS form.

PROBLEM: The scan codes I need are not in the list below. Also I suspect some of the entries are incorrect, or might not be correct for my Ichiban Samurai (Steve Job's mythical Japanese computer company) computer.

SOLUTION: Terje Mathisen has written a companion utility called KEYCODE2.ZIP available on BIX in the IBM.UTILS/LISTINGS section that will display the scan codes and ASCII codes of any keystrokes as you depress them.

PROBLEM: Stuffit hiccoughs every time I use any command with a ^ in it such as {^Z} and says there is a syntax error. I have looked at the line until I am blue in the face. It is perfect!

SOLUTION: I bet you are using 4DOS. 4DOS reserves the ^ character to allow you to glue two commands together on one line. There are four ways out:
1.) stop using 4DOS.
2.) use COMMAND.COM temporarily by typing COMMAND.
3.) use the sequence Ctrl-X ^ in place of ^ to warn 4DOS this is a literal ^, not a line glue character.
4.) look for an alias -- a different way of specifying the same character, e.g. {Sub} for {^Z}, or 0:116 for {^Right}

PROBLEM: I feel like a mushroom, kept in the dark. I want to understand better what is going on. I just don't get this scan code stuff.

SOLUTION: Roedy Green has written a companion essay called "How an AT Keyboard Works" that will explain how BIOS processes keystrokes.

PROBLEM: Stuffit seems too complicated for what I need. My applications don't even throw away keystrokes. I want something simpler -- not a TSR.

SOLUTION: Try Roedy Green's SAY! posted on BIX in IBM.UTILS/LISTINGS. It is a simple utility that generates keystrokes you can feed to your applications using pipes -- e.g.

```
SAY "Y" 13 | DEL *.*
```

PROBLEM: I am going stark raving bonkers trying to figure out the absolute screen co-ordinates that various applications are using to paint the strings I need to get Stuffit to look for. Often the string I want is hanging out in the middle of nowhere. How are you supposed to count columns without a reference grid?

SOLUTION: Use PRNDSK available on BIX in IBM.UTILS/LISTINGS to capture the screen contents to a file as you run the application manually. Then use a text editor to examine the screens and count the rows and columns.

PROBLEM: The application program I am using sometimes gives me an error message, and sometimes it does not. If I wait for the error message, Stuffit will wait for ever. If I blithely assume there will be no error, my script will get stuck in the weeds when there is an error.

SOLUTION: Run TWO copies of Stuffit. Let one wait for the error message using the FIND command, and generate the keystrokes to handle it, and the other copy handle only the normal case. Stuffit normally assumes that when you run it, you just want to tack on more to the script of the copy already running. If you want to start up a second script to run in parallel you must do something like this:

```
Stuffit.Com /F:C:\Doc\MyScript.txt /SAA

Stuffit.Com /F:C:\Doc\Script2.txt /SBB
```

ALTERNATE SOLUTION. Use (**--Stuffkey--** 4.0).
It fixes exactly this problem. You can now test if the find command was successful. See ~ {Timeout}, {IfSucc}, {Else}, and {Endif} earlier in the documentation. Also have a look at the usage example for logging into a Novell network; it uses If/Else logic with the prompt command.

WINDOWS, DESQVIEW, MULTI-TASKING PROBLEMS

PROBLEM: I run Stuffit in a Windows 3.1 Dos box, or in Desqview, and all the time functions take longer than they should.

SOLUTION: Switch from Stuffit to Stuffkey; it fixes this, though not all, multitasking quirks. Both versions rely on the timer tick interrupt to operate. Stuffit assumed it would be interrupted on each tick, which doesn't happen in multi-tasking environments. Stuffkey double checks how many timer ticks have actually gone by, so it can 'catch up' if it has missed some. Note though, that Stuffkey will not do anything if it receives no timer tick interrupts. Make sure the 'run in background' option is on for any background windows containing Stuffkey.

PROBLEM: I run Stuffkey in a Windows 3.1 Dos window, and it never does anything!

SOLUTION: Most likely, the window you started Stuffkey in is not the active one. Stuffkey, like any other program, needs processor time. Check to make sure the Dos Window you started stuffkey in is set to run in the background. This is normally defined in a .pif file; use the Windows Pif editor to verify background processing is enabled.

PROBLEM: I run Stuffkey in a Desqview Window, switch it to run in the background, but it stuffs its keystrokes into my foreground window.

SOLUTION: As of 16 Nov. 92, there is no solution. Desqview, by default (unlike Windows), shares the keyboard buffer among its tasks. It may be possible to change this with DV's myriad keyboard settings....this is being checked.

HOW IT WORKS UNDER THE HOOD

Most typists can type faster than programs can process keystrokes. Thus the BIOS stores keystrokes in a type-ahead buffer. Stuffit works by sneakily poking keystrokes to the type-ahead buffer when BIOS is not looking. Stuffit thus has no need of taking over INT 16 to control the handing over of keystrokes to the application. The advantage of this method is, Stuffit works with programs that sneak past INT 16 to use the Enhanced 101-key keyboard features.

There are 254 different extended ASCII codes that can be stored in a byte. However there are control sequences such as F1, Home, Ctrl-F1 which don't have a 1-byte code. These require a two byte code, the first byte being a 0-byte lead in. Stuffit can easily handle poking that pair into the keystroke buffer, using the @nn feature.

There are some keys duplicated on the keyboard. For example there is a + key sharing the = key. There also a + key on the numeric keypad. Most programs do not care which + key you pressed. Some do. These programs find out by using interrupt HEX 16 function 00. The processed character is returned in AL and the semi-cooked scan code from the keyboard is returned in AH. If you the program is picky, you can get Stuffit to generate the precise character and scan code using the cc:ss command.

Because of history, with each new keyboard design attempting to maintain partial compatibility with earlier designs, the process of generating a keystroke has become more and more complex. Each keypress down generates a code, as does each key release.

The keyboard itself translates the codes from row-column on the keyboard to a raw scan code format. Then the keyboard controller in the PC (a miniature computer in its own right) translates them again. Then the BIOS translates them yet again, and matches up the ups and downs, notices chords (multiple keys pressed at once such as Ctrl Alt and Shift) and generates the keystrokes and puts them in the keystroke type-ahead buffer. Further DOS, with its KEYB utility, may interfere with BIOS to support foreign keyboards. So you can see the term "scan" code has many meanings -- depending just where in this assembly line you look at the codes.

CREDITS

- Terje W. Mathisen wrote Stuffit and the first documentation.
- Roedy Green polished the documentation.
- Juergen Geist added his 2 bits worth to Stuffit and called it Stuffkey (3.3).

It remains to be seen whether Terje absorbs these mods into Stuffit. Juergen will definitely be adding the {IfSuccess} {Else} {Endif} logic to Stuffkey if Terje doesn't wish to add this to Stuffit.

- 16 Nov 92, Stuffkey 4.0, Juergen has added If/Else logic.

PAYMENT AND LICENSING

The program Stuffit is absolutely FREE. I (Terje) don't want money; I have money. What I want is recognition from my peers. You may use it for whatever you like, but it is not warranted for any purpose.
Copyright 1991, 1992 by Terje W. Mathisen.

Terje W. Mathisen
Spanish Fork, Utah, June 1992.

The added/modified code contained in Stuffkey is also free. I (Juergen in this paragraph) applaud Terje's efforts and will extend his philosophy - even though I could use some money!
I probably would not have expended this effort had Terje not provided his program, including source code, with no strings attached. I believe Terje's work, like mine, was/is a labor of love. Bravo Terje!
Stuffkey is copyrighted the same as Stuffit; I also like recognition and would like my name mentioned. Terje?

Juergen Geist, bixname 'poltergeist'
Vancouver, B.C. November 1992

FUTURES VAPOR WARE (Some are done, actually!)

Possible futures for Stuffit include:

1. Allowing comments in the script files. This is done in Stuffit 3.21. The ; (semicolon) marks the rest of a line as a comment, provided it is not inside a string or token. This item to be dropped from documentation in next release.

2. Allowing very large e.g. greater than a megabyte sized scripts that are stored on disk or in expanded RAM.

3. Friendlier error messages. This is done in Stuffkey 3.3. Some examples are given in the troubleshooting section of this document. Error messages are improved again in Stuffkey 4.0. This item to be dropped from 'ideas' documentation in next release.

4. Automatic /B: command.

--------------- Following ideas from J.Geist (Stuffkey)

5. {IfSuccess} {Else} {Endif} logic.
   Done in Stuffkey 4.0. This item to be dropped from 'ideas' documentation in next release.

6. * {Repeat} command, i.e., *5 {Right} or maybe {Right 5}.

7. Definitions. i.e.:
   Define {SloYes} +0 +.6 {y} +0 {Cr} +0 +.6 ; Slow y, Enter

8. Macros, Subroutines, Looping, or Gotos ???

9. A way to pause and/or abort a script from the keyboard. Eyeing the Scroll Lock key for this function.

10. A status function. i.e., Stuffkey /S[nn]? to get status of loaded copies - idle, waiting for Prompt, etc.

11. DesqView script compatibility. Desqview/Windows awareness.

12. Variables, integers at first, for looping, counting, etc.

13. String variables, including ability to "get" them from screen, keyboard, or Dos environment, and later "put" them somewhere.

SCAN CODES

Here are the Stuffit codes for some keystrokes you might want to generate. They are case insensitive. You may key them in any mixture of upper and lower case, except of course the codes for the letters themselves {A} and {a}. See your IBM AT Tech manual for more:

PLAIN KEYS

{Bs}	8:14	backspace
{Cr}	13:28	Enter (Carriage Return)
{Del}	0:83	Del
{Down}	0:80	Down
{Esc}	27:1	Esc key
{End}	0:79	End
{Home}	0:71	Home
{Ins}	0:82	Ins
{Left}	0:75	Left
{PgDn}	0:81	PgDn
{PgUp}	0:73	PgUp
{PrtScrn}	255:5	PrtScrn
{Right}	0:77	Right
{Tab}	9:15	Tab
{Up}	0:72	Up

[{]	123:26	left curly brace.
[}]	125:27	right curly brace.
{ }	32:57	space
{!}	33:2	exclamation
{"}	34:40	quote
{#}	35:4	sharp
{$}	36:5	dollar
{%}	37:6	percent
{&}	38:8	ampersand
{'}	39:40	tick, apostrophe
{(}	40:10	left parentheses
{)}	41:11	right parenthesis
{*}	42:9	star
{+}	43:13	plus
{,}	44:51	comma
{-}	45:12	minus
{.}	46:52	period
{/}	47:53	forward slash
{:}	58:39	colon

{;}	59:39	semi-colon
{<}	60:51	less than
{=}	61:13	equals
{>}	62:52	greater than
{?}	63:53	question mark
{@}	64:3	at sign
{[}	91:26	left square bracket
{\}	92:43	backslash
{]}	93:27	right square bracket
{^}	94:7	circumflex
{_}	95:12	underscore
{`}	96:41	grave
{|}	124:43	vertical bar
{~}	126:7	tilde
	174	<< French open quote
	175	>> French close quote

{0}	48:11	zero
{1}	49:2	one
{2}	50:3	two
{3}	51:4	three
{4}	52:5	four
{5}	53:6	five
{6}	54:7	six
{7}	55:8	seven
{8}	56:9	eight
{9}	57:10	nine

{¥}	157:39	Yen
		Norway O slash
{å}	134:26	a ring
{æ}	145:40	lower case ae
{¢}	155:39	cent sign
		Norway o slash
{a}	97:30	lower case a
{b}	98:48	lower case b
{c}	99:46	lower case c
{d}	100:32	lower case d
{e}	101:18	lower case e
	130	lower case e acute
{f}	102:33	lower case f
{g}	103:34	lower case g
{h}	104:35	lower case h
{i}	105:23	lower case i
{j}	106:36	lower case j
{k}	107:37	lower case k

{l}	108:38	lower case l
{m}	109:50	lower case m
{n}	110:49	lower case n
{o}	111:24	lower case o
{p}	112:25	lower case p
{q}	113:16	lower case q
{r}	114:19	lower case r
{s}	115:31	lower case s
{t}	116:20	lower case t
{u}	117:22	lower case u
{v}	118:47	lower case v
{w}	119:17	lower case w
{x}	120:45	lower case x
{y}	121:21	lower case y
{z}	122:44	lower case z

{F1}	0:59	F1
{F2}	0:60	F2
{F3}	0:61	F3
{F4}	0:62	F4
{F5}	0:63	F5
{F6}	0:64	F6
{F7}	0:65	F7
{F8}	0:66	F8
{F9}	0:67	F9
{F10}	0:68	F10
{F11}	0:133	F11
{F12}	0:134	F12

NUMERIC PAD CHARACTERS

{Enter}	13:224	numpad Enter
{n*}	42:55	numpad *
{n+}	43:78	numpad +
{n-}	45:74	numpad -
{n.}	46:83	numpad .
{n/}	47:224	numpad /
{n0}	48:82	numpad 0
{n1}	49:79	numpad 1
{n2}	50:80	numpad 2
{n3}	51:81	numpad 3
{n4}	52:75	numpad 4
{n5}	53:76	numpad 5
{n6}	54:77	numpad 6
{n7}	55:74	numpad 7
{n8}	56:72	numpad 8

{n9}	57:73	numpad 9
{^Enter}	10:224	numpad Ctrl-Enter
{^n5}	0:143	numpad Ctrl-5
{^n*}	0:150	numpad Ctrl-*
{^n+}	0:144	numpad Ctrl-+
{^n-}	0:142	numpad Ctrl--
{^n/}	0:149	numpad Ctrl-/
{aEnter}	10:224	numpad Alt-Enter
{an+}	0:78	numpad Alt-+
{an-}	0:74	numpad Alt--
{an/}	0:164	numpad Alt-/

EXTENDED ARROWPAD CHARACTERS

{eDel}	224:83	extpad Del
{eDown}	224:80	extpad Down
{eEnd}	224:79	extpad End
{eHome}	224:71	extpad Home
{eIns}	224:82	extpad Ins
{eLeft}	224:75	extpad Left
{ePgDn}	224:81	extpad PgDn
{ePgUp}	224:73	extpad PgUp
{eRight}	224:77	extpad Right
{eUp}	224:72	extpad Up
{aeDel}	0:163	extpad Alt-Del
{aeDown}	0:160	extpad Alt-Down
{aeEnd}	0:159	extpad Alt-End
{aeHome}	0:151	extpad Alt-Home
{aeIns}	0:162	extpad Alt-Ins
{aeLeft}	0:155	extpad Alt-Left
{aePgDn}	0:161	extpad Alt-PgDn
{aePgUp}	0:153	extpad Alt-PgUp
{aeRight}	0:157	extpad Alt-Right
{aeUp}	0:152	extpad Alt-Up
{an*}	0:55	extpad Alt-*
{^eDel}	224:147	extpad Ctrl-Del
{^eDown}	224:145	extpad Ctrl-Down
{^eEnd}	224:117	extpad Ctrl-End
{^eHome}	224:119	extpad Ctrl-Home
{^eIns}	224:146	extpad Ctrl-Ins
{^eLeft}	224:115	extpad Ctrl-Left
{^ePgDn}	224:118	extpad Ctrl-PgDn

{^ePgUp}	224:132	extpad Ctrl-PgUp
{^eRight}	224:116	extpad Ctrl-Right
{^eUp}	224:141	extpad Ctrl-Up

SHIFT + KEY

{sTab}	0:15	Shift-Tab
		Shift-PrtSc (not avail)
{Å}	143:26	Upper case A ring
{Æ}	146:40	capital AE
{A}	65:30	capital A
{B}	66:48	capital B
{C}	67:46	capital C
{D}	68:32	capital D
{E}	69:18	capital E
{F}	70:33	capital F
{G}	71:34	capital G
{H}	72:35	capital H
{I}	73:23	capital I
{J}	74:36	capital J
{K}	75:37	capital K
{L}	76:38	capital L
{M}	77:50	capital M
{N}	78:49	capital N
{O}	79:24	capital O
{P}	80:25	capital P
{Q}	81:16	capital Q
{R}	82:19	capital R
{S}	83:31	capital S
{T}	84:20	capital T
{U}	85:22	capital U
{V}	86:47	capital V
{W}	87:17	capital W
{X}	88:45	capital X
{Y}	89:21	capital Y
{Z}	90:44	capital Z

{sF1}	0:84	Shift-F1
{sF2}	0:85	Shift-F2
{sF3}	0:86	Shift-F3
{sF4}	0:87	Shift-F4
{sF5}	0:88	Shift-F5
{sF6}	0:89	Shift-F6
{sF7}	0:90	Shift-F7
{sF8}	0:91	Shift-F8
{sF9}	0:92	Shift-F9

{sF10}	0:93	Shift-F10
{sF11}	0:135	Shift-F11
{sF12}	0:136	Shift-F12

ALT + KEY

{aBS}	0:14	Alt-Backspace
{aCR}	0:28	Alt-Enter
{aEsc}	0:1	Alt-Esc
{a'}	0:40	Alt-'
{a*}	0:55	Alt-*
{a,}	0:51	Alt-,
{a-}	0:130	Alt--
{a.}	0:52	Alt-.
{a/}	0:53	Alt-/
{a;}	0:39	Alt-;
{a=}	0:131	Alt-=
{a[}	0:26	Alt-[
{a\}	0:43	Alt-\
{a]}	0:27	Alt-]
{a`}	0:41	Alt-`
{a0}	0:129	Alt-0
{a1}	0:120	Alt-1
{a2}	0:121	Alt-2
{a3}	0:122	Alt-3
{a4}	0:123	Alt-4
{a5}	0:124	Alt-5
{a6}	0:125	Alt-6
{a7}	0:126	Alt-7
{a8}	0:127	Alt-8
{a9}	0:128	Alt-9
{aA}	0:30	Alt-A
{aB}	0:48	Alt-B
{aC}	0:46	Alt-C
{aD}	0:32	Alt-D
{aE}	0:18	Alt-E
{aF}	0:33	Alt-F
{aG}	0:34	Alt-G
{aH}	0:35	Alt-H
{aI}	0:23	Alt-I
{aJ}	0:36	Alt-J
{aK}	0:37	Alt-K
{aL}	0:38	Alt-L

{aM}	0:50	Alt-M
{aN}	0:49	Alt-N
{aO}	0:24	Alt-O
{aP}	0:25	Alt-P
{aQ}	0:16	Alt-Q
{aR}	0:19	Alt-R
{aS}	0:31	Alt-S
{aT}	0:20	Alt-T
{aU}	0:22	Alt-U
{aV}	0:47	Alt-V
{aW}	0:17	Alt-W
{aX}	0:45	Alt-X
{aY}	0:21	Alt-Y
{aZ}	0:44	Alt-Z

{aF1}	0:104	Alt-F1
{aF2}	0:105	Alt-F2
{aF3}	0:106	Alt-F3
{aF4}	0:107	Alt-F4
{aF5}	0:108	Alt-F5
{aF6}	0:109	Alt-F6
{aF7}	0:110	Alt-F7
{aF8}	0:111	Alt-F8
{aF9}	0:112	Alt-F9
{aF10}	0:113	Alt-F10
{aF11}	0:139	Alt-F11
{aF12}	0:140	Alt-F12

CONTROL + KEY

{Nul}	0:3	Ctrl-@ 0
{Soh}	1:30	Ctrl-A 1
{Stx}	2:48	Ctrl-B 18
{Etx}	3:46	Ctrl-C 2
{Eot}	4:32	Ctrl-D 19
{Enq}	5:18	Ctrl-E 3
{Ack}	6:33	Ctrl-F 20
{Bel}	7:34	Ctrl-G 4
{Bs}	8:35	Ctrl-H 5
{Ht}	9:23	Ctrl-I 6
{Lf}	10:36	Ctrl-J 7
{Vt}	11:37	Ctrl-K 8
{Ff}	12:38	Ctrl-L 9
{Cr}	13:50	Ctrl-M 10
{So}	14:49	Ctrl-N 11

{Si}	15:24	Ctrl-O 12
{Dle}	16:25	Ctrl-P 13
{Xon}	17:16	Ctrl-Q 14
{Dc2}	18:19	Ctrl-R 15
{Dc3}	19:31	Ctrl-S 16
{Dc4}	20:20	Ctrl-T 17
{Nak}	21:22	Ctrl-U 21
{Syn}	22:47	Ctrl-V 22
{Etb}	23:17	Ctrl-W 23
{Can}	24:45	Ctrl-X 24
{Em}	25:21	Ctrl-Y 25
{Eof}	26:44	Ctrl-Z 26
{Esc}	27:1	ctrl-[27
{Fs}	28:43	Ctrl-\ 28
{Gs}	29:27	Ctrl-] 29
{Rs}	30:7	Ctrl-^ 30
{Us}	31:12	Ctrl-_ 31

{^Bs}	127:14	Ctrl-Backspace
{^@}	0:3	Ctrl-@ null
{^Break}	255:6	Ctrl-Break
{^Cr}	10:28	Ctrl-Enter
{^Del}	0:147	Ctrl-Del
{^Down}	0:145	Ctrl-Down
{^End}	0:117	Ctrl-End
{^Home}	0:119	Ctrl-Home
{^Ins}	0:146	Ctrl-Ins
{^Left}	0:115	Ctrl-Left
{^PgDn}	0:118	Ctrl-PgDn
{^PgUp}	0:132	Ctrl-PgUp
{^PrtScrn}	0:114	Ctrl-PrtScrn
{^Right}	0:116	Ctrl-Right
{^Tab}	0:148	Ctrl-Tab
{^Up}	0:141	Ctrl-Up

{^[}	27:26	Ctrl-[Esc
{^\}	28:43	Ctrl-\ fs
{^]}	29:27	Ctrl-] gs
{^^}	30:7	Ctrl-^ rs
{^_}	31:12	Ctrl-_ us
{^A}	1:30	Ctrl-A soh
{^B}	2:48	Ctrl-B stx
{^C}	3:46	Ctrl-C etx
{^D}	4:32	Ctrl-D eot
{^E}	5:18	Ctrl-E enq
{^F}	6:33	Ctrl-F ack

{^G}	7:34	Ctrl-G bel
{^H}	8:35	Ctrl-H bs
{^I}	9:23	Ctrl-I ht hor tab
{^J}	10:36	Ctrl-J lf
{^K}	11:37	Ctrl-K vt
{^L}	12:38	Ctrl-L ff
{^M}	13:50	Ctrl-M cr
{^N}	14:49	Ctrl-N so
{^O}	15:24	Ctrl-O si
{^P}	16:25	Ctrl-P dle
{^Q}	17:16	Ctrl-Q dc1 xon
{^R}	18:19	Ctrl-R dc2
{^S}	19:31	Ctrl-S dc3
{^T}	20:20	Ctrl-T dc4 xoff
{^U}	21:22	Ctrl-U nak
{^V}	22:47	Ctrl-V syn
{^W}	23:17	Ctrl-W etb
{^X}	24:45	Ctrl-X can
{^Y}	25:21	Ctrl-Y em
{^Z}	26:44	Ctrl-Z sub/eof

{^F1}	0:94	Ctrl-F1
{^F2}	0:95	Ctrl-F2
{^F3}	0:96	Ctrl-F3
{^F4}	0:97	Ctrl-F4
{^F5}	0:98	Ctrl-F5
{^F6}	0:99	Ctrl-F6
{^F7}	0:100	Ctrl-F7
{^F8}	0:101	Ctrl-F8
{^F9}	0:102	Ctrl-F9
{^F10}	0:103	Ctrl-F10
{^F11}	0:137	Ctrl-F11
{^F12}	0:138	Ctrl-F12