

APTCONTROL.HYPER

Andy Grifo

COLLABORATORS

	TITLE : APTCONTROL.HYPER		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY	Andy Grifo	July 20, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	APTCONTROL.HYPER	1
1.1	MAIN	1
1.2	COMMANDS_IDX	1
1.3	APTMAYCALL	1
1.4	ROGUERUN	2
1.5	FILE_RELATED	2
1.6	DISPLAY_RELATED	3
1.7	ACCESS_USER	3
1.8	LIBRARY_LIST	3
1.9	APPLYTEMPLATE	4
1.10	MSG	4
1.11	RAWMSG	4
1.12	SHOWFILE	5
1.13	SHOWGLOBALFILE	5
1.14	SHOWRAWFILE	6
1.15	EASYPROMPT	6
1.16	DROPLINE	7
1.17	ISDROP	7
1.18	SETBREAK	7
1.19	HOTKEY	9
1.20	HOTCUR	9
1.21	DOIT	10
1.22	PAGESYSOP	11
1.23	EDITSTATS	11
1.24	CALLFUNCTION	12
1.25	GETPASSWORD	12
1.26	HASEXPIRED	13
1.27	CHATMODE	13
1.28	FLUSHACCOUNT	13
1.29	CHAINAPT	14

1.30 RUNDOS	15
1.31 RUNEASYDOS	15
1.32 XFEROPTS	16
1.33 BUTTONOPTS	17
1.34 AUTOMSG	18
1.35 INFORM	18
1.36 HOTOPTS	18
1.37 SCROLLBODY	20
1.38 LOCKDOOR	21
1.39 UPDATEHISTOGRAMS	22
1.40 INITFILEAREA	22
1.41 FREEFILEAREA	22
1.42 SCANAREA	23
1.43 SELECTAREA	23
1.44 TOUCHINDEX	24
1.45 LOCALNEWSCAN	25
1.46 CONTINUESCAN	25
1.47 GLOBALNEWSCAN	25
1.48 REMOVEMARKED	26
1.49 UPLOAD	26
1.50 DOWNLOAD	26
1.51 LISTMARKED	26
1.52 NEXTAREA	27
1.53 PREVAREA	27
1.54 VIEWARCHIVE	27
1.55 FILEAREAINFO	27
1.56 FILTERMENU	28
1.57 FILTERSOFF	28
1.58 FILESTYLE	28
1.59 FINDDEFAULTBASE	29
1.60 MAILCMD	30
1.61 RUNPARADOOR	31
1.62 MARKFILE	32
1.63 SETSIG	32
1.64 GETSTATICSTR	33
1.65 GETSTATICVAR	34
1.66 GETSTR	34
1.67 GETVAR	35
1.68 Contacting Technical Support	35

1.69 DEVELOPERS_GUIDELINES	36
1.70 SETSTR	36
1.71 SETVAR	37
1.72 USERFLAG	37
1.73 GFXFLAG	40
1.74 MOVEUSER	40
1.75 DELETEACCOUNT	41
1.76 ALERTLOG	42
1.77 GETUSERDIR	43
1.78 REMOVEFILE	43
1.79 RETURNCODE	43
1.80 USEACCOUNT	44
1.81 SMALLCMD	44
1.82 CLASSICDEFS	45
1.83 _VAR	46
1.84 _STR	46
1.85 _CALL	46
1.86 _STVAR	46
1.87 _STSTR	47
1.88 FKEYS	47
1.89 CURSORDEFS	47
1.90 _FLG	47

Chapter 1

APTCONTROL.HYPER

1.1 MAIN

ApT-BBS Rexx Guide

Copyright (c) 1994 ApT-Design

~Developers~Guidelines~::~::~::~
~ApT-Rexx~Commands~::~::~
~ApT-Rexx~Classic~Defines~::~::~
~Scripts~ApTline~may~call~::~
~Running~rogue~scripts~while~user~online~::~
~Contacting~Technical~Support~::~

1.2 COMMANDS_IDX

ApT-BBS Rexx Command Index

~Alphabetical~List~of~Commands~::~
~File~Related~Commands~::~
~Display~Related~Commands~::~
~Accessing~User~Info~&~Storing~::~

1.3 APTMAYCALL

ScriptName	Called from/at
interlog.ap	Called just after entering password in.
interlogoff.ap	within the early logging off process.
logoff_final.ap	Just prior to BBS window closing after logoff.
carrierdrop.ap	Whenever a user exits abnormally from system.
chatsound_on.ap	Just prior to the chat-window appearing.

```
chatsound_off.apt | Just after the chat-window closing.
newuser_logon.apt | Whenever a user is found to be new to the system.
```

NOTE:

Some of the scripts will only be called depending on how much control of the BBS you are taking on yourself. If for example you are simply using the lowest level of control, the use of the inbuilt menu system, then certain scripts should as the "interlogoff.apt" script may be called, controlling your BBS entirely through arexx would not be the case.

1.4 ROGUERUN

Commands can be issued almost any-time while a user is online. SysOps who own WShell can make an easy ALIAS commands such as:

```
alias CHat "'address APTREXX[] chatmode"
```

to use the command one would use, from shell.

```
chat 0 ; to enter chatmode on line 0 ch 1 ; to enter chatmode on line 1
```

or even direct from the shell,

```
'address APTREXX0 chatmode<return>
```

or

```
'address APTREXX0 showglobalfile "ram:whatever.doc"
```

which would directly show the ram:whatever.doc to the screen.

```
alias CHat "'address APTREXX[] chatmode" alias DROpline "'address APTREXX[]
DROPLINE'"
```

1.5 FILE_RELATED

ApT-BBS Rexx File Related Commands

```
~~CONTINUESCAN~~~~ ~~~~DOWNLOAD~~~~ ~~~~FREEFILEAREA~~~~
~~FILEAREAINFO~~~~ ~~~~FILTERMENU~~~~ ~~~~FILTERSOFF~~~~
~~FILESTYLE~~~~ ~~~~FINDDEFAULTBASE~~~ ~~~~GLOBALNEWSCAN~~~~
~~INITFILEAREA~~~~ ~~~~LISTMARKED~~~~ ~~~~LOCALNEWSCAN~~~~
~~MARKFILES~~~~ ~~~~NEXTAREA~~~~ ~~~~PREVAREA~~~~
~~SCANAREA~~~~ ~~~~SELECTAREA~~~~ ~~~~TOUCHINDEX~~~~
~~UPLOAD~~~~ ~~~~VIEWARCHIVE~~~~ ~~~~XFEROPTS~~~~
```

1.6 DISPLAY_RELATED

ApT-BBS Rexx Display Related Commands

```
~~DOTIT~~~~~  ~~EASYPROMPT~~~~~  ~~MSG~~~~~
~~RAWMSG~~~~~  ~~SCROLLBODY~~~~~  ~~SHOWFILE~~~~~
~~SHOWGLOBALFILE~~~  ~~SHOWRAWFILE~~~~~
```

1.7 ACCESS_USER

ApT-BBS Rexx User Variable Commands

```
~~GETSTATICSTR~~~~~  ~~GETSTATICVAR~~~~~  ~~GETSTR~~~~~
~~GETVAR~~~~~  ~~GFXFLAG~~~~~  ~~SETSIG~~~~~
~~SETSTR~~~~~  ~~SETVAR~~~~~  ~~USERFLAG~~~~~
```

1.8 LIBRARY_LIST

ApT-BBS ApTControl.library List

```
~~ALERTLOG~~~~~  ~~APPLYTEMPLATE~~~~~  ~~AUTOMSG~~~~~
~~BUTTONOPTS~~~~~  ~~CONTINUESCAN~~~~~  ~~CHATMODE~~~~~
~~CHAINAPT~~~~~  ~~DELETEACCOUNT~~~~~  ~~DOWNLOAD~~~~~
~~DOTIT~~~~~  ~~DROPLINE~~~~~  ~~EASYPROMPT~~~~~
~~EDITSTATS~~~~~  ~~FREEFILEAREA~~~~~  ~~FLUSHACCOUNT~~~~~
~~FILEAREAINFO~~~~~  ~~FILTERMENU~~~~~  ~~FILTERSOFF~~~~~
~~FILESTYLE~~~~~  ~~FINDDEFAULTBASE~~~  ~~GETPASSWORD~~~~~
~~GETSTATICSTR~~~~~  ~~GETSTATICVAR~~~~~  ~~GETSTR~~~~~
~~GETVAR~~~~~  ~~GETUSERDIR~~~~~  ~~GFXFLAG~~~~~
~~GLOBALNEWSCAN~~~~~  ~~HASEXPIRED~~~~~  ~~HOTKEY~~~~~
~~HOTCUR~~~~~  ~~HOTOPTS~~~~~  ~~INFORM~~~~~
~~INITFILEAREA~~~~~  ~~ISDROP~~~~~  ~~LISTMARKED~~~~~
~~LOCKDOOR~~~~~  ~~LOCALNEWSCAN~~~~~  ~~MAILCMD~~~~~
~~MARKFILES~~~~~  ~~MOVEUSER~~~~~  ~~MSG~~~~~
~~CALLFUNCTION~~~~~  ~~NEXTAREA~~~~~  ~~PAGESYSOP~~~~~
~~PREVAREA~~~~~  ~~RAWMSG~~~~~  ~~REMOVEMARKED~~~~~
~~REMOVEFILE~~~~~  ~~RETURNCODE~~~~~  ~~RUNDOS~~~~~
~~RUNEASYDOS~~~~~  ~~RUNPARAGONDOOR~~~~~  ~~SCROLLBODY~~~~~
~~SCANAREA~~~~~  ~~SELECTAREA~~~~~  ~~SETSIG~~~~~
~~SETSTR~~~~~  ~~SETVAR~~~~~  ~~SETBREAK~~~~~
~~SHOWFILE~~~~~  ~~SHOWGLOBALFILE~~~~~  ~~SHOWRAWFILE~~~~~
~~SMALLCMD~~~~~  ~~TOUCHINDEX~~~~~  ~~UPDATEHISTOGRAMS~~~
~~UPLOAD~~~~~  ~~USEACCOUNT~~~~~  ~~USERFLAG~~~~~
~~VIEWARCHIVE~~~~~  ~~XFEROPTS~~~~~
```


1.9 APPLYTEMPLATE

NAME:

APPLYTEMPLATE

FUNCTION:

This function is very powerful in that it will 'apply/transpose' a saved user template from the ApTManager system to the current users account. This allows for a LOT of scope, generally the initial place that this function would be used is upon a new user logging onto the system for the very first time. You would apply a 'newuser' template to his account that would setup things such as ratio's etc..

```
'APPLYTEMPLATE' "newuser"
```

Would load in the "Tplates:UserTemplates/NewUser.TRP" file and apply its contents pertaining to the set flags to the user that is currently online.

SEE-ALSO:

ApTManager Documentation on UserTemplates.

1.10 MSG

NAME:

MSG

FUNCTION:

Used to display text to the screen - The string can be any length and can use AML codes for additional control over the display.

EXAMPLE:

```
'MSG' "Line1\n,Line2\nHello world,.. Press a key %WAITKEY."
```

1.11 RAWMSG

NAME:

RAWMSG

FUNCTION:

This function is a raw text printing function. No actual function is carried out on the text, and is purely sent out to the screen AS IS. This is obviously going to be faster than all the other

printing routines but at a cost of not removing any escape codes and not having the AML embedded commands.

EXAMPLE:

```
'RAWMSG' "A would-be-AML %WAITKEY. command, is printed as"
```

```
"A would-be-AML %WAITKEY. command, is printed as"
```

1.12 SHOWFILE

NAME:

SHOWFILE

FUNCTION:

Shows (displays) a file within the 'tplates:' directory, adding the '.tpl' extension onto the end of the supplied filename. This command also uses AML codes, if required.

EXAMPLE:

```
/*  
** tplates:menus/mymenu.tpl displayed to screen.  
**/  
'SHOWFILE' "menus/mymenu"
```

RETURNS:

SEE-ALSO:

~SHOWGLOBALFILE~ ~SHOWRAWFILE~

1.13 SHOWGLOBALFILE

NAME:

SHOWGLOBALFILE

FUNCTION:

Same as the SHOWFILE command but requires a full path and filename rather than relying on tplates: and .tpl being added to the given name.

Uses AML.

EXAMPLE:

```
/*  
** displays the file ram:wishingwell.txt to the
```

```
** screen.  
**/  
'SHOWGLOBALFILE' "ram:wishingwell.txt"
```

SEE-ALSO:

~SHOWFILE~ ~SHOWRAWFILE~

1.14 SHOWRAWFILE

NAME:

SHOWRAWFILE

FUNCTION:

This allows the displaying of a file to the screen that is displayed "as is", without any removing of ansi-codes or any form of parsing.

EXAMPLE:

```
'SHOWRAWFILE' "t:LhaOutput."lineno
```

SEE-ALSO:

~SHOWFILE~ ~SHOWGLOBALFILE~

1.15 EASYPROMPT

NAME:

EASYPROMPT

TEMPLATE:

```
"STRING/A, LEN/N, CASE/S, LETTER/S, NUMBER/S, ASCII/S, UPFIRST/S, PUNCT/S"
```

FUNCTION:

Allows EASY prompting, ..

EXAMPLE:

```
DefaultText = "Hello World"  
'EASYPROMPT' DefaultText "LEN=30 ASCII UPFIRST"  
  
NewText = result  
'MSG' "You typed, "NewText  
  
[Would display]
```

```
"Hello World"<Cursor here, and add>", my name is Joe Bloggs."
```

The user could then backspace and remove the "Hello World" text or add to it, if "my name is Joe Bloggs" was simply added:

```
"Hello World, my name is Joe Bloggs."
```

RETURNS:

```
Pointer to string..
```

SEE-ALSO:

```
~FIREPROMPT~
```

1.16 DROPLINE

NAME:

```
DROPLINE
```

FUNCTION:

```
Drops the line,.
```

EXAMPLE:

```
if(BadBadUser) then 'DROPLINE'
```

1.17 ISDROP

NAME:

```
ISDROP
```

FUNCTION:

```
Finds out what the state of the line is, dropped or not.
```

EXAMPLE:

```
'ISDROP' ; if(result=1) then break
```

RETURNS:

```
~BOOL~ - TRUE denotes line has dropped.
```

1.18 SETBREAK

NAME:

SETBREAK

FUNCTION:

This function allows your arexx menus to define what characters will interrupt your menus. If you have only 'G' for logoff, 'C' for page sysop and 'F' for menus, as well as 'M' for message areas, then you would only wish for characters: "GCFM" to interrupt a given menu, while all other characters have no effect.

Uses for this command would be at the start of each of your 'menu' routines, Your main menu, utilities menu, etc.. simply entering the characters that ought to break your menus display.

You should also set up a the ^C (Ctrl+C) break code, which can be specified as '3'x - This would allow your users to break the display of templates using this key (as well as others..) - If you are in some form of loop then you should also check for the ^C keypress and 'not' refresh a menu.

In some instances, such as the newuser_logon.apt script, it is wise to turn off all instances of 'breaking' - thus one would use the command: 'SETBREAK' '0'x to turn off such instances.

EXAMPLE:

```
'SETBREAK' "GCFM"
```

The above only allows your menus to be interrupted, while being displayed by the characters G C F M.

```
'SETBREAK' '0'x
```

The above would stop NO characters, so your menu(s) would display without interruption by any characters.

```
'SETBREAK' '1'x
```

This allows ANY character press to stop your menu(s).

It may be best to use a define within your scripts such as:

```
STOP_NONE = '0'x      /* Hex value 0, not ascii value of namesake. */  
STOP_ALL  = '1'x      /* Hex value 1, not ascii value of namesake. */
```

then you can simply use:

```
'SETBREAK' STOP_NONE /* To have no characters break your menu display */  
'SETBREAK' STOP_ALL  /* To have all characters break your menu display  
*/
```

1.19 HOTKEY

NAME:

HOTKEY

FUNCTION:

Allows the sending out of a initial string, before returning the very first character that has been pressed by the user.

'HOTOPTS' command which allows various aspects of the hotkey-rexx command to be changed. See HOTOPTS for more information on control of this command.

EXAMPLE:

```
'hotkey' "Press a key =>"
'msg' "You pressed key " result

'hotkey' ""
'msg' "You pressed key " result "\n"
```

RETURNS:

Character typed.

SEE ALSO:

See ~HOTOPTS~ for more information on control of this command.
also ~HOTCUR~ for other uses.

1.20 HOTCUR

NAME:

HotCur(string) (a0)

FUNCTION:

In many ways this is much the same as the standard HOTKEY command but with a few differences. The command returns exactly the same returns as HOTKEY but the difference is within its return of the CURSOR keys.

The command can be used to get CURSOR key movements from the user without much difficulty than otherwise would be the case if using the standard HOTKEY command. The other thing to note is that the 'HOTOPTS' MIN/MAX command is not taken into account here, and all keys in the range of 0-255 are returned back to the user. So if you wish to filter out keys of a certain range then you should use 'HOTOPTS' "SYSOPKEY" to see who the last typer was.

When ever it is required to monitor for a cursor key, the first

thing that you have to understand is that cursor movements are made up of 3 characters.

?[A indicates that the cursor is to be moved in the UP direction.

? is the ESCape character

[is the second character in the phase-chain

A,B,C or D are the actual Cursor direction requests.

If you monitor using the standard HOTKEY command for the above, it is easy to see that you require 3 different HOTKEY calls in order to find out what cursor key has been pressed. All of this is slower than normal because of the time it takes to call the HOTKEY command, 3 times in all. It just is not really worth it, unless you can call it once and expect it to return the next character as the direction. This is what the HOTCUR command does. It handles the monitoring of the cursor key(s) and returns one of the following values when pressed (both for local and serial line, unlike HOTKEY)

- It returns ONE single character of which the defines show below.

```
CUR_UP      = 252  /* the rawkey defines */
CUR_DOWN    = 253
CUR_RIGHT   = 254
CUR_LEFT    = 255
```

EXAMPLE:

```
'HOTCUR' "" ; rawkey=result ; key=upper(rawkey)
```

```
select
  when (rawkey=CUR_UP)      then 'msg' "You pressed Cursor up"
  when (rawkey=CUR_DOWN)    then 'msg' "You pressed Cursor Down"
  when (rawkey=CUR_RIGHT)   then 'msg' "You pressed Cursor Right"
  when (rawkey=CUR_LEFT)    then 'msg' "You pressed Cursor Left"
  otherwise
end
```

RETURNS:

Exactly the same as HOTKEY command but Cursor keys are returned as SINGLE characters of 'FC'x 'FD'x 'FE'x 'FF'x (Hex, or use above defines or 252,253,254,255 Decimal)

SEE-ALSO:

~HOTKEY~ ~HOTOPTS~

1.21 DOIT

NAME:

DOTIT

FUNCTION:

This function prints a '.' (dot), X positions from the current cursor position. This is generally used once a line such as the within the example below.

EXAMPLE:

```
msg "Enter your name =>"
dotit(18);

/* Prompt calls etc.. */

.

.

/* results in (minus speechmarks) */

"Enter your name =>                ."
```

1.22 PAGESYSOP

NAME:

PAGESYSOP

FUNCTION:

Actually does just that, pages for the system operator via the internally built chat conferencing system.

EXAMPLE:

```
select
  when (key='C') then do
    msg "Calling for the System operator..."
    'PAGESYSOP'
  end
.
.
.
otherwise
end
```

1.23 EDITSTATS

NAME:

EDITSTATS

FUNCTION:

Takes the user to one of the two inbuilt editing-stats options, either "edit personal" or "edit terminal" options.

EXAMPLE:

```
'EDITSTATS' 0 /* Edits 'Personal' preferences.. */  
'EDITSTATS' 1 /* Edits 'System' preferences.. */
```

1.24 CALLFUNCTION

NAME:

CALLFUNCTION

FUNCTION:

Calls one of the various forms of "internal" new-user (or other) functions, normally asking for information such as computer-make, etc.

EXAMPLE:

```
'CALLFUNCTION' functionnumber  
  
'CALLFUNCTION' COMPUTERTYPE_CALL /* providing you have the defines.. */
```

SEE-ALSO:

~_CALL~Defines~

1.25 GETPASSWORD

NAME:

GETPASSWORD

FUNCTION:

This function is most handy in that it will allow you to get a string from the user, but whenever a key is pressed it will display the # (HASH) symbol in the characters place.

EXAMPLE:

```
'msg' "Enter your password to enter private area =>"  
  
'GETPASSWORD' 8  
  
password=result
```

```
'msg' "You entered:"result
```

```
"Enter your password to enter private area =>###<etc..>"
```

```
"You entered:Wobble"
```

The above would only allow a maximum of 8 characters to be entered.

RETURNS:

Password string entered.

1.26 HASEXPIRED

NAME:

HASEXPIRED

FUNCTION:

This will allow you to know if the users account has come up for review or not. You should then take actions if this is true, generally this would be used within the interlog.apt (or module) file, giving some indication that he has come up for review before giving him a chance to page for the sysop and/or leave feedback before he is dropped off.

RETURNS:

Boolean - 0 or 1 depending if he has come up for review or not (1 denoting user has come up for review)

1.27 CHATMODE

NAME:

CHATMODE

FUNCTION:

Enters chatmode, directly.

SEE-ALSO:

~PAGESYSOP~

1.28 FLUSHACCOUNT

NAME:

FLUSHACCOUNT

FUNCTION:

Flushes the users information from memory to disk. This should be used when you need to make sure changes are noted by the system, such things as a "guru" or a "reset" would render memory contained changes un-seen during new calls by the user.

EXAMPLE:

```
'ALERTLOG' "Dropped carrier - There is a fine for this offence!"

'GETVAR' BYTESLEFT_VAR ; bytesleft = result - 10240
'SETVAR' BYTESLEFT_VAR bytesleft

'FLUSHACCOUNT'

'ALERTLOG' "10K of bytes removed as a penalty for carrier drop."
```

1.29 CHAINAPT

NAME:

CHAINAPT

FUNCTION:

This is a most powerful and useful function; At times you may require a lot of various different code to be placed within your Core.apr files. If this is the case then sometimes it makes sense for portions that are only used very little (maybe even a game door), or are large in their own right, to be placed in a script file by themselves. This command would then be used to call the script, and once the script has been called it will then be used 'there and then'. Once the script has finished and returned, your current script will then take over from where it left off. If you wish to return any variables back then you would use the 'RETURNCODE' command.

```
/*
** User just pressed 'A' for a special menu to be loaded.
**/
'CHAINAPT' <script> /* aptrexx: and .apr are added by the command */
if(result=<whatever>) then /* do something */
```

RETURNS:

Success - The command will return whatever the 'script you called' has stored within its 'RETURNCODE' setting.

SEE ALSO:

'RETURNCODE'

1.30 RUNDOS

NAME:

RUNDOS

FUNCTION:

Allows the running of a DOS program with the callers own argument parsing string. Re-direction will have to be passed within the string, as well as any other required settings for the dos program to actually work.

This routine should only be used if you know what you are doing. It allows far greater control over the way dos programs are called.

EXAMPLE:

'RUNDOS' "run >nil: newshell >nil: %HANDLER. from t:game.script %RAW."

This would have the effect of calling a standard DOS BATCH and all actions would be acted out within the ApTline window.

RETURNS:

SEE ALSO:

~RUNEASYDOS~

1.31 RUNEASYDOS

NAME:

RUNEASYDOS

FUNCTION:

Allows the running of DOS programs the easy way. Unlike the RUNDOS command, this command adds all of the re-direction handlers itself. This leaves the caller free to supply a very basic string for a game/utility to be called. Generally this would be the command that would be used. It acts out exactly the same as the MENU RunDos function.

EXAMPLE:

'RUNEASYDOS' "%RAW. DOORS:KNOT/KNOT %LNAME."

This would have the effect of calling a game with RAW mode on. The arguments to the right of the %RAW. command are the path to the door and any other arguments that may follow. This is the easiest way of running doors that do not require batch files to be worked.

RETURNS:

SEE ALSO:

~RUNDOS~

1.32 XFEROPTS

NAME:

XFEROPTS

TEMPLATE:

"CLEARLIST/S, TAGFILE/K, SENDFILES/S, UPDATERATIO/S"

FUNCTION:

Allows control over tagging and sending of global files, with or without ratio stats updating.

CLEARLIST - Clears the download list.
TAGFILE - Tag a filename-entry to the download list.
SENDERFILES - Send all of the tagged files as soon as possible.
UPDATERATIO - Update the users bytes etc.. ratio after download.

EXAMPLE:

```
/*
** Send ONE single file to the user as soon as possible.
** The 'TPlates:sys/sys_protocols.tpl' is always shown before
** the upload takes place, allowing the user to change his
** protocol.
**/
'XFEROPTS' 'CLEARLIST TAGFILE "RAM:File1.LHA" SENDFILES'

/*
** Clears the download list and then tags 3 files for
** download. The 'SENDERFILES' command is then issued along
** with the UPDATERATIO which updates the users bytes etc..
**/
'XFEROPTS' 'CLEARLIST'
'XFEROPTS' 'TAGFILE "RAM:File1.LHA"'
'XFEROPTS' 'TAGFILE "RAM:File2.LHA"'
'XFEROPTS' 'TAGFILE "RAM:File3.LHA"'
'XFEROPTS' 'SENDERFILES UPDATERATIO'
```

RETURNS:

1.33 BUTTONOPTS

NAME:

BUTTONOPTS

TEMPLATE:

```
"LOADBANK/K,INITBANK/S,DISPLAY/S,USEBANK/S,GETPOS/S,SETPOS/N,CLEARBUTTON/S"
```

FUNCTION:

Using a combination of commands related to this command enables various display functions relating to the 'button/highlight' look.

LOADBANK="tplates:button/main.bank" (Path to bank to load)

INITBANK - Simply re'init's the bank in question to defaults.

DISPLAY - Forces the entire button list to be 'refreshed'

USEBANK - Actually uses the loaded bank. Cursor appears at the first button and input continues until a key has been pressed.

GETPOS - Returns the number of the button currently selected within result.

SETPOS - Forces a button number to be the 'selected' one.

CLEARBUTTON - Simply clears a 'highlighted' button.. it is only VISUAL and does not affect the current setting of the button. Used with such things as when you have more than one button bank displayed on the same screen, but wish to keep the cursor highlighted on only one button bank.

EXAMPLE:

See included example files.

RETURNS:

Various, .. However, the ones that are important are:

LOADBANK - returns 0 for failure.

USEBANK - returns the key that was actually pressed within result. This can then be used to do various things depending on what the script requires.

SEE ALSO:

Included example rexx files and Manager 'BUTTONDATA' rexx command.

1.34 AUTOMSG

NAME:

AUTOMSG

TEMPLATE:

B=Body/A/K, A=Area/A/N, T=To/A/K, F=From/A/K, S=Subject/A/K,
P=Private/K/S, U=Upload/K

FUNCTION:

Allows the posting of messages to a user who is currently online.

Body - actual ascii text file to be used as the base of the message.
Area - The area number the message is to be placed into.
To - The user who the message is directed to.
From - The entity that the message is addressed from.
Subject - The subjet text.
Private - If supplied the message is classed as private.
File - Allows the sending of a file along with the message. Overrides
the 'Subject' token if it is also supplied.

RETURNS:

1.35 INFORM

NAME:

INFORM

FUNCTION:

Informs ApT-Server what the user of the line is currently up to.
This can be used to inform on such things as 'entering file area'
and such. However, it is worth taking note that ApTline itself may
also be updating the information for the current function. If this
is the case then it is wise to leave it to ApTline ...

EXAMPLE:

'INFORM' "User entering 'The wise old game of yar..'"

1.36 HOTOPTS

NAME:

HOTOPTS

TEMPLATE:

"MIN/K/N,MAX/K/N,ALL/S,RESET/S,SYSOPKEY/S"

FUNCTION:

This function setup the options for use with the Hotkey(..) command.

The Character set is based on a 0-255 character scale, RETURN being positioned at 10 and the normal QWERTY keyboard characters being positioned at 32 - 127.

So, in normal use whenever you wish to have a key returned from the user, you are going to require, most of the time, characters in the range of: 0-127 , with the outer limit 128-255 being filtered out.

This way it allows for the sysop to program hidden functions in menus that monitor for keypresses greater than 127, for such things as the Function keys, based around positions 240-249, and sysop cursor presses a littler further.

MIN & MAX allow you to change the bounds of the "user monitored" keypresses, and filters out characters not within that range, thus never quite returning them to the rexx script.

MIN & MAX are normally set to 0 and 127 (7F Hex).

ALL allows the full range of 0-255 characters to be returned if the user hits any combination of key-presses. Not normally used, but sometimes it may become required.

"RESET" - This resets the values to the default, and both MIN and MAX contain 0 and 127, default values.

"SYSOPKEY" - This returns a 0 or a 1 depending on WHO typed the last character in the HOTKEY command. 0 is returned if the key was pressed at the "Local Side", and 1 if the keypress was typed at the "Connected, terminal side." - This allows you to obviously test to see if the key was SYSOP pressed, and act on it, but not if the user accessed it.

EXAMPLE:

```
/*
** Only ever return USER key-presses in
** the range of 0-127, standard default.
**/
'HOTOPTS' "MIN=0 MAX=127"

/*
** Return ALL of the USER key-presses in
** the range of 0-255.
**/
'HOTOPTS' "ALL"

/*
** Reset the MIN & MAX values to
```

```

** 0-127.
**/
'HOTOPTS' "RESET"

/*
** Find out who pressed the last key,
** was it from the Console Side, or the
** Connected-Terminal side.
**/
'HOTOPTS' "SYSOPKEY" ; typer=result
if( typer ) then 'msg' "Console -> SysOp Hit a key, wow!"
else          then 'msg' "Terminal -> User Hit a key, wow!"

```

RETURNS

Depending on command, who pressed the last key.

SEE ALSO:

~HOTKEY~

1.37 SCROLLBODY

NAME:

```
'SCROLLBODY'
```

TEMPLATE (of-sorts..):

```
'SCROLLBODY' <filename> <headersize> [continue scroll flag]
```

FUNCTION:

This function is intended to be used for any form of page scrolling. It allows for the placement of a header at the top portion of the screen and then for a body to be scrolled while keeping the header intact. After this the command can be re-used along with the continue-scroll-flag set and a filename pointing to a 'footer' which would finish off the display according to ApT Standards.

EXAMPLE:

```

/*
** We first display our header, .. this is 3 lines long as one
** can see. We also turn OFF the cursor for FASTER displays.
**/
'msg' "@\o0Header1\nHeader2\n"
'msg'
"\c1AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA\c7\n"

/*
** We now pass our body file (the file to be scrolled within the window
** yet not over-writing our previous header by supplying the filename

```

```
** along with the number of lines that our header took up. So in our
** example header this would be 3 (3 lines for the header)
**/
'SCROLLBODY' "ram:Message.TXT" 3

/*
** Once the body has been been scrolled it is normal to finish off with
** 'tear line(s)' of some form to close the windows display. We would
** do as we did in the previous command but this time we would supply
** a extra flag of '1' which would denote that it is to be 'scroll
** appended' to the very end of the previous text.
'SCROLLBODY' "ram:Footer.TXT" 3 1

'msg' "\o1"
```

NOTES:

It is VERY IMPORTANT that whenever this command is used the programmer first turns the cursor OFF, this results in a very greater speed increase than would be of normal. Obviously once the routine has been finished with the cursor can be re-turned on.

The command will act just like a 'SHOWFILE' command if the user can not take ansi etc.

The routine is quite logical and is ideal for appending different body files onto the end of a previous one and thus continuing the scroll. To append always make quite sure you supply the '1' flag as the last argument.

RESULTS:

Success - Boolean 0 or 1 within result.

1.38 LOCKDOOR

NAME:

```
'LOCKDOOR' <Arguments>
```

TEMPLATE:

```
"NAME/A, ARGS/K"
```

FUNCTION:

Allows the running of a ApT-Only programmed module or door.

EXAMPLE:

```
'LOCKDOOR' "apt:modules/interlogin args=CONFIGURE");
```

would actually run the door called "interlogin" from "apt:modules/" and would then pass the LINENUMBER (so the program knows exactly what line it is to communicate to) followed by the arguments that

you supply 'for that program' (it may not require any, see documentation for the module/door being used)

1.39 UPDATEHISTOGRAMS

NAME:

UPDCALLHIST

FUNCTION:

Update the various histograms for apt, .. used partly by the manager to view the status of the calls over a period of time.

The information is written to the directory:

apt:Histograms/

NOTES:

This command will be moved into a Sub-Command at some stage.

1.40 INITFILEAREA

NAME:

INITFILEAREA

FUNCTION:

This HAS to be called when you taking a user to your file section. It may be that you only require to call it once, but it has to have one 'FREEFILEAREA' at some stage to free resources. Normally you would have the 'INITFILEAREA' call at the start of your rexx file area entrance, and the 'FREEFILEAREA' call at the end before you return out of the subroutine.

EXAMPLE:

See various included source files for examples.

RETURNS:

1.41 FREEFILEAREA

NAME:

FREEFILEAREA

FUNCTION:

Whenever a 'INITFILEAREA' command is used, there must be a 'FREEFILEAREA' at some stage, generally at the end of the rexx-routine. This frees various resources back to the system.

EXAMPLE:

See the various included sources..

RETURNS:

1.42 SCANAREA

NAME:

SCANAREA

FUNCTION:

This function starts the 'Displaying' of the actual filenames and descriptions in either the normal display format, or the more advanced display format.

NOTES:

You are required to have Initiated your file area before attempting to use this command. See examples for information on when and how to call this command and other file area sub-commands.

EXAMPLE:

See the various included sources..

RETURNS:

1.43 SELECTAREA

NAME:

SELECTAREA

FUNCTION:

Calls up the 'File Area Selection' screen, asks the user to select a file area and keeps track of such, but does no actual loading of the index.

EXAMPLE:

See various included examples.

RETURNS:

Returns the base selected or -1 denoting the user pressed RETURN (which should be deemed as a QUIT from file area code, or area that they do not have access to)

1.44 TOUCHINDEX

NAME:

TOUCHINDEX

TEMPLATE:

FileArea in Numeric form of 0 onwards.

FUNCTION:

Attempts to lock into a given file index (numbered from 0 onwards..)

Once a file area has been initiated and you have decided that you either have a 'area' number from the user via the SELECTAREA command, or have a area you wish to 'force' touch, then this is the command that needs to be used.

EXAMPLE:

```
/*
** This method is what one would may normally have within
** a file area enviroment.
**/
'INITFILEAREA'
'SELECTAREA' ; selected_area = result

if( selected_area == -1) then do; Sharp_Exit = 1; end
else 'TOUCHINDEX' selected_area

/*
** OR, you could opt for the 'forced' choice of area,
** a example of such maybe rather than bringing up the
** 'SELECTAREA' requester, default by forcing the user
** into a Uploads area, or Default file area.
**/

'INITFILEAREA'

my_forced_choice_of_area = 13

TOUCHINDEX' my_forced_choice_of_area
if(result==0) then FAILED_TO_LOAD_INDEX
else EVERYTHING_WENT_FINE
```

RETURNS:

BOOL, 0 for failure, otherwise normal.

1.45 LOCALNEWSCAN

NAME:

LOCALNEWSCAN

FUNCTION:

The user is given the chance of scanning for new files SINCE the last date he was on the system, but scanning is only done for the file area that the user is 'within' at the moment of selection.

EXAMPLE:

See the various included sources.

1.46 CONTINUESCAN

NAME:

CONTINUESCAN

FUNCTION:

Once a FileArea has been touched, and in order for the user to have any descriptions displayed, the need for this command is required. It displays the given file areas until the user selects a command which indicates a abort from the 'description scanning', the user may still resume scanning from the position he left by re-calling this command.

EXAMPLE:

See the various examples.

RETURNS:

1.47 GLOBALNEWSCAN

NAME:

GLOBALNEWSCAN

FUNCTION:

The user is given the chance of scanning for new files SINCE the last date he was on the system. Scanning is GLOBAL and all file areas that the user has access to are scanned.

EXAMPLE:

See the various included sources.

1.48 REMOVEMARKED

NAME:

REMOVEMARKED

FUNCTION:

Gives the user a chance to remove files that are not required for download.

1.49 UPLOAD

NAME:

UPLOAD

FUNCTION:

Allows the user to upload a file to the currently touched index.

1.50 DOWNLOAD

NAME:

DOWNLOAD

FUNCTION:

Currently marked files are sent to the user, first giving the user the chance to alter his protocols etc.

1.51 LISTMARKED

NAME:

LISTMARKED

FUNCTION:

Lists all currently marked files.

1.52 NEXTAREA

NAME:

NEXTAREA

FUNCTION:

Advances automatically onto the 'Next' file area that the user has access to. The area is auto-touched and generally everything is done for you within this instance. Allows for quick and easy area advancements. Upon reaching the end of a file area boundy (the maximum in the direction) the function then stars once more from the start, so acts in a "Circle" format.

1.53 PREVAREA

NAME:

PREVAREA

FUNCTION:

Same as previous example but in the reverse direction.

1.54 VIEWARCHIVE

NAME:

VIEWARCHIVE

FUNCTION:

Asks the user for a file-number, and then displays the contents of said file to the screen.

1.55 FILEAREAINFO

NAME:

FILEAREAINFO

FUNCTION:

Returns various Strings pertaining to that of the FileArea that the user is currently within. Including access to various variables that may at times be required.

EXAMPLE:

```
'FILEAREAINFO' "TOTALFILES" ; totalfiles = result
'FILEAREAINFO' "AREANUMBER" ; areanumber = result
'FILEAREAINFO' "FILEPOS"      ; position   = result
'FILEAREAINFO' "CREDITS"      ; credits   = result
'FILEAREAINFO' "AREANAME"     ; areaname  = result
'FILEAREAINFO' "INDEXSORT"    ; indexsort = result
'FILEAREAINFO' "DIRECTION"    ; direction = result
'FILEAREAINFO' "FILTER"       ; filter     = result
'FILEAREAINFO' "FILESMARKED"; filesmarked= result
```

SEE-ALSO:

AML Variants, and %STRWID. AML command.

The same variables can also be displayed to screen more **easily** by using the AML variants. The same token names are used as in the above example, simply applying them in the form of.

%TOTALFILES. etc.

1.56 FILTERMENU

NAME:

FILTERMENU

FUNCTION:

Gives the user a chance to set up a filter which can then be used to locate files that are 'patternmatched' - Used in conjunction with Global-scanning produces very powerful search criteria.

1.57 FILTERSOFF

NAME:

FILTERSOFF

FUNCTION:

Turns OFF all file area filters that maybe on.

NOTE:

This command will be moved into a sub-command at some stage.

1.58 FILESTYLE

NAME:

FILESTYLE

TEMPLATE:

"RESET/S,OLD/S,NEW/S"

FUNCTION:

By default the file-area displayer system is user-defaulted by what form of ANSI the user has selected. If they are unable to display ANSI for some reason, or have one or two specific ANSI flags turned off, then they would find the file-displayer would be that of the "old" style BBS display. Other than that they would have the more "newer" and advanced file-display system shown while viewing file descriptions.

In some instances it may be wishable to override the "computer" defaulted selection of style to that of the user or sysops choice for a area/section. Maybe contending to only give one selection to users who have fast modems as opposed to slow modems.

Whatever the instance, this option allows just that, the overriding of the method used by the selection of your own.

EXAMPLE:

```
/*
** This example tells the computer to re-compute the
** normal file-area display method for the current
** user, and stick to it until changed.
**/
'FILESTYLE "RESET"

/*
** Both of the following either FORCE the user to have
** the old style of file-display, or the newer more
** advanced display - It may be that you would like
** to use one or the other in part, and then "RESET"
** back to normal after exiting the routine.
**/
'FILESTYLE' "OLD"
'FILESTYLE' "NEW"
```

1.59 FINDDEFAULTBASE

NAME:

FINDDEFAULTBASE

TEMPLATE:

"START/S,END/S"

FUNCTION:

Internally sets the requested filearea number and returns said value back to caller, yes, in other words it finds out a area that the user has access to in relation to the "start" or "end".

This function comes into its own when you wish to take the user to a given file area but know very little about any areas they might have access to. Simply plucking a area number out thin area and using it with various other commands is not the done thing. This command is used to locate a area that hinges on the boundry of either the "first" or the "last" base that the user has access to.

EXAMPLE:

```
'FINNDEFAULTBASE' "START" ; first_base_user_has_access_to = result

'FINNDEFAULTBASE' "END"    ; last_base_user_has_access_to = result

/*
** Notice here we do not supply the "START" token, this is
** because the function will default to that of the "START"
** token of one is not supplied.
**/
'FINNDEFAULTBASE' ; first_base_user_has_access_to = result
```

RETURNS:

It will return the area number that the user has access amongst other hidden functions.

1.60 MAILCMD

NAME:

```
'MAIL_CMD'
```

TEMPLATE:

```
"FEEDBACK/S,HASMAIL/S,SHOWMAIL/S,LIST/N,AREA/N"
```

FUNCTION:

This function is a multi-caller one, it allows the caller of the command to supply one of a number of argument strings. Depending on the one given, the command will then act out in a different way.

```
FEEDBACK
HASMAIL
SHOWMAIL
LIST          /* Same as the old 'MAILLIST' <signumber> command */
AREA
```

EXAMPLE:

```
'MAIL_CMD' FEEDBACK    /* Force user into leaving feedback */

/*
** Check if user has mail
**/
'MAIL_CMD' HASMAIL
if(result=1) then do
    /* user has mail/
    'MAIL_CMD' SHOWMAIL
end

'MAIL_CMD' "LIST 1" /* Displays Msg area selector, with SIG 1 areas */

[new example]

/*
** This function acts rather like its partner LIST, only this
** command will take the user directly into a message <area number>.
** This allows for 'combination menus' to be created that would
** contain various menu items along with easy access to a associated
** 'message base'.
**/
'MAIL_CMD' "AREA 2"
```

EXAMPLE:

```
'MAILAREA' 2 /* Jump into mail area number 2 */
```

RETURNS:

Success - Boolean 0 or 1 (or other, depending on command)

1.61 RUNPARADOOR

NAME:

RUNPARAGONDOOR

FUNCTION:

This allows the running of a Paragon style door. It is sad to say that this interface works much better than the original Paragon door interface in a few key-respects, generally the Paragon door call from StarNet/Paragon does not allow the passing of arguments along to a door. Calling a paragon door allows passing of full arguments, something both Paragon and Starnet never allowed, which is odd.

EXAMPLE:

```
'RUNPARAGONDOOR' "DOORS:SPACEEMPIRE/SPCEMP JOE_BLOGS RELOAD 23"
```

would correctly run the door "DOORS:SPACEEMPIRE/SPCEMP" and pass the rest of the string as arguments along to the door in question, thus allowing newly programmed paragon doors to support arguments.

1.62 MARKFILE

NAME:

MARKFILE

TEMPLATE:

"QUERY/S,CLEAR/S" & NO argument.

FUNCTION:

Allows the user to mark a given file, while outside of the file description display, but still within a touched index.

EXAMPLE:

```
/*
** Ask the user to mark a file by giving a number
** etc. of the file in question, to be marked.
**/
'MARKFILE'

/*
** Do NOT ask the user to mark a file, but simply Query if
** the user HAS any files marked, returning 1 for Yes,
** and 0 for no.
**/
'MARKFILE' "QUERY" ; got_any_files = result
if( got_any_files==1 ) then 'msg' "We have some files for download
Sir.."
else                          'msg' "We have no files for download, don't
you know.."

/*
** Clears any marked files from the list.
**/
'MARKFILE' "CLEAR"
```

SEE ALSO:

DOWNLOAD

1.63 SETSIG

NAME:

SETSIG

TEMPLATE:

```
"NEWSIG/N,QUERY/S"
```

FUNCTION:

Allows the changing of a sig to a value of your choice (0-255), returns either the "previous sig value" or the "current sig value"

EXAMPLE:

```
/*
** Set the sig value to 2, and store the 'previous' sig value
** in a variable for later 'replacing' if needs be.
**/
'SETSIG' "NEWSIG=2" ; old_sig = result

/*
** Find out what the 'current' sig value is
** without actually changing the sig value.
**/
'SETSIG' "QUERY" ; current_sig = result
```

1.64 GETSTATICSTR

NAME:

```
GETSTATICSTR
```

FUNCTION:

This will return a string of information back to the caller that is based on the users variables. However, the reason this is a 'static' command (data can not be written to..) is that the values returned are computed values. As a example, a date string will be converted from integer data into string format and then returned back, in its raw format it would just be a string of numbers meaning nothing to the caller.

EXAMPLE:

```
'GETSTATICSTR' 0
'msg' "last date called: "result

'GETSTATICSTR' 1
'msg' "last time called: "result

'GETSTATICSTR' 2
'msg' "Your quote of the day is: "result
```

RETURNS:

Various strings depending on the command number given.

SEE ALSO:

~_STSTR~Defines~

1.65 GETSTATICVAR

NAME:

GETSTATICVAR

FUNCTION:

This is much the same as the GETSTATICSTR in that it returns computed values rather than raw data. This differs from that of the GETSTATICSTR in that it returns integer values rather than strings.

EXAMPLE:

```
'GETSTATICVAR' DAYS2REVIEW_STVAR
'msg' "your account comes up for review in "result" days"

'GETSTATICVAR' DAYSLASTCALL_STVAR
'msg' "You called this bbs "result" days ago"

'GETSTATICVAR' BAUDRATE_STVAR
'msg' "BaudRate Connection: "result"
```

RETURNS:

Other return strings are available, such as returns for the DAY, MONTH, YEAR values pertaining to that of the users Date of Birth, etc.

SEE-ALSO:

~_STVAR~Defines~

1.66 GETSTR

NAME:

GETSTR

FUNCTION:

Returns STRING information pertaining to the current user that is online, more easier to use this command to find information such as name, address, from a user rather than talking directly to the actual user-structure.

EXAMPLE:

```
'GETSTR' USERNAME_STR
'MSG' "Your user name is " result

'GETSTR' HANDLE_STR
'MSG' "Your user handle is " result
```

RETURNS:

String information..

SEE-ALSO:

~_STR~Defines~

1.67 GETVAR

NAME:

GETVAR

FUNCTION:

Returns numeric information based on the users records for the line that you are locked into.

EXAMPLE:

```
'GETVAR' ACCESSLEVEL_VAR
'MSG' "Your access level is " result

'GETVAR' CALLSMADDE_VAR
'MSG' "Your have made " result " calls to the system"
```

RETURNS:

Returns various Interger information based on the users information statistics.

SEE-ALSO:

~_VAR~Defines~

1.68 Contacting Technical Support

Before contacting Technical Support, you should have the following:

- Your registration number
 - Your version number
 - A complete and succinct problem description
 - A list of which compiler and linker options you are using
 - A list of which parts of the product that you have problems with
-

To contact Technical Support, call:

In the UK, Manchester, (+44) (0)161-799-4922 (BBS)

In the UK, Scotland, (+44) (0)1501-44262 (BBS)

NetMail 2:250/120, 2:259/13

ApT-BBS Support Echo.

1.69 DEVELOPERS_GUIDELINES

Developers are required to pass on all script files that are "released", both as Shareware and Public Domain to ApT-Design. This allows the gathering of a actual catalogue of all software that will be released, postings made known to various support systems, all resulting in users hearing about new developed products, and updated products.

All software that is not firstly shown to ApT-Design before release will not get the so called, "ApT-BBS Seal of Approval, failing also to be included within full catalogue list of external scripts/modules/utilities.

The Classic~Defines will serve as to aiding you on sub-commands that may be accssed by a root command. It is the choice of the programmer to either use a full "defined" (english viewed define) or the actual integral variable of the command, as within this instance:

[Example 1]

```
BYTESLEFT_VAR               = 9
```

```
'GETVAR' BYTESLEFT_VAR  
'msg' "You have: "result "bytes left."
```

[Example 2]

```
'GETVAR' 9  
'msg' "You have: "result "bytes left."
```

Both of the above GETVAR commands are exactly the same. The only difference being that one of the commands uses the outlined define for the option in question (this being placed within the head part of the script), while the latter does not, and as a result does not require any defined information to be placed within the header of the script.

1.70 SETSTR

NAME:

SETSTR

FUNCTION:

Stores the required string into the users information folder. This allows for a easy method of storing STRING information within a users folder without needing to be bothered with the actual pointer to the memory contained folder.

EXAMPLE:

```
'SETSTR' HANDLE_STR "Fish Finger"
```

Has the effect of storing the supplied handle within the users folder.

SEE-ALSO:

~_STR~Defines~

1.71 SETVAR

NAME:

SETVAR

FUNCTION:

Stores the required Variable into the users information folder. This allows for a easy method of storing NUMERIC information within a users folder, without needing to be bothered with the actual pointer to the memory contained folder.

EXAMPLE:

```
'SETVAR' CALLSMAD_VAR 201
```

Has the effect of storing the 201 integer variable within the users folder pertaining to the number of calls made.

SEE-ALSO:

~_VAR~Defines~

1.72 USERFLAG

NAME:

```
'USERFLAG' <Arguments>
```

TEMPLATE:

```
"TEST/K/N, SET/K/N, CLR/K/N"
```

FUNCTION:

Allows you to retrieve and store Flags in one of 512 flag variables. The first thing to note about this command is that the 512 flags are stored within each of the users data files. They should be thought of as simply ON and OFF flags, they can not and do not contain numerical numbers, either they are ON or OFF, TRUE or FALSE.

By providing easy access to your "own" flags, you are able to do many and varied different things depending on the result of various flags.

As a example, you may wish to split your login sequence into various chunks - You would then ask a user to configure his login procedure (if a certain flag was set to 0 (off)) and would then proceed to ask various questions as to what should and should not appear at login. The next time they logs on, the required flags would result in parts of the login sequence being displayed or ignored. - There are many many many different uses for this command, all of which are controlled and decided by you, rather than ApT itself.

EXAMPLE:

```
'USERFLAG' "TEST 50" ; configured_login = result

/*
** If the user has NOT configured his login sequence
** EVER then we "at least once" present him with the
** chance - When we have finished, no matter what the
** outcome maybe, we set flag 50 so that he is never
** asked the question again, unless he asks for the
** option to present itself as part of a menu command.
**/
if(configured_login=0) then do

    'SHOWFILE' "info/dis_info"
    'msg' "\n\nDo you want the above information file displayed on login
[Y/n] =>"
    call YESNO
    if(result) THEN 'USERFLAG' "SET 52"

    'SHOWFILE' "info/display_news"
    'msg' "\n\nDo you want the above news file displayed on login [Y/n]
=>"
    call YESNO
    if(result) THEN 'USERFLAG' "SET 53"

    'SHOWFILE' "info/display_info"
    'msg' "\n\nDo you want the above status screen displayed on login
[Y/n] =>"
    call YESNO
    if(result) THEN 'USERFLAG' "SET 54"

end

/*
```

```

** Check to see if the 'configured_login' flag is ON
** We know that the user has configured it if so.
**/
if(configured_login~=0)

    /*
    ** Gather all of the required "login" result
    ** flags.
    **/
    'USERFLAG' "TEST 51" ; display_none = result
    'USERFLAG' "TEST 52" ; display_info = result
    'USERFLAG' "TEST 53" ; display_news = result
    'USERFLAG' "TEST 54" ; display_stats = result

    /*
    ** if the display_none is ON then we start
    ** displaying the login intro-sequence,
    ** otherwise we ignore the lot and drop
    ** to the main menu, without any display
    ** of the login sequence.
    **/
    if(display_none~=0) then do

        if(display_info~=0) then 'SHOWFILE' "info/dis_info"
        if(display_news~=0) then 'SHOWFILE' "info/dis_news"
        if(display_stats~=0) then 'SHOWFILE' "info/dis_stats"

    end

end

```

NOTES:

This command should not be used without "care and thought" - The first thing that you should do is try to split the 512 flags that are on offer up into many different sub-sections (or banks) - In order to do this (as a pure example) you could think of them as:

512 flags / 10 flags-per-bank = 51 banks.

You now have 51 banks each containing 10 flags per bank, which should be enough for controlling most aspects of a function. You should always make sure that you are thinking for the future as a time may come when you wish to add more flags but find that there is no room left at the position you would have liked to store the required flag, thus it has to be moved to a totally different position - this all leads to untidy and very often un-manigable flags.

Do NOT store everything in sequential 1,2,3... 10,15,30,..40 order. But try to split items into banks, and make sure enough room exists for future expansion for the functions you are providing.

If you are programming by using the method of having "banks" then it is wise to keep the very first flag of each of the new banks as SPECIAL flags that may have a dramatic effect on the other flags

for that bank. As in the example above you will note that if flag 50 (the start of the bank for the login sequence) was set, then all the following flags for that bank would be ignored.

It is recommended that flags 0-20 are kept for the most important flag-keeping, and others used for the sundry flags.

THINK ABOUT IT ALL VERY VERY CAREFULLY - IT IS TO YOUR ADVANTAGE!

RETURNS:

Returns either 1 or 0 (TRUE/FALSE) for "TEST <flagnumber>"

SEE-ALSO:

1.73 GFXFLAG

NAME:

GFXFLAG

TEMPLATE:

"TEST/K/N, SET/K/N, CLR/K/N"

FUNCTION:

Allows the Setting, testing and clearing of the users graphical flags.

```
'SET_GFX_FLAG' ANSI_COL_FLG /* Turn on ANSI master flag */
'SET_POS_FLAG' ANSI_POS_FLG /* Turn on ANSI POSITIONING codes */
```

EXAMPLE:

```
'GFXFLAG' "SET "ANSI_COL_FLG /* Turn ON the users ANSI COLOUR flag */
'GFXFLAG' "CLR "ANSI_POS_FLG /* Turn OFF the users ANSI positioning flag
*/
'GFXFLAG' "TEST "ANSI_FLG_FLG

if(result) then 'msg' "You have Ansi turned on, .. good isn't it\n"
else 'msg' "No Ansi, .. why not?\n"
```

SEE ALSO:

~_FLG~Defines~~

1.74 MOVEUSER

NAME:

MOVEUSER

FUNCTION:

This is generally used along with the HASEXPIRED command.

Once a user has been found to have come up for review you would then have his account moved. This SHOULD take place within the interlog.aprt or the logoff_final.aprt, doing this at other times within the normal running of the BBS will result in data being written to the wrong directory etc..

SEE-ALSO:

~HASEXPIRED~

1.75 DELETEACCOUNT

NAME:

DELETEACCOUNT

FUNCTION:

This is rather usefull in that you are allowed to delete any users account by passing his username along. Generally this has many uses that are beyond the scope that I am going to give here, but one handy one would be something in english terms of..

```
[interlogoff script]
```

```
Check to see if the user has called 0 times
if TRUE then check to see if he has left 0 mail
  if TRUE then WARN him he is about to be deleted
  give option of <paging/feedback/etc>
  if not accepted then delete account
```

which is handy if you are the kind of person who wants every user to leave feedback mentioning something about themselves, or failing that they are obviously not bothered about your system so why should you bother keeping a account...

EXAMPLE:

```
'DELETEACCOUNT' "STEVE STARKEY"
if(result=1) THEN 'MSG' "Steve Starkey has been deleted...."
```

If you wanted to delete the user that is currently online (within the logoff_final.aprt or interlog.aprt ONLY!

Now follows a very basic example of how you might use the command within your 'logoff_final.aprt' file. The script below will delete a account IF he/she has called 0 times (first time caller) and IF

he/she has not left any mail prior to the 'logoff' menu command. If you are not of the harsh SysOp kind then you could simply send out a msg/textfile saying there account will be purged at midnight if they do not re-log and leave feedback.

```
'GETVAR' CALLSMADe_VAR ; callsmade = result /* how many calls made */
'GETVAR' MSGSLEFT_VAR  ; msgslft = result  /* How many msgs left  */
'GETSTR' USERNAME_STR  ; username = result /* username           */

/*
** Now see if they have called 0 times before (thus a new user)
** and have left 0 msgs (so no feedback), if so then we obviously
** are going to delete the person.
**/
if(callsmade=0 & msgslft=0) THEN DO
  'DELETEACCOUNT' username
  'MSG' "\b1\c1\n\nYour account has been PURGED!!!!!!"\n\n\n"
END
```

RETURNS:

Returns 0 for FAILED or 1 for DELETED

1.76 ALERTLOG

NAME:

ALERTLOG

FUNCTION:

This will write out a line of text to the 'users' own directory under the name of Alert.Log. It will add the 'date and time' that the command was issue (in normal log-file style). Whenever the user logs onto the system the file 'Alert.log' should be searched for and displayed via other arexx commands. A Example of how this would be done is supplied and is included within the 'interlog.apt' script.

EXAMPLE:

```
'ALERTLOG' "Dropped carrier - There is a fine for this offence!"

/* Other actions here and removing of credits.... */

'ALERTLOG' "10K of bytes removed as a penalty for carrier drop."
```

Would save out the following text to the users account under Alert.Log

```
05-Mar-93 10:08:02 Dropped carrier - There is a fine for this offence!
05-Mar-93 10:08:02 10K of bytes removed as a penalty for carrier drop.
```

The file should then be displayed, by your program, the next time the user logs into the system - after displaying the file should

then be deleted.

1.77 GETUSERDIR

NAME:

GETUSERDIR

FUNCTION:

This option requires a username to be supplied as its argument, it will then return back the actual full path to the users directory. Here it would be possible for you to store other optional files depending on what kind of program you are writing etc..

EXAMPLE:

```
'GETSTR' USERNAME_STR    ; UserName  = result
'GETUSERDIR' UserName    ; UserDir   = result
'MSG' "Your directory path is: " UserDir"\n"
```

The above would first find out the name of the user that is currently online. It would then go on to request the user-directory path and return back (if the username was "JOE BLOGGS") :

Your directory path is: ApT:Users/JOE_BLOGS/

1.78 REMOVEFILE

NAME:

REMOVEFILE

FUNCTION:

This allows for a easy way of removing a file from the system. Generally it is used for removing such things as unwanted display files or logfiles etc..

EXAMPLE:

```
'REMOVEFILE' "t:fudge.display"
```

1.79 RETURNCODE

NAME:

RETURNCODE

FUNCTION:

This command is useful when ApTline calls a aptrexx script and requires a return value to make to be returned in order for a decision to be made. This command allows such a aptrexx script, that is called by aptline, to actually make the choice.

upon return from the script to aptline, aptline would look at the value, depending on what is required and "take action" upon the result.

EXAMPLE:

```
'RETURNCODE' 0
```

would pass a return code of 0 back to aptline when it called the current script.

You should read the various script programs for more information on when this should be used and what values pertain to what actions.

1.80 USEACCOUNT

NAME:

USEACCOUNT

FUNCTION:

This is used along with a username, and once the name has been given the account pertaining to that name will then be loaded into (and thus overwriting within memory) the current user information file. This is generally used inside of the logon.apt script, however it can be used anytime even if a user is already online.

EXAMPLE:

If "Steve Starkey" is online and you wish him to take on the identity of a young, but chubby Tom Snow, you would:

```
'USEACCOUNT' "Tom Snow"
```

which would overwrite the account of "Steve Starkeys" user folder contained within memory, thus the identity of the person has now changed for the duration of the call.

RETURNS:

0 for failure, otherwise success.

1.81 SMALLCMD

NAME:

SMALLCMD

TEMPLATE:

"SETUPDLG/S,REMOVEDLG/S"

FUNCTION:

Commands that are not deemed worthy of the luxury of having a actual "full callable command" are placed within this command tree, the less frequently used commands are also placed here.

EXAMPLE:

```
/*
** Placed within the starting portion of the interlog.apl script
** would result in a user who has just logged on having a special
** converted neuron account setup which will result in the ability
** to run numerous DLG based doors.
**/
'SMALLCMD' "SETUPDLG"

/*
** During logoff time it is required for the neuron account to
** be removed from tempory storage. This command does just
** that.
**/
'SMALLCMD' "REMOVEDLG"
```

1.82 CLASSICDEFS

ApT-Rexx Classic Defines Index

All of the following Defines should be used along with the commands that they relate to. The reason why this is partly recommended is because the source code, as starters, looks far more understandable and keeps up with the ApT-Rexx guidelines that are set out.

It is only required to include the defines that relate to commands that you are going to use within your source. This keeps the header size down to the smallest possible over having the entire batch of defines included within each and every source code. Use this section as a reference when creating your rexx programs and simply copy the defines over to your source.

```
~_VAR~~~ ... Define for SETVAR & GETVAR
~_STR~~~ ... Define for SETSTR & GETSTR
~_STVAR~ ... Define for SETSTATICVAR & GETSTATICVAR
~_STSTR~ ... Define for SETSTATICSTR & GETSTATICSTR
~_CALL~~ ... Define for CALLFUNCTION
```

~F-KEYS~ ... Defines for Function Keys
 ~CURSOR~ ... Defines for Cursor Keys

1.83 _VAR

ApT-Rexx Classic Defines for _VAR

ACCESSLEVEL_VAR	= 0	VALID_VAR	= 1	CALLSMAD_VAR	= 2
MSGSLLEFT_VAR	= 3	MSGSLREC_VAR	= 4	UPLOADS_VAR	= 5
DNLOADS_VAR	= 6	BYTESUP_VAR	= 7	BYTESDN_VAR	= 8
BYTESLEFT_VAR	= 9	FILERATIO_VAR	= 10	BONUSRATIO_VAR	= 11
HOTKEY_VAR	= 12	EDITOR_VAR	= 13	COMPTYPE_VAR	=
14 SCRLLENGTH_VAR	= 15	SCRWIDTH_VAR	= 16	GENDER_VAR	= 17
MORE_VAR	= 18	PROTOCOL_VAR	= 19	ARCHIVER_VAR	= 20
MENUSET_VAR	= 21				

1.84 _STR

ApT-Rexx Classic Defines for _STR

USERNAME_STR	= 0	ADDR1_STR	= 1	ADDR2_STR	= 2
ADDR3_STR	= 3	POSTCODE_STR	= 4	COUNTRY_STR	= 5
VOICENO_STR	= 6	DATANO_STR	= 7	PASSWORD_STR	= 8
HANDLE_STR	= 9				

1.85 _CALL

ApT-Rexx Classic Defines for _CALL

COMPUTERTYPE_CALL	= 0	COUNTRY_CALL	= 1
BIRTHDAY_CALL	= 2	PROTOCOL_CALL	= 3
ARCHIVER_CALL	= 4	SHOW_PASSWORD_AND_NAME_CALL	= 5

1.86 _STVAR

ApT-Rexx Classic Defines for _STVAR

DAYS2REVIEW_STVAR	= 0	DAYSLASTCALL_STVAR	= 1	ANSIDETECTED_STVAR	=
2 BAUDRATE_STVAR	= 3	DOB_DAY_STVAR	= 4	DOB_MONTH_STVAR	=
5 DOB_YEAR_STVAR	= 6				

1.87 _STSTR

ApT-Rexx Classic Defines for _STSTR

LASTDATECALL_STSTR = 0 LASTTIMECALL_STSTR = 1 GETQUOTE_STSTR = 2

1.88 FKEYS

ApT-Rexx Classic Defines for Function Keys

F1 = 240 F2 = 241 F3 = 242 F4 = 243 F5 = 244 F6 = 245 F7 = 246 F8 =
247 F9 = 248 F10 = 249

1.89 CURSORDEFS

ApT-Rexx Classic Defines for Cursor Keys

CUR_UP = 252 CUR_DOWN = 253 CUR_RIGHT = 254 CUR_LEFT = 255

1.90 _FLG

ApT-Rexx Classic Defines for Graphic Flags

ANSI_COL_FLG = 0 // Ansi colour (user can accept colour codes)
ANSI_POS_FLG = 1 // Ansi Pos (user can accept positional codes)
ANSI_FLG_FLG = 2 // Ansi flag (user can accept ANSI (required for
above)) ANSI_SCR_FLG = 3 // Ansi Scroll (user wishes for partial scroll
(requires ANSI_POS)) ANSI_CUR_FLG = 4 // Ansi Cursor off/on SCRN_CLR_FLG
= 7 // Screen Clears