

APTCONTROL.HYPER

Andy Grifo

COLLABORATORS

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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	APTCONTROL.HYPER	1
1.1	MAIN	1
1.2	APTLINKLIB	1
1.3	DMSGF	1
1.4	EPSTR	2
1.5	EPDEC	2
1.6	GETYESNO	3
1.7	LIBRARY_LIST	4
1.8	APTCLASSICDEFS	4
1.9	LINKINTODOOR	5
1.10	REMOVEDOORLINK	5
1.11	DMSG	6
1.12	SHOWTPL	6
1.13	SHOWFILE	6
1.14	EASYPROMPT	7
1.15	DROPLINE	7
1.16	ISDROP	8
1.17	SETBREAK	8
1.18	HOTKEY	9
1.19	HOTCUR	10
1.20	DOIT	11
1.21	PAGESYSOP	12
1.22	EDITSTATS	12
1.23	NEWUSERFUNCTION	13
1.24	GETPASSWORD	13
1.25	HASEXPIRED	14
1.26	CHATMODE	14
1.27	FLUSHACCOUNT	15
1.28	CHAINAPT	15
1.29	RUNDOS	15

1.30	RUNEASYDOS	16
1.31	XFEROPTS	17
1.32	BUTTONOPTS	17
1.33	AUTOMSG	18
1.34	HOTOPTS	19
1.35	SCROLLBODY	21
1.36	GETCHAR	22
1.37	GETUSERINFO	23
1.38	LOCKDOOR	23
1.39	UPDATEHISTOGRAMS	24
1.40	INITFILEAREA	24
1.41	FREEFILEAREA	24
1.42	SCANFILES	25
1.43	SELECTFILEAREA	25
1.44	TOUCHINDEX	26
1.45	LOCALNEWSCAN	27
1.46	CONTINUESCAN	27
1.47	GLOBALNEWSCAN	27
1.48	REMOVEMARKED	28
1.49	UPLOAD	28
1.50	DOWNLOAD	28
1.51	LISTMARKED	28
1.52	NEXTAREA	28
1.53	PREVAREA	29
1.54	VIEWARCHIVE	29
1.55	FILEAREAINFO	29
1.56	FILTERMENU	30
1.57	FILESTYLE	30
1.58	FINDDEFAULTBASE	31
1.59	MAILCMD	32
1.60	LINEEDITOR	33
1.61	RUNPARADOOR	33
1.62	UNFIXEDCMD	34
1.63	MARKFILE	34
1.64	SETSIG	35
1.65	GETSTATICSTR	35
1.66	GETSTATICVAR	36
1.67	GETSTR	37
1.68	GETVARIABLE	37

1.69 LIBERATECMD	38
1.70 Contacting Technical Support	39
1.71 DEVELOPERS_GUIDELINES	39
1.72 WHICH_EDITOR	40
1.73 SETSTR	40
1.74 SETVAR	41
1.75 ROBOMSG	42
1.76 FLAGCMD	42
1.77 FILESIZE	43
1.78 MOVEUSER	44
1.79 GETUSERFROMDIR	45
1.80 UNDERSCORE	45
1.81 LOADFILE	46
1.82 LOADACCOUNT	47
1.83 SAVEACCOUNT	47
1.84 MOVEFILE	48
1.85 APPLYTEMPLATE	48
1.86 INFORM	49
1.87 GETLIVECHAR	50
1.88 GETTIMEDCHAR	50
1.89 SHOWRAWFILE	51
1.90 HASACCESS	51
1.91 SMALLED	52
1.92 CENTRESTRING	53

Chapter 1

APTCONTROL.HYPER

1.1 MAIN

ApT-BBS Control.library Guide

Copyright (c) 1995 ApT-Design

```
~Developers~Guidelines~::~::~::~::~
~Which~Editor?~--aid~on~picking~a~text~editor~::
~Library~Comamnds~::~::~::~::~
~ApTLib.lib~link~time~library~commands~::
~A~word~about~the~ApTClassicDefs.h~file~::
~Contacting~Technical~Support~::
```

1.2 APTLINKLIB

ApT-BBS ApTLib.lib List

This is a link-time library - That is, it is not a actual .library but rather a collection of useful functions that should normally be linked with all of your projects, thus giving access to functions that will make developing easier than without. As a small example, the DMsgf command is just the same as the standard printf() command but allows passing of formatted arguments which could not be done 'easily' with the standard DMsg() .library command.

```
~~DMsgf(str,args)~~~~~ .... Formatted printing, like printf()
~~EPstr(prompt,args)~~~ .... EasyPrompt, string related
~~EPdec(decimal,args)~~ .... EasyPrompt, decimal related
~~GetYesNo(flag)~~~~~ .... Yes/no calls made easier
```

1.3 DMSGF

NAME:

```
DMsgf(string,arguments)
```

FUNCTION:

Allows the displaying of text with the additional power of using formatting codes much the same way as the printf() command works.

EXAMPLE:

```
WORD xval=20,yval=15;
```

```
DMsgf("The X value=%ld and the Y value=%ld\n",xval,yval);
```

```
"The X value=20 and the Y value=15"
```

SEE-ALSO:

```
DMsg()
```

1.4 EPSTR

NAME:

```
EPstr(prompt,args)
```

FUNCTION:

A form of EasyPrompt'ing made even easier, but mainly for uses when only a string is required to be returned back.

You pass a string which is to act as the default "string" which then may be changed by the user. Upon accepting the new string by the pressing of return, the string is then returned back.

You also pass arguments which contain the remainder of the initiating required to make EasyPrompt filter the required items, as well as setup the length of the user-editable prompt.

EXAMPLE:

```
DMsg("\n\nEnter the handle that you wish to use on this system.\n:");  
DotIt(14);  
strcpy(udat->us_alias,EPstr(udat->us_alias,"LEN=30 LETTER NUMBER"));
```

SEE-ALSO:

```
EPdec() EasyPrompt()
```

1.5 EPDEC

NAME:

EPdec(decimal, args)

FUNCTION:

A form of EasyPrompt'ing made even easier, but mainly for uses when only a decimal/integer value is required to be returned back.

You pass the decimal function which is to act as the initial outputted value which then may be changed by the user. Once the user has finished with the value the result is returned back in decimal.

You also pass arguments which contain the remainder of the initiating required to make EasyPrompt filter the required items, as well as setup the length of the user-editable prompt.

EXAMPLE:

```
udat->us_sclen = (UBYTE)EPdec(udat->us_sclen, "LEN=2 NUMBER");
```

SEE-ALSO:

EPstr() EasyPrompt()

1.6 GETYESNO

NAME:

GetYesNo(flag)

FUNCTION:

It is often required to get a response prompt from the user in the form of a Yes or No. It is also sometimes required to have the return key act as a 'default' to either call Yes or No depending on what is required - The function also displays the text "Yes" or "No" leaving you to bother about the return value only.

EXAMPLE:

```
/*
** In this example we default to NO (false being the call flag)
**/
DMsgf("\n\nDo you wish to search for files since your last call? [y/N]
=>");

if(GetYesNo(FALSE))
{
    ... user selected YES
}
else
```

```

{
    ... user selected NO
}

/*
** Here we default to YES
**/
DMsg("\n\nYou have waiting mail!! view Mail Index now? [Y/n] =>");

if (GetYesNo(TRUE))
    MailCmd("SHOWMAIL");

```

RETURNS:

TRUE for YES , FALSE for NO

1.7 LIBRARY_LIST

ApT-BBS ApTControl.library List

```

~~AutoMsg()~~~~~~~ ~~ApplyTemplate()~~~~ ~~ButtonOpts()~~~~~~
~~CentreString()~~~~~ ~~ContinueScan()~~~~~ ~~ChatMode()~~~~~~~
~~ChainApT()~~~~~~~ ~~DMsg(String)~~~~~~~ ~~Download()~~~~~~~
~~DotIt(var)~~~~~~~ ~~DropLine()~~~~~~~ ~~EasyPrompt()~~~~~~~
~~EditStats()~~~~~~~ ~~FreeFileArea()~~~~~ ~~FlagCmd()~~~~~~~
~~FlushAccount()~~~~~ ~~FileAreaInfo()~~~~~ ~~FileSize()~~~~~~~
~~FilterMenu()~~~~~~~ ~~FileStyle()~~~~~~~ ~~FindDefaultBase()~~
~~GetChar()~~~~~~~ ~~GetLiveChar()~~~~~~~ ~~GetTimedChar()~~~~~
~~GetPassword()~~~~~ ~~GetUserInfo()~~~~~~~ ~~GetStaticStr()~~~~~
~~GetStaticVar()~~~~~ ~~GetStr()~~~~~~~ ~~GetUserFromDir()~~~
~~GetVariable()~~~~~ ~~GlobalNewScan()~~~~~ ~~HasAccess()~~~~~~~
~~Hotkey()~~~~~~~ ~~HotCur()~~~~~~~ ~~HotOpts()~~~~~~~
~~Inform()~~~~~~~ ~~InitFileArea()~~~~~ ~~IsDrop()~~~~~~~
~~LinkIntoDoor()~~~~~ ~~ListMarked()~~~~~~~ ~~LineEditor()~~~~~
~~LiberateCmd()~~~~~ ~~LoadAccount()~~~~~~~ ~~LoadFile()~~~~~~~
~~LockDoor()~~~~~~~ ~~LocalNewScan()~~~~~ ~~MailCmd()~~~~~~~
~~MarkFile()~~~~~~~ ~~MoveFile()~~~~~~~ ~~MoveUser()~~~~~~~
~~NewUserFunction()~~ ~~NextArea()~~~~~~~ ~~PageSysop()~~~~~~~
~~PrevArea()~~~~~~~ ~~RemoveDoorLink()~~~ ~~RemoveMarked()~~~~~
~~RoboMsg()~~~~~~~ ~~RunDos()~~~~~~~ ~~RunEasyDos()~~~~~~~
~~RunParaDoor()~~~~~ ~~SaveAccount()~~~~~~~ ~~ScrollBody()~~~~~
~~ScanFiles()~~~~~~~ ~~SelectFileArea()~~~ ~~SetSig()~~~~~~~
~~SetStr()~~~~~~~ ~~SetVariable()~~~~~~~ ~~SetBreak()~~~~~~~
~~ShowTpl()~~~~~~~ ~~ShowFile()~~~~~~~ ~~ShowRawFile()~~~~~
~~Smalled()~~~~~~~ ~~TouchIndex()~~~~~~~ ~~UpdateHistograms()~
~~Underscore()~~~~~ ~~UnfixedCmd()~~~~~~~ ~~Upload()~~~~~~~
~~ViewArchive()~~~~~ ~~XferOpts()~~~~~~~

```

1.8 APTCLASSICDEFS

The ApTClassicDefs.H file contains all of the defines that will serve the following commands:

```
~GetStaticStr(strnumber)~~~ _STSTR defines applicable.  
~GetStaticVar(varnumber)~~~ _STVAR defines applicable.  
~GetStr(strnumber)~~~~~ _STR defines applicable.  
~GetVariable(varnumber)~~~ _VAR defines applicable.  
~SetStr(varnum,string)~~~~ _STR defines applicable.  
~SetVariable(varnum,val)~~~ _VAR defines applicable.  
~NewUserFunction(Function)~ _CALL defines applicable.
```

1.9 LINKINTODOOR

NAME:

```
LinkIntoDoor(linenum) (d0)
```

FUNCTION:

After opening the ApTControl.library it is required link into the ApT-line that you wish to control from external sources (such as a module/door) - This command does just this. Once locked into a door you are then able to use the various functions within the library.

EXAMPLE:

```
if(LinkIntoDoor(LineNumber))  
{  
    DoThing();  
  
    RemoveDoorLink();  
}
```

RETURNS:

BOOL - FALSE for failure.

SEE-ALSO:

```
RemoveDoorLink()
```

1.10 REMOVEDOORLINK

NAME:

```
RemoveDoorLink()
```

FUNCTION:

Closes down your communication between your controlling program and ApT, thus rendering ApT to once again resume normal service.

EXAMPLE:

```
RemoveDoorLink();
```

SEE-ALSO:

```
LinkIntoDoor()
```

1.11 DMSG

NAME:

```
DMsg(String) (a0)
```

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:

```
DMsg("Hello world,.. Press a key %WAITKEY.");
```

1.12 SHOWTPL

NAME:

```
ShowTpl(TemplateName) (a0)
```

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.  
**/  
ShowTpl("menus/mymenu");
```

RETURNS:

1.13 SHOWFILE

NAME:

```
ShowFile(FileName) (a0)
```

FUNCTION:

Same as the ShowTpl(..) 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.
**/
ShowFile("ram:wishingwell.txt");
```

1.14 EASYPROMPT

NAME:

EasyPrompt (Arguments) (a0)

TEMPLATE:

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

FUNCTION:

Allows EASY prompting, ..

EXAMPLE:

```
/*
** ask the user for a string no more than 60 characters
** in length, storing the result in the 'input' buffer
** upon pressing of return. Does not store the actual
** return code itself, just the pure string.
**/
UBYTE input[64];

DMsg("\nEnter something> ");
strcpy(input, EasyPrompt("STRING=\"\" LEN=60 ASCII"));
```

RETURNS:

Pointer to string..

1.15 DROPLINE

NAME:

DropLine()

FUNCTION:

Drops the line,.

EXAMPLE:

```
if (BadBadUser)
{
    DMsg("Your outta here..\nSeeya..\n");
    DropLine();
}
```

1.16 ISDROP

NAME:

IsDrop()

FUNCTION:

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

EXAMPLE:

```
if (IsDrop())
{
    /*
    ** The user has vanished..
    **/

    .
    .
}
else
{
    DMsg("Phew, .. you still exist!\n");
}
```

RETURNS:

BOOL - TRUE denotes line has dropped.

1.17 SETBREAK

NAME:

SetBreak(KeyTokens) (a0)

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 );
```

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

```
SetBreak( 1 );
```

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

1.18 HOTKEY

NAME:

```
Hotkey(String) (a0)
```

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 command to be changed.

EXAMPLE:

```
UBYTE Character;
```

```
Character = HotKey( "Press a key =>" );
```

RETURNS:

```
UBYTE Character.
```

SEE ALSO:

See HotOpts() for more information on control of this command.

1.19 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 directon.

? 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:

```

#define CUR_UP      252  /* the rawkey defines */
#define CUR_DOWN    253
#define CUR_RIGHT   254
#define CUR_LEFT    255

UBYTE rawkey;

rawkey = HotCur("")

switch( rawkey )
{
    case CUR_UP:
        DMsg("You pressed Cursor up");
        break;
    case CUR_DOWN:
        DMsg("You pressed Cursor Down");
        break;
    case CUR_RIGHT:
        DMsg("You pressed Cursor Right");
        break;
    case CUR_LEFT:
        DMsg("You pressed Cursor Left");
        break;
    default:
        break;
}

```

RETURNS:

Exactly the same as the Hotkey(..) command but Cursor keys are returned as SINGLE characters of 0xFC 0xFD 0xFE 0xFF (Hex, or use above defines or even 252,253,254,255 Decimal)

SEE-ALSO:

Hotkey() ~link~HOTKEY}~, ~@{

1.20 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:

```
DMsg("Enter your name =>");
DotIt(18);

/* Prompt calls etc.. */

.

.

/* results in (minus speechmarks) */

"Enter your name =>                ."
```

1.21 PAGESYSOP

NAME:

```
PageSysop()
```

FUNCTION:

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

EXAMPLE:

```
switch(Character)
{
    case 'C':
        DMsg("Calling for the System operator...");
        PageSysOp();
        break;
    case 'F':
        .
        .
}
```

1.22 EDITSTATS

NAME:

```
EditStats(StatsFunction) (d0)
```

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.. */
```

NOTES:

Changes have been made to ApT in such a way that it will now call an external "arexx script" - Modules will be created which you can easily call via the LOCKDOOR options.. However, the rexx versions should suffice.

1.23 NEWUSERFUNCTION

NAME:

NewUserFunction(Function) (d0)

FUNCTION:

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

EXAMPLE:

```
NewUserFunction( functionnumber );
```

SEE-ALSO:

~ApTClassic~Defines~

1.24 GETPASSWORD

NAME:

GetPassword(Length) (d0)

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:

```
RawText( "Enter your password to enter private area =>" );  
strcpy(PBuffer,GetPassword(8));
```

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

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

RETURNS:

Password string entered.

1.25 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:

TRUE or FALSE depending if he has come up for review or not (TRUE denoting user has come up for review)

1.26 CHATMODE

NAME:

ChatMode()

FUNCTION:

Enters chatmode, directly.

EXAMPLE:

```
.  
.
default:
    switch(RawChar)
    {
        case 240:        /* Function key 1 pressed (F1) */
            ChatMode();
            break;
```

```
        default:
            break;
    }
    break;
}
```

1.27 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.

1.28 CHAINAPT

NAME:

```
ChainApT(FileName) (a0)
```

FUNCTION:

Loads in a ApTREXX file and uses for the duration until it returns back to the above call. Simply pass the filename of the file minus the aptrexx: and .apt additions (which are auto-added)

EXAMPLE:

```
/*
** Loads and runs "ApTRexx:WishingWell.tpl";
**/
ChainApT("WishingWell");

/*
** Loads and runs "ApTRexx:Daleks/Leader.tpl";
**/
ChainApT("Daleks/Leader");
```

RETURNS:

TRUE for success, FALSE for failure.

1.29 RUNDOS

NAME:

RunDos (FileName) (a0)

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()

1.30 RUNEASYDOS

NAME:

RunEasyDos (FileName) (a0)

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 ()

1.31 XFEROPTS

NAME:

XferOpts (Arguments) (a0)

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.
SENDFILES - 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 'SENDFILES' 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 ("SENDFILES UPDATERATIO");
```

RETURNS:

1.32 BUTTONOPTS

NAME:

ButtonOpts (Arguments) (a0)

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.

1.33 AUTOMSG

NAME:

AutoMsg(Arguments) (a0)

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, F=File/A/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 subject 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.

EXAMPLE:

```
AutoMsg("ram:body.txt area=1 to=\"Andy Grifo\" from=\"Dave McIntosh\"  
subject=\"Life..\"");
```

RETURNS:

1.34 HOTOPTS

NAME:

HotOpts(Arguments) (a0)

TEMPLATE:

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

FUNCTION:

This function setups 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.
**/
typer = HotOpts("SYSOPKEY");
if(typer) DMsg("Console -> SysOp Hit a key, wow!");
else      DMsg("Terminal -> User Hit a key, wow!");
```

RETURNS

Depending on command, who pressed the last key.

SEE ALSO:

Hotkey()

1.35 SCROLLBODY

NAME:

ScrollBody(Arguments) (a0)

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.
**/
DMsg("%CLS.%CURSOR=0.Header1\nHeader2\n");
DMsg("\c1AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA\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");

DMsg("%CURSOR=1.");
```

NOTES:

It is VERY IMPORTANT that whenever this command is used that 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.

1.36 GETCHAR

NAME:

GetChar()

FUNCTION:

Allows the returning of the next character typed, unaltered.

EXAMPLE:

```
UBYTE RawChar,UpperChar;

RawChar  = GetChar();
UpperChar = toupper(RawChar);

switch(UpperChar)
{
    case 'C':
        PageSysop();
        break;
    case 'F':
        FilesMenu();
        break;
    default:
        switch(RawChar)
        {
            case 240:
                DMsg("\nYou pressed F1\n");
                break;
            case 241:
                DMsg("\nYou pressed F2\n");
                break;
            default:
                break;
        }
        break;
}
```

RETURNS:

The next character typed.

1.37 GETUSERINFO

NAME:

```
GetUserInfo()
```

FUNCTION:

It is often required to tap into the actual "users" data file, so that one has total control over what is both read and entered into it. By using this command it allows you to do just that, returning a pointer to the 'userdata'.

EXAMPLE:

```
UBYTE Buffer[80];

struct UserInfo *ui=NULL;

ui = (struct UserInfo *)GetUserInfo();

sprintf(Buffer,"Your name is:%s\n Your handle is:%s\n",
        ui->us_name,ui->us_alias);

DMsg(Buffer);
```

NOTES:

Be very careful what you put into the structure as size differences do make a difference and you may find that you are overwriting important items of the users file.

RETURNS:

A pointer to the UserInfo structure pertaining to the line that you are currently locked into.

1.38 LOCKDOOR

NAME:

```
LockDoor(argumets) (a0)
```

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:

```
UpdateHistograms()
```

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 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 file-routine. This frees various resources back to the system.

EXAMPLE:

See the various included sources..

RETURNS:

1.42 SCANFILES

NAME:

ScanFiles()

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 SELECTFILEAREA

NAME:

SelectFileArea()

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 ~0 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(index) (d0)

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 SelectFileArea() command, or have a area you wish to 'force' touch, then this is the command that needs to be used.

EXAMPLE:

```
ULONG selected_area;

selected_area = SelectFileArea();

if(selected_area !=~0)
{
    TouchIndex(selected_area);
}
```

another example would be,

```
if(TouchIndex(12))
    DMsg("Successful touch of index 12..");
else
    DMsg("Failure, ..");
```

RETURNS:

BOOL, FALSE 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 maxmimum 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(args) (a0)

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:

```
totalfiles = FileAreaInfo("TOTALFILES");
```

```
areanumber = FileAreaInfo("AREANUMBER");  
position   = FileAreaInfo("FILEPOS");  
credits    = FileAreaInfo("CREDITS");  
filesmarked= FileAreaInfo("FILESMARKED");
```

RETURNS:

Interger items, ..

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(cmd) (d0)

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.

EXAMPLE:

```
FilterMenu(TRUE); /* Enter the Filters menu */
```

```
FilterMenu(FALSE); /* Clear the Filters, turning them off */
```

1.57 FILESTYLE

NAME:

FileStyle(args) (a0)

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");
```

1.58 FINDDEFAULTBASE

NAME:

FindDefaultBase(args) (a0)

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:

```
first_base_user_has_access_to = FindDefaultBase("START");  
last_base_user_has_access_to  = FindDefaultBase("END");
```

RETURNS:

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

1.59 MAILCMD

NAME:

```
MailCmd(args) (a0)
```

TEMPLATE:

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

EXAMPLE:

```
MailCmd("FEEDBACK"); /* Force user into leaving feedback */  
  
/*  
** Check if user has mail  
**/  
if(MailCmd("HASMAIL"))  
{  
    /*  
    ** User has mail  
    **/  
    MailCmd("SHOWMAIL");  
}  
  
MailCmd("LIST 1"); /* Displays Msg area selector, with SIG 1 areas */
```

RETURNS:

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

1.60 LINEEDITOR

NAME:

```
LineEditor(quote) (a0)
```

FUNCTION:

Calls the LineEditor - By supplying a filename pointing to a quote file will result in the use of a quote within the editor if the user so wishes.

EXAMPLE:

```
UBYTE *MsgBuffer=NULL;

MsgBuffer = LineEditor("");

if (MsgBuffer)
{
    DMsg("This is what you typed in. [Press a key]%WAITKEY.");

    DMsg(MsgBuffer);

    /*
    ** This is REQUIRED to actually free the msg buffer contents.
    ** It should only be called once you have finished with the
    ** contents of the message.
    **/
    FreeVec(MsgBuffer);
}
else
{
    DMsg("Didn't bother to write a msg eh..\n");
}
```

RETURNS:

Pointer to buffer used to hold the message contents. Must have a call to FreeVec(Buffer); to free the contents and memory.

1.61 RUNPARADOOR

NAME:

```
RunParaDoor(args) (a0)
```

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:

```
RunParaDoor("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 UNFIXEDCMD

NAME:

```
UnfixedCmd(args) (a0)
```

FUNCTION:

Sometimes test routines are placed here before they are actually supported with a real command. Generally used for Beta testers of commands, so do not be alarmed by the lack of information here.

1.63 MARKFILE

NAME:

```
MarkFile(args) (a0)
```

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.
```

```
    **/  
    got_any_files = MarkFile("QUERY");  
    if(got_any_files) DMsg("We have some files for download Sir..");  
    else              DMsg("We have no files for download, don't you  
know..");  
  
    /*  
    ** Clears any marked files from the list.  
    **/  
    MarkFile("CLEAR");
```

SEE ALSO:

Download()

1.64 SETSIG

NAME:

SetSig(args) (a0)

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.  
    **/  
    old_sig = SetSig("NEWSIG=2");  
  
    /*  
    ** Find out what the 'current' sig value is  
    ** without actually changing the sig value.  
    **/  
    current_sig = SetSig("QUERY");
```

1.65 GETSTATICSTR

NAME:

GetStaticStr(strnumber) (d0)

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:

```
#define LASTDATECALL_STSTR 0

strcpy(last_date,GetStaticStr(LASTDATECALL_STSTR));
sprintf(buffer,"last date called: %s\n",last_date);
DMsg(buffer);
```

RETURNS:

Various strings depending on the command number given.

SEE ALSO:

~ApTClassic~Defines~
_STSTR defines.

1.66 GETSTATICVAR

NAME:

GetStaticVar(varnumber) (d0)

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:

```
#define DAYS2REVIEW_STVAR      0

days2review = GetStaticStr(DAYS2REVIEW_STVAR);
sprintf(buffer,"Your account comes up for review in %ld
days.\n",days2review);
DMsg(buffer);
```

RETURNS:

Various various, such as the birth date, days to review, baud rate,etc.

SEE-ALSO:

```
~ApTClassic~Defines~  
  
_STVAR defines.
```

1.67 GETSTR

NAME:

```
GetStr(strnumber) (d0)
```

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:

```
#define USERNAME_STR 0  
#define HANDLE_STR 9  
  
UBYTE UserName[80],Handle[80];  
  
strcpy(UserName, GetStr( USERNAME_STR ));  
DMsg("Your user name is: "); DMsg(UserName); DMsg("\n");  
  
strcpy(Handle, GetStr( HANDLE_STR ));  
DMsg("Your user handle is "); DMsg(Handle); DMsg("\n");
```

RETURNS:

String information..

SEE-ALSO:

```
~ApTClassic~Defines~  
  
_STR defines.
```

1.68 GETVARIABLE

NAME:

```
GetVariable(varnumber) (d0)
```

FUNCTION:

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

EXAMPLE:

```
#define ACCESSLEVEL_VAR 0
#define CALLSMADE_VAR 2

UBYTE Level;
ULONG CallsMade;

UBYTE Buffer[80];

Level = GetVariable(ACCESSLEVEL_VAR);
CallsMade = GetVariable(CALLSMADE_VAR);

sprintf(Buffer, "Calls Made:%ld\nAccess Level:%ld\n",
        CallsMade, Level);
DMsg(Buffer);
```

RETURNS:

Returns various Integer information based on the users information statistics.

SEE-ALSO:

~ApTClassic~Defines~
_VAR defines.

1.69 LIBERATECMD

NAME:

LiberateCmd(args) (a0)

TEMPLATE:

"SETUPDLG/S,REMOVEDLG/S"

FUNCTION:

Various commands are not deemed "all important" and are kept within the confined space of the LiberateCmd(..) - depending on the arguments supplied will result in the actions of the command taking place.

EXAMPLE:

```
LiberateCmd("SETUPDLG");
```

This converts various information from ApT to that of DLG variants. This allows 'certain' DLG based doors to run that require minimum information.

```
LiberateCmd("REMOVEDLG");
```

Removes information created for the use of the DLG utilities.

Normally the SETUPDLG command is placed within your Interlogin sequence and a following REMOVEDLG at some logoff stage, either in rexx or in coded form.

RETURNS:

1.70 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

Our Internet sites: -

andy@nwnet.demon.co.uk

dave@aptsoft.compulink.co.uk

ApT-BBS Support Echo.

1.71 DEVELOPERS_GUIDELINES

Developers are required to pass on all programs 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.

The main area where we would like to see developers aim their efforts is within the Utilities area.

Anything that is released as Shareware should have any form of initial Banner removed from both the start and the end of the program in question. If Banners are required to be placed within the program, thus indicating it's origins and where support can be found, it would be wise to place this within some form of About screen that can only be accessed under the users wishes.

The above is also recommended for PD software.

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

1.72 WHICH_EDITOR

Most of the commands used to access the various library functions require "case shifting", basically meaning that command functions are very rarely lowercase. For this we recommend that a editor is used that allows for the correction of cases, generally done when the cursor leaves the line that you are writing.

As a small example:

```
if(Wishing)
    dmsg("I love Lucy..");
```

After the cursor finishes with the 'dmsg("I love Lucy..");' - the line would be scanned and have it's case changed into.

```
DMsg("I love Lucy..");
```

You are always required in such editors to enter your own dictionary commands, but it serves developers best if the time and effort is taken to do so, life will be that bit easier.

The Following list of editors all can correct cases:

Turbotext Commercial Edge Commercial GoldED
Shareware (recommended overall editor)

Popular Editor that sadly can not correct cases as of 3.05 and below.

CygnusED Pro .. Commercial.

1.73 SETSTR

NAME:

```
SetStr(varnumber,string) (d0,a0)
```

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:

```
#define HANDLE_STR    9
```

```
SetStr(HANDLE_STR, "The Liberator");
```

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

The alternative would have been something like:

```
strcpy(udat->us_alias, "The Liberator");
```

However, the latter one does NOT monitor for the size of the string and will overwrite if a string that is too large is given. The SetStr(..,..) command is far more powerful and makes quite sure that nothing out of bounds is ever stored.

For this reason it is recommended to use the SetStr and SetVariable commands.

SEE-ALSO:

~ApTClassic~Defines~

_STR defines.

1.74 SETVAR

NAME:

```
SetVariable(varnumber, value) (d0, d1)
```

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:

```
#define CALLSMADE_VAR    2
```

```
SetVariable(CALLSMADE_VAR, 201);
```

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

The alternative would have been something like:

```
udat->us_calls = 201;
```

SEE-ALSO:

~ApTClassic~Defines~

_VAR defines.

1.75 ROBOMSG

NAME:

RoboMsg(Arguments) (a0)

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/S"

FUNCTION:

Allows the posting of messages without the need to be connected to a given line , .

EXAMPLE:

```
/*
** It should be noted to "new" programmers that the laws of 'C'
** require that speechmarks within a encased " ... " require that
** they be prefixed with a \ sign, this is replaced at compile
** time with the actual " sign. So,
** UBYTE *MyString = "Hello \"Sir George\"";
** dmsg(MyString);
**
** Hello "Sir George"
**
**/
```

```
RoboMsg("B=ram:Body.txt A=1 TO=\"Andy Grifo\" F=\"Dave
McIntosh\" S=Life");
```

arguments becomes..

B=ram:Body.txt A=1 TO="Andy Grifo" F="Dave McIntosh" S=Life

and the message is posted to Area 1 etc..

1.76 FLAGCMD

NAME:

FlagCmd(flagptr,Arguments) (a0,a1)

TEMPLATE:

"TEST/K/N,SET/K/N,CLR/K/N,BYTE/S,WORD/S,LONG/S"

FUNCTION:

Allows easy manipulation of BIT's, the actual flag pointer can pointer to any size of "contained" flags.

The one thing to note is that you have to tell the command what the length of the bit "container" actually is. In the following instances listed within the example(s) the container is a ULONG, so one would denote that the container is a LONG. If, for example bits are being defined as:

```
UBYTE MyFlags[32]; //; 32 * 8 = 256 Flags
```

and all references within the FlagCmd() would have BYTE as possibly the last argument.

EXAMPLE:

```
flag_50 = FlagCmd((APTR *)udat->us_setupflags,"TEST 50 LONG");

if(flag_50)
{
    DMsg("Flag 50 is set within the us_setupflags\n");
}
else
{
    DMsg("Flag 50 is NOT set, .. within the us_setupflags\n");
}
```

or

```
ULONG MyFlags[16]; //; 16 * 32 = 512 flags (0-511)

FlagCmd((APTR *)&MyFlags,"SET 0 LONG");
FlagCmd((APTR *)&MyFlags,"SET 1 LONG");
```

RETURNS:

Returns TRUE/FALSE denoting the form that the flag spcified is set to, only, that is, if you are TESTING for it.

1.77 FILESIZE

NAME:

```
Size = FileSize(filename) (a0)
```

FUNCTION:

Quite simply returns the size of a filename that is given.

EXAMPLE:

```
if (FileSize("t:myfile"))
{
    File exists..

    .

    .

    Go on to open the file and do with it what we will..

    .

    .
}

or

ULONG fsize;

fsize = FileSize("ram:someonesfile");

DMsgf("the size of that ugly 'ram:someonesfile' is '%ld'
bytes.\n", fsize);
```

RETURNS:

Size of the file, or NULL if file does not exist.

1.78 MOVEUSER

NAME:

Success = MoveUser(accountname, kind) (a0, d0)

FUNCTION:

Moves a users account, whose name is supplied, to one of three possible directories: Users, NewUsers/ , Review/

EXAMPLE:

```
if (HasExpired())
{
    MoveUser(udat->us_name, USER_REVIEW);
    ShowTpl("review/review");
    DropLine();
}
```

NOTE:

Care should be taken when moving a user who might be online. It is generally recommended that users who are online are only moved as soon as they have entered their names and are inside the actual interlog portion of the system, to which the line should be dropped soon after.

RETURNS:

Success is TRUE.

1.79 GETUSERFROMDIR

NAME:

AccountVariable = GetUserFromDir(Accountname, DestinationString) (a0, a1)

FUNCTION:

Finds out which directory a given user is actually being contained within. Generally the string that is placed in "Destination" is the full directory as in: "ApT:Users" or "ApT:NewUsers" etc. and no '/' sign is added, this is for you yourself to do, so take note to add it if required.

EXAMPLE:

```
UBYTE buffer[80];

if(GetUserDir("Joe Bloggs",buffer))
{
    .
    .

    DMsgf("Joe Bloggs resides within:%s directory\n",buffer);
}
```

RETURNS:

The 'variable' that is returned is the directory 'value' that is defined within the includes. i.e the same form as USER_REVIEW, USER_MEMBER, etc. ~0 denotes failure, .

1.80 UNDERSCORE

NAME:

String = Underscore(String) (a0)

FUNCTION:

Quite often it is required to have a string "underscored", which means that a string such as , "Hello World, I was here", would

become, "Hello_World,_I_was_here", which comes into it's own more than anything when dealing with users account directory names etc.

It should be noted that the string you supply is the actual string that is converted, so it is not the done thing to underscore "direct" pointers to the users account name, but firstly copy them and work on a buffer.

EXAMPLE:

```
UBYTE buffer[80],todo[80];

strcpy(todo,udat->us_name);

Underscore(todo);

DMsgf("Your name, underscored, is %s\n",todo);
```

RETURNS:

The newly converted string is returned.

1.81 LOADFILE

NAME:

```
success = LoadFile(filename,buffer,filesize) (a0,a1,d0)
```

FUNCTION:

Fast and simple to use LoadFile command, takes care of everything that could be required in the opening and loading in of a files contents.

EXAMPLE:

```
LONG fsize = FileSize("t:datablock.dat");

if(fsize)
{
    UBYTE *blockbuffer=NULL;

    blockbuffer = AllocVec(fsize,MEMF_CLEAR);

    if(blockbuffer)
    {
        if(LoadFile("t:datablock.dat",blockbuffer,fsize))
        {

            .

            .

            Do ones thing..
        }
    }
}
```

```
        .  
    }  
    FreeVec(blockbuffer);  
}  
}
```

RETURNS:

Success is TRUE

1.82 LOADACCOUNT

NAME:

```
struct UserInfo * = LoadAccount(account,kind) (a0,d0)
```

FUNCTION:

Loads in a users account, passes a pointer back to contents of the users folder.

EXAMPLE:

```
/*  
** Assuming we know for sure that 'Joe Bloggs' resides within  
** the apt:users/ directory.  
**/  
  
struct UserInfo *uinfo=NULL;  
  
uinfo = LoadAccount("Joe Bloggs",USER_MEMBER);  
  
if(uinfo)  
{  
    DMsgf("actual account name is:%s\n",uinfo->us_name);  
  
    DMsgf("actual handle is      :%s\n",uinfo->us_alias);  
  
    FreeVec(uinfo);  
}
```

RETURNS:

Pointer to the users account folder - This **MUST** be FreeVec()'ed after use, otherwise memory will not be returned back to the system.

1.83 SAVEACCOUNT

NAME:

```
Success = SaveAccount(accountname,UserInfo,kind) (a0,a1,d0)
```

FUNCTION:

Saves users folder that has been contained within memory, either from the returned pointer given from the LoadAccount command, or otherwise.

EXAMPLE:

```
/*
** Assuming we know for sure that 'Joe Bloggs' resides within
** the apt:users/ and we wish to re-write him back to that
** directory, and also that we have been given a pointer from
** the LoadAccount command that pertains to his account, .
**/

.

.

.

SaveAccount("Joe Bloggs",uinfo,USER_MEMBER);
```

RETURNS:

```
Success on save is TRUE
```

1.84 MOVEFILE

NAME:

```
Success = MoveFile(src,dst) (a0,a1)
```

FUNCTION:

Moves a file from one directory to another, ..

EXAMPLE:

```
MoveFile("ram:dir1/file","ram:dir2/file");
```

RETURNS:

```
Success on move is TRUE
```

1.85 APPLYTEMPLATE

NAME:

```
Success = ApplyTemplate(TemplateName, UserInfo *ui) (a0,a1)
```

FUNCTION:

Applies a given "template transpose" file into the supplied 'UserInfo' structure.

This can be used in such instances in gaining a pointer to a user who might be online and "tansposing" his account with a fresh template. There are many many instances when this function could be used.

EXAMPLE:

```
ApplyTemplate("FullMember", udat);
```

would apply the template:

```
TPlates:UserTemplates/FullMember.TRP
```

RETURNS:

Success TRUE or FALSE

1.86 INFORM

NAME:

```
Inform(String) (a0)
```

FUNCTION:

Updates the "User is doing.." portion of the ApT-Server - This string can also be seen if "another user" requests a "Who is online" option. They would see various information along with a portion of this string (depending on the set width..) - This being the case it is sometimes worth keeping the string as brief as possible.

EXAMPLE:

```
Inform("Files"); /* User has entered the files area... */
```

```
Inform("Games"); /* User has entered the games area... */
```

```
Inform("Playing Hack & Slash");
```

RETURNS:

VOID

1.87 GETLIVECHAR

NAME:

```
GetLiveChar()
```

FUNCTION:

This works exactly the same as the `GetChar()` option but with the difference that it does not "wait" for a key to be pressed before returning. Rather, it will return no matter if a key has been pressed.

This can be good for instances when a program is "doing a task" that the user may wish to abort out of. So checking for ^C's or S (for stop) could do this task.

It's other advantage could be for games.

EXAMPLE:

```
Key = GetLiveChar();
```

RETURNS:

Character or NULL (for no character typed)

1.88 GETTIMEDCHAR

NAME:

```
GetTimedChar(Tics)
```

FUNCTION:

This option is rather like combining both the `"GetChar()"` and `"GetLiveChar()"` together. It allows you to set a "timeout" value which will result in the command waiting for a key to be pressed, but IF it is not pressed within X amounts of tics, return back to you. This allows you to do many various exciting things ..

This allows for you to, in a way, multitask your programs.

It is useful for such tasks as having (what we call) "Multitasking Menus" (menus that are updated after x amounts of inactivity from the user) as well as many other instances.

With a lot of thought this option can achieve many goals.

TICS = 50 tics are equal to 1 second.

EXAMPLE:

```
Key = GetTimedChar(50*4); /* Timeout after 4 seconds.. */
```

```
if (Key)
{
    DMsgf("\nYou pressed the '%c' key!\n",Key);
}
else
{
    DMsg("\nBaggh, you just sat there doing nothing! No keypress!!\n");
}
```

RETURNS:

Character or NULL (for no character typed)

1.89 SHOWRAWFILE

NAME:

```
ShowRawFile(filename)
```

FUNCTION:

Displays a file to the screen with NO interpreting of AML codes. This purely shows a file "as it was intended"

EXAMPLE:

```
ShowRawFile("t:things.txt");
```

1.90 HASACCESS

NAME:

```
Success = HasAccess(Groups)
```

FUNCTION:

Allows you to check if the user falls within a specific access group.

EXAMPLE:

```
if (HasAccess("5,10,15,20"))
{
    DMsg("Your level must be either 5,10,15 or 20!");
}

if (HasAccess("20-50,255"))
{
    DMsg("Your level must be in the range of 20-50 or 255!");
}

if (HasAccess("10,90-100,63,29"))
```



```
{
    DMsg("Your level must be, 10 90 to 100, 63 or 29!");
}
```

RETURNS:

Success = TRUE or FALSE

1.91 SMALLED

NAME:

Pointer = Smalled(lines)

FUNCTION:

This command is rather interesting in that it has a lot of possibilities for the future as well as the present, such as allowing easy OLM (on line messaging) to other users on other lines, as well as many various other uses. The first thing to understand about it is that it is, kind of, a small simple text-editor that is still quite advanced in it's use, while still remaining simple. It allows for full "backspacing" onto PREVIOUS lines and will only abort when RETURN is pressed and STICK when the maximum line has been reached. It is ideal for gathering questionair style information from users as well as MANY other uses.

YOU MUST "FreeVec" the pointer that is supplied (as long as it is not NULL) - This frees the memory containing the text.

EXAMPLE:

```
UBYTE *Pointer=NULL;

Pointer = Smalled(10);

if(Pointer)
{
    DMsg("Press a key to view what you typed:%WAITKEY\n\n");
    DMsg(Pointer);
    FreeVec(Pointer);
}
```

RETURNS:

Pointer to the text OR NULL (for failure or no text)

NOTES:

The pointer MUST be FreeVec(Pointer); IF a pointer is returned.

1.92 CENTRESTRING

NAME:

```
Pointer = CentreString(Destination, Source);
```

FUNCTION:

Takes a String and centres it according to the users screen width, placing it within the supplied Destination buffer, as well as returning a pointer to this buffer.

The string must be PURE in that it has no embedded control or AML codes.

EXAMPLE:

```
UBYTE Destination[100];  
  
CentreString(Destination,"Charmed I'm Sure");  
  
DMsgf("\n%s\n",Destination);
```

RETURNS:

Pointer to the Destination
