Easylife

Paul Hickman

COLLABORATORS			
	TITLE : Easylife		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Paul Hickman	April 15, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Easylife

1.1 Easylife - An Extension For AMOS Creator & AMOS Professional

```
                    Easylife Extension V1.10 - By Paul Hickman


      E-Mail: ph@doc.ic.ac.uk


Welcome to Easylife. Use the
                Browse >>
                 Button to read the entire manual
for the first time, or the hypertext links for reference.


                Introduction/Distribution
                Section 1: Internal AMOS Improvements


                Zone Commands

                String Commands

                Bitwise Commands
                        Structured Variables

                Miscellaneuos Commands
                Section 2: Amiga OS & Library utilisation


                Powerpacker & XPK Commands

                Pattern Matching

                AmigaDOS / Intuition Commands
                Section 3: Magic User Interface programming

  MUI Introduction
  Taglist Parsing
  MUI Commands & Functions
```

1.2 Introduction

                        Welcome To Easylife V1.10

This manual is presented as a amigaguide document, to allow quick reference. However, this often hinders reading it as an instruction manual for the first time. To overcome this use the
                Browse >>
                 button to move
through the pages in a logical order. You will see all pages of the manual except Menus/Indexes. Press
                Browse >>
                 now to continue.

1.3 Introduction / How to lead an EASY LIFE

How To Lead An EASY LIFE

The EASY LIFE extension is designed to do just that - make life easier - especially the parts of said life spent writing AMOS programs. The commands included are not designed to speed up graphics to amazing levels, even on an bog standard A500 which has the MC68000 pulled out, and replaced with a lego brick, or to put a starfield in the background (Why is it virtually every AMOS extension written has starfied commands???), but to speed up the actual computing work done in the background, to allow you access to some of AMOS's internal variables, and to provide support for powerpacker, pattern matching, message banks, zone banks, and in the latest version XPK and MUI.

1.4 Introduction / What else to I need?

What else do I need?

This distribution contains everything you need for the majority of easylifes commands & functions to run. However, you will also need:

- The XPKmaster & XPK compressors libraries, if you wish to use the

  Elxpk commands
  . These can be obtained from
  aminet in the directory /pub/aminet/util/pack/. I think the filename is xpk25usr.lha, but don't quote me on that...

- The MUI user & developers archives, if you wish to use the Elmui commands. You will also need Kickstart 2.04+ to use MUI. Download /pub/aminet/dev/gui/mui22usr.lha & mui22dev.lha. (The version numbers may be 23, 24 or even 30 (hope) by the time you read this...) I have tested versions 2.1 & 2.2. Don't use anything lower. Install the user archive with the installer program, and copy the AUTODOCS, mui-developer's guide, C examples, and assembler mui.i files from the mui22dev archive to somewhere on your harddisk (Don't even *think* of running MUI from a floppy - I suppose it is theoretically possible), and ditch the rest of the developers archive, unless you also want to program MUI from other languages.

These are not required, but you will benefit from having them:

- The AMOS intuition extension.

- Powerpacker V3.0 / V4.0

1.5 Introducton / Installation

Installation

There are 2 parts to installing easylife - getting the files in the right

places on your system, and telling AMOS they are there. The Installer script
will to the first part, but you must do the second. If you don't have the
installer:

 - Copy the /libs directory to your AMOSPro bootdisk / harddisk LIBS:,
   checking that these are *newer* versions of any library you replace.

 - Copy the accesories to AMOSPro_Accessories: , or some other permanent
   location if there is no space.

 - Copy the demo, docs, Procedures and Mui drawers somewhere permanent

 - Edit you user startup to add the docs drawer to the HELP: path. For
   KS 1.3 people, thsi means assign HELP: to the docs drawer. For KS2.0+
   if HELP: is already assigned (And it should be), use Assign ADD.

To install for AMOS Creator:

    (NOTE: AMOS Creator version is not in this archive)

 - Copy Lib/Easylife.Lib to the AMOS_System directory if you didn't
   run the installer script.

 - Run Config1_3.AMOS, load the default configuration, select 'Loaded
   Extensions' and enter :AMOS_System/Easylife.Lib" into slot 16. Then
   save the default configuration.

To install for AMOS Professional

 - Copy Lib/AMOSPro_Easylife.Lib to AMOSPro_System:APSystem/ if
   you didn't run the installer script.

 - Load AMOSPro, and select "Set Interpreter" from the config menu. Load
   the default configuration, click on loaded extensions, and enter
   'AMOSPro_Easylife.Lib' into slot 16. Then save the default configuration.

Both Versions.

 - Now quit & reload your version of AMOS.

NOTE: Additional installation may be required to use MUI

1.6 Introduction / Conventions

Conventions Used In This Manual

Each page of this manual describes a group of related commands, and/or
function calls, and is divided into the following sections.

Command Syntax

This shows the syntax of all commands & functions covered on the page. If a keyword is preceeded with an '=' sign, then it defines a function call. This means that it should be used as an AMOS expession - E.g.

```
A = Znsx(10)
Print Pp Len(0);
```

If it does not, then it is an AMOS command, and should only be used at the start of a statement - E.g.

```
Set Bank Name 10,"Fred    "
If A=10 Then Pp Load 1,"RAM:Test",2
```

The command/function name is followed by it's arguments, which are in capitals. If a particular command/function can have several argument formats, all are listed here.

Any words in italics are the formal parameters of the command / function, in all sections.

Description

This section contains a few paragraphs describing the usual action of the command / function in general terms.

Notes

This section describes any special cases for the command where it does not behave in the expected manner, exceptions to the explanation given in the description section, and any conditions under which it should not be used etc.

Errors

This section lists all possible errors messages generated by the command / function, and explains their most likely causes.

NOTE: 'Out of variable space', and 'Out of memory' may also occur with some commands, but are not listed, as they as generally not a result of a problem with the specific instruction, but with the entire program.

See Also

This section list related commands that are not in the same sub-section of the manual that you are reading, or are not easylife commands.

Bugs

This section is non-existant (I Hope) :-)

1.7 Introduction / Compatibility with older versions of easylife

```
            Compatibility with older versions of easylife


Firstly, this may be a little confusing, since I changed version numbers
halfway thourgh development to comform to commodore standards. The
versions so far are:

Phase 1
=======

V1.0        Original AMOS Creator Version

V1.1        Improved Version Creator Version

V1.2        Ditto.


Phase 2
=======

V1.3.1      Dual Creator / Pro version. Some new commands.

V1.3.2      Bugfix to previous version.

V1.3.3      More Bugfixes / AMOSPro specified optimisations.


Phase 3
=======

V1.3.4      Added the 'El' prefix to all command names.

V1.4        Internal restructuring - Bug fixes.

V1.4.1      XPK commands added.

V1.4.2      Bug fixes.

V1.4.3      Bug fixes.

V1.4.4      Elf Char now search for many characters.

V1.5        Primitive MUI support added. Intuition routines
            improved.

V1.06       Switched to commodore version numbering. Refined MUI
            functions.


Phase 4
=======


V1.07       Taglist editor support added & all MUI commands renamed.
```

Various other improvements. Structure variable commands added.

V1.08 Easylife.Library created. Structure variable commands moved to library.

V1.09 Amigaguide Viewer Written & AMOSPro help replaced. Minor bug fixes & changes to the other programs. Due to an error in the V1.08 token table, programs which use the new font commands will need retokenising.

This version should be nearly 100% compatible with other Phase 4 versions. The only changes are that the Ellock Font & Elunlock Fonts commands were removed in V1.08. Use Elopen Font instead.

Converting Phase 3 programs should mainly be done by Retokenising them (AMOSPro_Tutorials/ReTokenise.AMOS) - A few commands may need changing (E.g. Old style MUI comands / XPK commands). Phase 1 & 2 versions would have to changed by hand from a text editor using AMOSPro's Save AscII as the command names have all changed.

1.8 Zone Commands & Functions - Contents

Screen Zone Support

The AMOS Manuals imply that the only use of screen zones is to detect when objects or points lie within them. Easylife provides the commands necessary for you to use zones for any purpose that requires an area of the screen to be marked, and is particularly useful for control buttons.

There are three types of zone support command:

Section 1: Reading / Modifing standard zones

Reading a zones co-ordinates

Moving zones
Section 2: Multi-Zones

What is a multi-zone

Reserving multi-zones

Deallocating multi-zones

Defining a multi-zone

Reading co-ordinates

Checking multi-zones

1.9 Zones / Read a zones co-ordinates

Command Syntax

```
  =ElZnsx( ZONE )
  =ElZnsy( ZONE )
  =ElZnex( ZONE )
  =ElZney( ZONE )

  =ElZnsx( SCREEN , ZONE )
  =ElZnsy( SCREEN , ZONE )
  =ElZnex( SCREEN , ZONE )
  =ElZney( SCREEN , ZONE )
```

Description

When you use the AMOS command Set Zone, you supply it with the parameters 'sx,sy to ex,ey' to define the co-ordinates of the zone. These function calls allow you to read back the current values of these parameters for a given ZONE. E.g.

```
Set Zone 1,10,20 To 30,40
Print ElZnsx(1);",";ElZnsy(1)
Print ElZnex(1);",";ElZney(1)
```

Will display the values:

```
10, 20
30, 40
```

The optional SCREEN parameter specifies the screen number the zones are defined on. If as above, it is omitted, it is assumed they are defined on the current screen.

Notes

These commands return signed integers. (-32768 to 32767)

If a zone has been reserved, but not defined with Set Zone, all four functions will return 0.

Errors

Screen Not Opened

The SCREEN number you have defined is not open. If you have omitted this parameter, this error means that no screens are open.

Illegal Function Call

This means the specified ZONE has not been reserved on the specified SCREEN.

1.10 Zones / Moving Zones

Command Syntax

```
ElZn Shift SCREEN , DX , DY

ElZn Shift SCREEN , DX , DY , START To FINISH
```

Description

The first instruction will scroll all the zones defined on the given SCREEN DX pixels to the right, and DY pixels downwards.

To scroll to the left, make DX negative. To scroll upwards, make DY negative.

If specified START and FINISH allow only the zones with numbers between START and FINISH (Inclusive) to be scrolled. E.g.

```
Elzn Shift 0,-16,10,4 To 7
```

Will scroll zoes 4,5,6 & 7 to the left by 16 pixels, and down 10 on screen 0.

Notes

- Scrolling a zone does not change the image displayed on the screen, it merely moves the co-ordinates of your zones to reflect the changes to the image made by screen copy, scroll, or the AMOS Turbo Plus Blit commands.

```
    - You can shift a single zone by making both START and FINISH equal to
      the zone number.

    - ElZn Shift does not work with multi-zones.

    - It is not an error to shift a zone off the screen.

    - The arithmatic for zone co-ordinates is done modulo 65536. This means
      if you shift a zone with co-ordinates 10,10 to 50,20 by 20 pixels to
      the left, the new co-ordinates will be 65526,10 to 30,20.

      This will confuse the AMOS =Zone(X,Y), Mouse Zone, and related
      functions, but not
                Elznsx
                 and the other easylife
      zone reading commands.


Errors

    Screen Not Opened

    The given SCREEN is not open.


    Illegal function call

    Either:

    - No zones are reserved on the given screen

    - START or FINSH is greater than the number of zones reserved

    - START is greater than FINISH
```

1.11 Multi Zones / What is a multi zone

```
          What Is A Multi Zone?

A multi zone is similar to a normal screen zone. It is a rectangular area
of the screen, and you can detect if given co-ordinates fall within it.
The differences are:

  - Multi zones are identified by a GROUP number, and a ZONE
    number. Neither need be sequential, you can define zones
    1,3,10 of group 1, then zone 431 of group 839 if you want -
    it will still only take 4 zones worth of memory.

  - All Multi zones can in one group can be erased with one
    command, without affecting the other groups.

  - You can detect if co-ordinates lie in any multi zone, or if
```

they lie in any multi zone from one particular group.

- You can find all the zones a point lies in, not just the first one in the list (unlike standard zones).

1.12 Multi Zones / Reserving Space

Command Syntax

ElMz Reserve NUM

Description

This command is the equivilent of the Reserve Zone command. It reserves memory for the chosen number of zones. NUM should be even. If the value you supply is not, it is rounded up. A maximum of 5460 multi zones can be defined. (There is a good reason for that number!)

The Multi zones take the place of the normal screen zones in the screen data structure, so they cannot be used at the same time. Multi zone commands will produce an error message if you attempt to use them when normal screen zones are defined. Normal screen zones will not work with multi zones installed, but will not produce error messages, just unreliable results.

Notes

- When you call ElMz Reserve, the previously installed zones/multi zones are erased - you do not get a Zones already reserved error.

Out of memory

Not enough memory could be reseved for the zone table.

1.13 Multi Zones / Eraseing All Multi Zones

Command Syntax

Reserve zone

Description

This the standard AMOS reserve zone command. Used with no parameters, it will erase all multi zones. They are also erased properly when the screen is closed / default command used, or under any other circumstance when normal screen zones are removed.

See Also

Changes to AMOS Commands

1.14 Multi Zones / Defining & Erasing A Multi Zone

Command Syntax

```
ElMz Set GROUP , ID , X1 , Y1 To X2 , Y2
ElMz Set GROUP , ID
```

Description

This is virtually identical to Set Zone command, except you must supply a GROUP number. GROUP and ID can be any number between 1 and 65535. X1,Y1,X2,Y2 can be between -32768 and 32767.

X1,Y1 and X2,Y2 are automatically sorted so X1 <= X2, and Y1 <= Y2, so

Elmznsx
etc. always return the correct values.

The second form of the instruction erases the specified zone.

Notes

- If you call ElMz Set twice with the same GROUP & ID numbers, the old co-ordinates are overwritten.

Errors

Illegal Funciton Call

Neither GROUP or ID can be 0, or greater than 65535.

Zone table full

You have already used all
Reserved
multi zone space.

```
    Multi Zones Not Reserved

    You have not reserved any multi zone space with the
                ElMz Reserve
                , or have since
    used the standard
                Reserve Zone
                 command.
```

1.15 Multi Zones / Reading multi zone co-ordinates

```
                Command Syntax

  = ElMznsx( GROUP , ID )
  = ElMznsy( GROUP , ID )
  = ElMznex( GROUP , ID )
  = ElMzney( GROUP , ID )



Description

   These are multi zone equvilents of the commands to read a standard AMOS
   zone's co-ordinates,
                Elznsx
                 etc. Each function call returns the
   coresponding co-ordinate of the edge of the zone.



Notes

  - All operate on the current screen only.

  -
                Elmz Set
                 sorts X1,X2,Y1,Y2 so ElMznsx/y are always
    greater than ElMznex/y.

  - The values returned are signed (-32768 to 32767).


Errors

    Multi Zone Not Defined

    The zone with the specified ( GROUP , ID ) does not exist.


    Screen Not Opened

    Since these commands operate on the current screen, this
```

error means that no screen was open.

1.16 Multi Zones / Find the multi-zones containing a point

Command Syntax

```
= ElMzone( X , Y )
= ElMzone( X , Y , GROUP )
= ElMzonen
= ElMzoneg
```

Description

The first form of ElMzone returns the zone ID of the first multi zone on the current screen that the point X,Y lies within. The second form of Mzone only checks the zones in one particular GROUP.

ElMzonen returns the zone ID of the next multi zone that the point specified in the last ElMzone command lies within. If the second form of ElMzone was used to specify the point, ElMzonen will return the next zone in the searched GROUP that contains the point.

Mzoneg returns the group number of the last multi-zone ID returned by the previous ElMzone or ElMzonen commnad.

Notes

- All of these commands return 0 if there is no remaining zone which contains the point specified.
- ElMzoneg will still work if you specify a group in the ElMzone command, but it will only ever return group number GROUP, or 0 if no zone was found in the GROUP that contained the point.
- The zones containing the point won’t be returned in any particular order.

Errors

Multi Zones Not Reserved

No multi zones have been reserved on the current screen.

1.17 MultiZones / Removeing A Multi Zone Group

Command Syntax

ElMz Erase GROUP

Description

This command erases all zones in the given GROUP. Unlike standard AMOS zones, when you reserve multi-zones, you reserve space for a certain number of zones, but those zones may have any group and zone id numbers. Erasing a group of zones frees the space in the zone table for use by other groups.

Notes

- This command does not deallocated any memory.
- To remove all multi zones & deallocate the memory, use the standard AMOS

Reserve Zone
command.

- A single zone can be erased with the {"ElMz Set" link C_ElmzSet} command.

Errors

Multi Zones Not Reserved

You have not reserved any multi zones on the current screen.

1.18 Zone Banks / What Is A Zone Bank?

What Is A Zone Bank

A zone bank is an AMOS bank, which contains one or more sets of AMOS zone descriptions, created by the Zone Editor program. Zone banks have the same structure as multi zones - the bank is split into groups, and each group contains any number of zones, However unlike multi zones, a zone bank has all groups defined in numerical increasing order, e.g. With multi zones you can just define groups 1 & 10, but with a zone bank groups 2 - 9 must also be defined. The same applies to the zone numbers within each group of the bank.

Easylife allows you to use zones from a zone bank in 3 ways:

Install A Single Group As AMOS Zones

Install A Single Group As Multi Zones

Install All Groups As Multi Zones

1.19 Zone Bank / Installing A Group As AMOS Screen Zones

Command Syntax

```
ElZb Add SCREEN , BANK , GROUP
```

Description

This command installs zones from a Zone Bank, created by the zone editor program. SCREEN is the screen you wish to set the zones on. BANK is the bank number containing the zones, and GROUP is the set of zones within the bank you wish to install.

Notes

- Any previously reserved zones or multi zones are removed when this command is executed.
- Once installed, all normal AMOS & Easylife zone operations can be performed on the zones as if they were created by set zone instructions

Errors

Bank does not exist

The bank number given is that of a non-existant bank.

Not a Zone Bank

The bank number given is that of another type of bank. Zone banks are identified by them having the name "Zones   "

Illegal Function Call

The group number you have specified does not exist in the selected zone bank.

Screen not opened

The screen number you have selected is not open.

1.20 Zone Bank / Installing A Zone Bank Group As Multi Zones

Command Syntax

```
ElZb Multi Add BANK , GROUP
```

Description

This command installs all zones from the chosen group as multi zones. Unlike the other install instructions, this does not erase the currently defined zones. This means that you will have to reserve enough multi zones to contain the group first. The only zones overwriten by the command are those with the same GROUP & ID numbers as those in the bank.

Errors

Bank does not exist

The bank number given is that of a non-existant bank.

Not a Zone Bank

The bank number given is that of another type of bank. Zone banks are identified by them having the name "Zones "

Illegal Function Call

The group number you have specified does not exist in the selected zone bank.

Screen not opened

The screen number you have selected is not open.

Multi Zones Not Reserved

You must have already reserved some multi zones to contain the zone bank group.

Zone Table Full

There is not enough room in the space reserved for multi zones remaining to hold the entire group.

1.21 Zone Bank / Installing an entire zone bank as Multi Zones

Command Syntax

```
  ElZb Multi Add BANK
```

Description

```
    This command installs all zones in all groups the BANK as multi zones.
```

Notes

```
  - This command erases any previously defined (multi) zones and reserves
    the exact number of multi zones required to hold the entire zone bank.

  - The zones are installed on the current screen.
```

Errors

```
    Bank does not exist

    The bank number given is that of a non-existant bank.

    Not a Zone Bank

    The bank number given is that of another type of bank. Zone banks are
    identified by them having the name "Zones   "

    Screen not opened

    The screen number you have selected is not open.
```

1.22 String Functions - Contents

```
                                        String support functions

Section 1: Character Search Functions

                Character Searching

                Forwards Searches

                Backward Searches

                Control Character Searches
```

1.23 Character Search Functions

Character Search Functions

If you want to find the first occurance of a character in a string, you can use the AMOS functinon =instr$, but as this is designed to find substrings, it is in-efficient for single characters.

However more sigificantly, to find the last occurance of a character, or the first occurance of any of a set of characters in a string requies a loop in AMOS which is *very* inefficient. To overcome this, Easy Life provides a variety of string searching commands, most of which accept either AscII values to search for a single character, or a second string to search for any of the characters that occur in the string. There are also commands to search for any character but those list, and to search backwards.

All EasyLife 'Find' character commands begin with the prefix 'Elf'.

1.24 String Searches / Forwards Searhing

Command Syntax

```
=Elf Asc( S$ , A )
=Elf Char( S$ , A$ )
```

```
=Elf Asc( S$ , A , P )
=Elf Char(S$ , A$ , P )

=Elf Not Asc( S$ , A )
=Elf Not Char( S$ , A$ )
=Elf Not Asc( S$ , A , P )
=Elf Not Char( S$ , A$ , P )
```

Description

The Asc & Char varients of each form of the find function are identical, except the second argument (The character to find) is given as either an AscII value between 0 and 255, or a second string of characters. When a string is given, the search is for any of the characters in the string. These commands will not search for substrings.

The first form, =Elf Asc(S$,A) returns the first occurance of the character in the string S$. The second form, =Elf Asc(S$,A,P) begins searching a position P+1. Any occurance in position P, or before are ignored.

The "Elf Not" forms are equivilent to the simple forms with the same arguments, except they return the position of the first character found that is not the character specifed in A, or the first character that does not occur in the string A$.

Notes

- If no character is found, these functions return all return 0.
- When you specify the argument P, the search begins at position P+1. The AMOS =instr$ function would begin at position P. The reason for what at first seems strange behaviour is that the most frequent use of these commands is to parse a string which contains seperator characters e.g. spaces. With my method, to find the next occurance, you simply put the position of the last occurance as the P parameter of the next search.
- Any value of P is accepted, but is taken to be unsigned, so negative numbers are treated as very high positive numbers. If P is greater than the length of the string, 0 is returned.

Errors

Illegal Function Call

Either A$ is an empty string, or A is not between 0 and 255.

1.25 String Searches / Backwards Searching

```
                Command Syntax

  =Elf Last Asc( S$ , A )
  =Elf Last Char( S$ , A$ )
  =Elf Last Asc( S$ , A , P )
  =Elf Last Char( S$ , A$ , P )

  =Elf Last Not Asc( S$ , A )
  =Elf Last Not Char( S$ , A$ )
  =Elf Last Not Asc( S$ , A , P )
  =Elf Last Not Char( S$ , A$ , P )



Description

    Each of these commands coresponds to a
                Forward Searching
                 command, but
    begins the search from the end of the string S$, and works backwards.
    The Asc varients take the AscII value of the character to search for,
    whereas the Char varients take a second string containing the set of
    characters to search for.

    = Elf Last searches from the end of the string for the chosen character,
      returning the first position from which it occurs. If the optional
      parameter P is specified, the search begins at position P-1.

    = Elf Last Not searches from the end of the string for any character
      other than the chosen character, and is very useful for removing the
      padding from padded strings, or for removing trailing spaces.



Notes

  - If no character is found, these functions return 0.

  - When you specify the argument P, the search begins at position P-1.
    This is so the second occurance can be found by returning the position
    of the first as the argument P.

  - Any value of P is accepted, but is taken to be unsigned, so negative
    numbers are treated as very high positive numbers. If P is greater than
    the length of the string, the search begins at the end of the string.



Errors

    Illegal Function Call

    Either A$ is an empty string, or A is not between 0 and 255.
```

1.26 String Searches / Finding Control Characters

Command Syntax

```
  = Elf Control( S$ )
  = Elf Control( S$ , P )
```

Description

These special versions of the forward search commands find the first occurance of a character with AscII below 32 in the string S$ (Starting at position P+1 if P is supplied). The position of the first control character is returned, or 0 if the string contains no control characters.

E.g. This can be used to determine if a string is printable. A string which contains control characters may invoke any of the AMOS text formatting functions in chapter 8 of the AMOS Creator manual such as At(X,Y), Pen$(C), Cdown$, etc. (Chapter 5.06 of the AMOS Pro manual)

```
  If Not Elf Control(A$)
     Print A$
  Else
     Print "String Is Unprintable"
  End If
```

1.27 String Searches / Finding the Nth Occurance

Command Syntax

```
  = Elf Nth Char( S$ , A$ , N )
  = Elf Nth Asc( S$ , A , N )
```

Description

This varient of the find function finds the Nth occurance a character in a string. S$ is the string to search, and N is the occurance number you wish to find. The character to search for is the character with AscII value A in the Elf Nth Asc function, or the any characters in the string A$ in the Elf Nth Char functino.

The position of the Nth occurance of the character is returned, or 0 if the character(s) does/do not occur N times in the string.

Errors

Illegal Function Call

Either A$ is an empty string, or A is not between 0 and 255.

1.28 String Searches / Character Counting

Command Syntax

```
  = Elf Num Char( S$ , A$ )
  = Elf Num Asc( S$ , A )
```

Description

These functions return the number of times the specified character(s) are found in the string S$. In the Asc varient, A is the AscII code of the character to be counted. In the Char varient, occurances of any character from A$ are counted.

Notes

- If the string A$ contains more than one occurance of the same character it is still only counted once.

Errors

Illegal Function Call

Either A is not between 0 and 255, or A$ is an empty string.

1.29 Strings / Read a banks name

Command Syntax

```
  = ElBank Name$( BANK )
```

Description

This function call returns the name of the given BANK. This is the 8 character string displayed when you use the AMOS listbank command.

The string returned is always 8 characters long, and is padded with trailing spaces, which may be removed with:

```
  NAME$ = ElBank Name$(BANK)
  NAME$ = Left$(NAME$,Elf Last Not Asc(NAME$,32))
```

Errors

Bank Not Reserved

The obvious - you can't get the name of a non-existant bank.

1.30 Strings / Setting A Banks Name String

Command Syntax

```
Els Bank Name BANK , NAME$
```

Description

This command changes the name of the BANK which is returned by the

```
ElBank Name$
 function and the ListBank command to NAME$
```

Notes

- NAME$ must be exactly 8 characters long, so shorter strings should be padded with spaces E.g.

```
Els Bank Name BANK,ElPad Asc(NAME$,32,8)
```

- Some AMOS commands / programs use the bank name to detect the bank type, so you should be careful when changing the name of bank types other than Work or Data. Easylife itself uses the bank name to detect Message Banks and

```
Zone Banks
Errors
```

Bank Not Reserved

The bank specified does not exist.

Illegal Function Call

Either the bank number specified was not a legal bank number (e.g. it was negative), or NAME$ was not exactly 8 characters long.

1.31 Strings / String <--> Integer Conversion

Command Syntax

```
= ElLong$( NUM )
= ElLong( NUM$ )
```

```
  = ElWord$( NUM )
  = ElWord( NUM$ )
```

Description

ElLong$ converts the integer into a 4 byte string holding that integer, so that it may be output to a file compactly with a fixed length, or so that a list of integers may be built within a string easily.

ElLong converts the first 4 bytes of a string back into an AMOS integer.

ElWord$ and ElWord do the same thing, except only two bytes are used. This means that the integer must be between -32768 and 32767.

Notes

- The integer is stored in the string as it is stored in memory - most sigificant byte first.
- ElWord$ does not give error messages if the value is out of range, it simply stores the lower 2 bytes. Therefore you can use word$ to store unsigned words (0-65535), but you must then read them bank with:

  NUM = Asc(NUM$)*256+Asc(Mid$(NUM$,2))

  As Elword will return negative numbers for 32768-65535.

Errors

Illegal Function Call

The NUM$ string must be at least 2 bytes long in ElWord$, and at least 4 bytes in length in ElLong$.

1.32 Strings / Reading Memory As A String

Command Syntax

```
  =ElMem$( ADDR , SLENGTH )
  =ElMem$( ADDR , SLENGTH , DELIMITER )
```

Description

The first format copies then next SLENGTH bytes from address ADDR into an AMOS string, and returns that string. The second version attempts to do the same thing, but stops when it finds the first occurance of a

character with AscII code DELIMETER.

AMOS already has peek,deek & leek - thing of this as 'Seek' (!)

Notes

- A COPY of the memory area is returned. It may be modified like any other string, without affecting the memory area, and conversly any changes to the memory made since it was read are not reflected in the string.

- If the memory reading is terminated by reading a DELIMETER character, that character is not returned as the last character of the string - only those characters up to the DELIMETER are returned. 0 is a useful delimeter for reading strings from Amiga OS functions, which are null-terminated.

Errors

Variable Buffer Full / Out Of Memory

These may occur at any time, but this command is particularly susceptable if the string is very large, or the delimeter does not occur where you expect it to.

Illegal Function Call

SLENGTH must be between 1 and 65535 inclusive, and DELIMETER between 0 and 255.

1.33 Strings / Writing A String To Memory

Command Syntax

```
Elmem ADDR , S$
= Elmem inc( ADDR , S$ )
```

Description

This command copies the string S$ into memory begining at address ADDR. Only the actual characters in the string are copied - the length does not preceed it as with AMOS strings within the variable buffer, and it is not automatically null terminated like C strings.

The second format returns ADDR+Len(S$) as a result, allowing many strings to be copied into consecutive memory addresses easily using:

```
ADDR = Elmem Inc(ADDR,S1$)
ADDR = Elmem Inc(ADDR,S2$)
```

```
  Elmem ADDR,S3$
```

Notes

- If S$ is an empty string, this command has effect.

Errors

If you write to just any memory address, *anything* can happen - make sure you know what is at the address you write to.

1.34 Strings / Padded Strings

Command Syntax

```
  = ElPad Asc(S$,A,L)
  = ElPad Char(S$,A$,L)
```

Description

If the length of the string S$ is greater than or equal to L, these two functions return S$. Otherwise they return a string which is S$ followed by enough repetitons of the character A/A$ to make it's length L. The first varient takes the AscII code of the "pad character" in the A argument. The second varient uses the first character of A$.

Notes

- If A$ contains more than one character, the second and subsequent characters are ignored. In the future I intend to change this to repeatedly use the whole of A$ to pad S$.

Errors

Illegal Function Call

A must be between 0 and 255, and A$ must not be and empty string.

1.35 Strings / Message Banks

Command Syntax

```
  = ElMessage$( BANK , GROUP , NUMBER )
```

```
  = ElMessage$( ADDR , GROUP , NUMBER )
```

Description

Easy Life makes life much easier when using the Message Bank Compiler PratchED extension program. This command reads a message from a bank into a string.

BANK is the bank number to read the message from, GROUP and NUMBER give the identifier of the message your wish to read. Alternatively, you may give the absolute address of the message bank in memory in the ADDR parameter.

For more information, read the message bank compiler documentation. (Which one day, I might even release!)

Example:

To read a message 1,2 from a crunched message bank that has been loaded into powerpacker buffer 0, use the call:

```
M$ = ElMessage$(ElPp Buf(0)+20,1,2)
```

The +20 is because all AMOS banks have a header of 20 bytes.

Notes

- The method defined in the above example is not the only way to read crunched message banks - you can use {"Elxpk Load" link C_Elxpkload}
- A COPY of the message is returned in the string, and modifying it will not affect the stored message.
- The use of message banks in AMOS professional is limited in AMOSPRO, as the text string part of resource banks do pretty much the same thing, except message banks will be AMOS Creator compatible.

Errors

Bank Not Reserved

The first parameter is a valid bank number, but that bank does not exist. If the first parameter is not a legal bank number, it is taken to be the absolute address of the message bank.

Not A Message Bank

The chosen bank exists, but is not a message bank, or there is not message bank at the specified address. Message banks are recognised by

```
    thier name, so you must not rename them with the
                Els Bank Name
                 command.


    Illegal Function Call

    Either the GROUP or message NUMBER is illegal for this particular bank.
    The error can be avoided using the
                Message Exists
                 function
    first.
```

1.36 Strings / Testing if a message exists

```
                Command Syntax

  = ElMessage Exists( BANK , NAME , START )
  = ElMessage Exists( ADDR , NAME , START )


Description

    This command is used to return whether a given message has been defined
    within a message bank, or not as a boolean. The arguments, and error
    conditions are the same as for the
                ElMessage$
                 function, but you
    will never get an illegal function call as the purpose of this function
    is to determine whether the message exists.
```

1.37 Bitwise Commands - Contents

```
                                            Bitwise Commands



                Bit Testing

                Bit Modifying

                Sign Extension
```

1.38 Bits / Bit Testing

```
Command Syntax

  =ElWtst( BIT , ADDR )
  =ElLtst( BIT , ADDR )
```

Description

The AMOS =Btst function allows you to detect if a bit is set in a given byte of memory, or in an integer variable. EasyLife provides these two functions to test if a bit is set in words/longwords of memory.

BIT is the bitnumber to test. Bits are numbered from the right, 15 being the highest bit of a word, and 31 the highest bit of a longword.

ADDR is the address of the word/longword to test.

True is returned if the bit is set in the given word/longword.

Notes

- The second argument ADDR is always taken as an address, even if it is a simple variable, and not a complex expression. To test the bits of a variable use the AMOS =btst function - this allows you to test all 31 bits of integer variables.

Errors

Illegal Function Call

BIT must be in the range 0-15 for words, and 0-31 for longwords. ADDR must be a valid even address.

1.39 Bits / Bit Modifying

Command Syntax

```
ElWset BIT , ADDR
ElLset BIT , ADDR

ElWclr BIT , ADDR
ElLclr BIT , ADDR

ElWchg BIT , ADDR
ElLchg BIT , ADDR
```

Description

These commands are equivilent to AMOS Bset,Bclr and Bchg instructions, but they allow you to modify the bits of words & longwords of memory.

ElWset sets bit BIT of the word at address ADDR.

ElWclr clears bit BIT of the word at address ADDR.
ElWchg inverts bit BIT of the word at address ADDR.

The ElL... commands perform the same functions on longwords.

Errors

Illegal Function Call

BIT must be in the range 0-15 for words, and 0-31 for longwords. ADDR must be a valid even address.

1.40 Bits / Sign Extension

Command Syntax

```
= ElExtb( NUM )
= ElExtw( NUM )
```

Descrition

These functions will sign extend numbers from bytes/words to long words. To see what this means, imagine AMOS integers as 32 bit binary numbers e.g. 42 is:

```
00000000 00000000 00000000 00101010
                           *
```

Sign extending from a byte (ElExtb) looks at bit 7 (*) and copies it's value into all the bits to it's left. This has no effect on the number 42, but 139 is changed:

```
00000000 00000000 00000000 10001011
                           *
```

returns:

```
11111111 11111111 11111111 10001011
```

AMOS will interpret the new bit pattern as -117,because 10001011 is the bit pattern for -117 in signed 8 bit arithmatic, but for 139, in unsigned 8 bit arithmatic.

ElExtw copies bit 15 into all the places to it's left. Extb & Extw can be used to translate signed bytes & words into AMOS integers.

Notes

- The state of bits 32-16 (Elextw) / 32-8 (Elextb) is ignored by the sign extension. Only bits 15/7 count.

- Both commands sign extend to a longword unlike the 68000 sign extension instructions on which they are based.

1.41 Fonts / Contents

New Font Commands

The original font handling of AMOS was, to put it mildly abyssmal. Easylife provides you with a system to locate the font you want quickly & easily, and you won't have to worry about it being flushed out of memory because AMOS thinks your not using it anymore.

=Elopen Font

Elset Font

Elclose Font

Elclose Fonts

=Ellock Font

Elunlock Fonts

1.42 Fonts / Opening A Font

Command Syntax

=Elopen Font (NAME$, SIZE)

Description

When you want to use a new font, call Elopen Font giving the name of the font (Including a ".font" at the end), and the point size of the font, and you will be returned a font ID number to use with Elset Font.

If the font you request is not in memory, it will be loaded from disk.

Notes

- The value returned is a pointer, not a consecutive integer like AMOS font numbers.

- You do not need to use any of the AMOS 'Get Fonts' commands - Elopen Font is a replacement for these.

- If you open the same font twice, you are returned the original pointer the second time, and the font is only actually opened once. Therefore you should only close it once.

- You can access the AmigaOS 'TextFont' structure of the opened font with:

```
F=Open Font("topaz.font",8)
TF=Leek(F+4)
```

  TF is now the address of the TextFont structure for topaz 8.

Errors

Can't open diskfont.library

Either diskfont.library is not in LIBS:, or there is not enough memory to load it.

Unable to lock font

The font you have specified cannot be found.

1.43 Fonts / Using Fonts

Command Syntax

```
Elset Font FONTID
```

Description

This command behaves the same as the AMOS 'Set Font' command, except it take a FONTID returned from Elopen Font as a parameter instead of an AMOS font number. See the AMOSPro manual on Set Font for more information.

Notes

- You do not need to use any of the 'Get Fonts' commands before using Elset Font.

Errors

Illegal Function Call

The parameter you supplied is not a FONTID returned from Elopen Font (Or it has been closed again).

1.44 Fonts / Closing A Font

Command Syntax

```
Elclose Font FONTID
Elclose Fonts
```

Description

Elclose Font is called to tell the OS that you are finished using a font you have previously opened, and that it is free to deallocate the memory assigned to that font if no other task is using it. The parameter must be a font returned by Elopen Font.

Elclose Fonts closes all fonts that are currently open. This command is automatically called by the easylife default routine.

Errors

Illegal Function Call

The parameter you have passed to Elclose Font is not an open FONTID.

1.45 Fonts / Old Commands

Command Syntax

```
=Ellock Font( NAME$ , SIZE )
Elunlock Fonts
```

Description

These 2 commands have been removed from easylife, as they were unreliable with KS2.0+, and have been superceeded by the new Open/Close/Set font commands.

1.46 XPK / Powerpacker Compression - Contents

Powerpacker & XPK Compression

1.47 Compression / What Is Powerpacker?

What Is Powerpacker ?

Powerpacker is a popular Program and Data cruncher for the amiga, written by nico francois (Also author of reqtools). Early versions were public domain, but more recent & much faster, not to mention more efficient versions are commerical programs.

Easylife not only allows you to load data crunched with powerpacker into your AMOS programs, but also lets you save data crunched with the powerpacker algorithmn using the speed & high compression ration of the latest commerical

```
version. (I.E. It calls powerpacker.library :-)


You Can:

  - Read files crunched in data mode by powerpacker or any
    compatable program.

  - Load the files crunched by Easylife into powerpacker, ppmore etc.

  - Crunch any block of memory and save it to a file.


However, you cannot:

  - Read or Write crunched executable files / segments. Easylife only loads
    & crunches as data.

  - Use Powerpackers encrypted mode for reading or crunching. This is
    because it pops the password window up on an intuition screen.


You can also load powerpacked data directly to an AMOS bank using the

                Elxpk Bload
                 function.
```

1.48 Compression / Loading The Powerpacker Library

```
                Command Syntax

  ElPp Keep On
  ElPp Keep Off


Description

    To use the commands
                ElPp Load
                 &
                ElPp Crunch
                , the powerpacker library
    version 35 or greater is needed, therefore the file powerpacker.library
    (supplied) must be in LIBS:

    The library is loaded into memory when you first use either of these
    commands, but may sometimes be removed again by the exec memory manger
    afterwards. To make sure the library stays in memory these two commands
    are provided.

    Using Pp Keep On makes Easylife load the powerpacker library if it is
    not already in memory, and prevents being removed from memory until you
    use Pp Keep Off.

    Pp Keep Off does not always removed the library from memory - other
```

processes may also be using it, but it informs the memory manager that EasyLife has no objection to it being removed.

Notes

- Pp Keep Off is automatically called when you use the AMOS

Default
command.

Errors

Unable To Load PowerPacker Library V35+

Either the file powerpacker.library is not in LIBS:, the device LIBS: could not be accessed, or the version of the library in LIBS: is not high enough.

A suitable version of the library is supplied with EasyLife.

1.49 Loading Powerpacked Data

Command Syntax

```
ElPp Load BUF , FILE$ , DECRUNCH
```

Description

This command will load a file into memory, then decrunch it, if it was compacted with powerpacker. BUF is a number from 0-7 used to choose which of the 8 buffers to load the file into.

FILE$ is the name of the file to load. It is important that the full path be given, and that the file exists. E.g.

```
If F$<>""
    If Exist(F$)
        Pp Load 0,F$,2
    Else
        Print "File Not Found"
    End If
End If
```

DECRUNCH must be an integer between 0 and 4, and determines what happens while the file is decrunched. I recommend that option 3 is not used with AMOS screens, but it does not generate an error message, as your program may run on an intuition screen, if you have the intuition extension.

0 : Flash colour 0
1 : Flash colour 1

```
    2 : Flash colour 17 (Mouse Pointer - Recomended)
    3 : Wobble Screen (No effect on AMOS screens)
    4 : Do nothing while decrunching
```

Notes

- If the chosen buffer already contained data, it is freed first.
- Pp Load will load uncrunched data without any problems, so you don't have to worry about whether the file you are loading is crunched or not. The only problem is if it was not crunched with powerpacker.
- AMAL & Bobupdates etc. are temporarily suspended while decrunching takes place.

Errors

Can't Load Powerpacker Library V35+

The [4\2\Powerpacker Library] is required to be in LIBS: even if the file your are loading in not crunched.

Illegal Powerpacker Header

Either the file is corrupt, was crunched with unrecognised version of powerpacker, or was not crunched as a "Data File".

File Encrypted - Can't Decrunch

Easylife does not support decrunching encrypted files.

Out of Memory while Loading / Decrunching File

Oh no! (As the lemming said to the warhead...)

Unable To Open File

Powerpacker library could not open the filename you passed it. It probably means it doesn't exist, but it could be read protect bits, or another reason.

Error Reading File

A Disk Error occured while reading in the crunched file.

1.50 Compression / Accessing PowerPacker Buffers

Command Syntax

```
= ElPp Buf( NUM )
= ElPp Len( NUM )
```

Description

An Easylife Powerpacker Buffer is similar to an AMOS bank of type "work". ElPp Buf returns the address of the start of the buffer. It is similar to the start() function for banks. ElPp len returns the length of the buffer. It is similar to the length() function for banks. Vaild buffer numbers are 0 - 7.

The main differences between buffers & 'work' banks are:

- A Buffer can be created explicitly with the ElPp Allocate command, or implicitly by the ElPp load function. ElPpLoad cannot load directly to a bank. (However this can be done via the Elxpk Load function.

- If you try to recreate an existing buffer, the old buffer is freed first. You do not get an error, as you would with AMOS banks.

- In AMOS Creator, one set of powerpacker buffers is shared between all AMOS programs. If you have two programs loaded into AMOS, and one Prun's the other, the second will share the same buffers with the first. Therefore powerpacker buffers can be used to hold shared data.

Notes

- ElPp Buf & ElPp Len do not require the powerpacker library.

- The buffer can be saved with either Bsave, or ElPp Crunch, or

  Elxpk Bsave.

- If the buffer is not allocated, both functions return 0.

- You can also use Elmem$ to transfer the buffer contents to the AMOS variable buffer. If the file is a text file, you can read it one line at a time with:

```
ElPp Load 0,file$,2
POS=0
While POS<ElPp Len(0)
   A$=ElMem$(ElPp Buf(0)+POS,Pp Len(0)-POS,10)
```

```
     POS=POS+Len(A$)+1
     '
      'Now do whatever your going to do with the
      'line in A$
      '
  Wend
  ElPp Free 0
```

This segment of code will read the buffer up to asc code 10 (Line Feed) into the string A$, and point POS to the next line. If no Line Feed is found, the remainder of the buffer is read into A$

Errors

Illegal Function Call

You must supply a valid buffer number (0-7).

1.51 Compression / Disposing of Powerpacker Buffers

Command Syntax

```
  ElPp Free NUM
  ElPp Free All
```

Description

The first form of this command removes buffer NUM , returning the memory to the system. The second form removes all buffers.

Notes

- Freeing a buffer which is not allocated does not cause an error, it does nothing.
- ElPp Free All can also be
  Implictly called
  .

Errors

Illegal Function Call

You must give a valid buffer number (0 - 7)

1.52 Compression / Saving Powerpacked Data

Command Syntax

```
  = ElPp Crunch( FILE$ , START , LENGTH , EFFICIENCY , BUFFER )
```

Description

Easy life allows you to save any block of memory, after crunching it with the powerpacker routine - it does not have to be a powerpacker buffer.

ElPp Crunch will crunch a block of memory, save it to a file, and return the length of the crunched file.

FILE$ = The name to save the crunched file to. It must have a full path, and the directory must exist.

START, and LENGTH define the block of memory to save.

EFFICIENCY is an integer between 0 and 4 setting the crunch effeciency (0 = Fastest Speed, Poorest Compaction, 4 = Slowest Speed, Best Compaction).

BUFFER is the size of crunch speedup buffer to use. 0=Large, 1=Medium, 2=Small. If there is not enough memory for the buffer you choose, a smaller one will be used

Notes

- To save a powerpacker buffer, use the call:

  ```
  NL = ElPpCrunch(FILE$,ElPp Buf(NO),ElPp Len(NO),EFF,0)
  ```

- To save the string S$, use:

  ```
  NL = ElPpCrunch(FILE$,Varptr(S$),Len(S$),EFF,0)
  ```

- Don't confuse the speedup buffer of ElPp Crunch with the powerpacker buffers. The speedup buffer, is used to accelerate the crunching speed and is internal to the powerpacker.library.

- IMPORTANT: The crunched data overwrites the uncrunched data before it is saved - If you need to keep the original, make a copy before crunching.

Errors

Illegal Function Call

LENGTH must be a positive integer.
EFFICENCY must be in the range 0-4.
BUFFER must be in the range 0-2.

Crunched File Longer Than Source

If the crunched file becomes longer than the uncrunched file, an error occurs, as crunched data overwrites the original, and if it is longer, next bit of original data to crunch is corrupted. Therefore you should always place On Error commands around the section code using ElPp Crunch, so you have save the file uncrunched if crunching fails. (Or use the AMOSPro 'Trap' instruction).

Various I/O errors can also occur when saving the file.

1.53 Manual Buffer Allocation

Command Syntax

```
ElPp Allocate NO, LENGTH
```

Description

If you are using powerpacker buffers to load crunched files, you do not need to allocate the buffer, as this is automatically done by the ElPp Load instruction. However, if you want to take advantage of the fact that all AMOS Creator programs share one set of buffers to pass data between them, or if you have simply run out of banks for AMOS 'work' banks, you can use ElPp Allocate to manually reserve a buffer.

NO is the buffer number to allocate (0-7).

LENGTH is the number of bytes to allocate to the buffer.

Notes

- Powerpacker buffers will be allocated in fast memory if possible, otherwise chip memory will be used.
- I can see no reason for using Elpp Allocate in AMOSPro, apart from not having to convert creator programs.

Errors

Illegal Function Call

NO must be in the range 0 - 7. LENGTH must be a positive integer.

1.54 Compression / What Is XPK ?

```
                                                What Is XPK ?

XPK is an acronym for eXternal PacKing, and is a collection of libraries. All
outside programs interact with XPK via the xpkmaster.library, which provides
routines to crunch & decrunch data, the source of which may be a block of
memory, or a file, as may the destination. When calling the XPK library, you
specify a packing method in the form of a 4 character string e.g NUKE. This
is the name of the XPK compression library which contains the actual packing
algorithmn you wish to use.

The different compressors vary in their speed of compression / decompression,
their compression ratio, and their suitablitity to certain types of file, but
all share a common interface and calling commands from Easylife.

I recomend the use of XPK over powerpacker, as it is more general purpose,
and has a better interface in easylife, plus the fact that the xpkmaster
library will load & decrunch powerpacked files anyway.

NOTE: The XPK libraries are not included in this distribution, you must
obtain the
                XPK compression archive
                 seperately.
```

1.55 Compession / Loading XPK crunched banks

```
                Command Syntax

  Elxpk Load FILENAME$
  Elxpk Load FILENAME$ To BNKNO

  Elxpk Load FILENAME$ , PASSWORD$
  Elxpk Load FILENAME$ , PASSWORD$ To BNKNO



Desscription

    The Elxpk Load command loads an AMOSPro bank that has been saved with
    the
                Elxpk Save
                 command. The bank may be packed with any XPK compressor,
    as long as the required compressor library is available. If no bank
    number is specified, the bank is loaded back to the number from which
    it was saved.



Notes

  - You cannot load uncrunched or powerpacked banks with this method since
    they are normal .Abk files, as the .Abk format is different to the
    format in which Elxpk Save stores the bank file. However, if you
```

```
    save your banks with:

   Bsave FILENAME$,Start(BNKNO-24) To Start(BNKNO)+Length(BNKNO)

    You can load them back with this command if they are uncrunched, or
    powerpacked.

  - This command does not work for sprite / icon banks (Yet)



Errors

    See the
                Elxpk Error
                 function for details.
```

1.56 Compression / Loading XPK Crunched Data

```
                Command Syntax

  Elxpk Bload FILENAME$ To ADDR
  Elxpk Bload FILENAME$ , PASSWORD$ To ADDR



Description

    These commands load XPK crunched data directly into memory at address
    ADDR. You must have allocated enough memory for the uncompressed file,
    plus 256 bytes decompression space. Elxpk Bload will transparently
    load uncrunched data &
                powerpacked
                 data, but you must still allocate
    the 256 bytes.



Notes

  -
                Elxpk Lof
                 can be used to find the uncompressed length.

  - If the address you are loading to is an AMOS work bank, you can
    use the AMOSPro bank shrink command to remove the extra 256 bytes
    after loading.

  - The value 256 is from the xpk.i file's XPK_MARGIN equate for version
    2.4 of the XPK master library. For newer versions, check this file to
    see if the margin is increased.



Errors
```

See the
Elxpk Error
function for details.

1.57 Compession / Saving XPK Crunched Banks

Command Syntax

```
Elxpk Save BNKNO To FILENAME$ , METHOD$
Elxpk Save BNKNO To FILENAME$ , METHOD$ , PASSWORD$
```

Description

These commands will compress bank BNKNO, and save the compressed data to FILENAME$. METHOD$ is the 7 character string. The first 4 letters are then name of the compressor library to use. These are followed by a '.' and a two digit decimal number to indicate the depth of compression - e.g. "HUFF.23" will use the dynamic huffman packing method, at 23% of its maximum efficiency. Some methods will ignore the efficiency setting as they only operate at one level. Others may have say 4 levels, in which caes the values 0 - 24 will the first level, 25-50 the second etc.

If the packing method you select also encrypts the data, you must supply a password string for the encryption. See the XPK documentation for details of the various packing methods.

Notes

- You may not save sprite or icon banks with this command
Yet
  - In general, the higher the efficiency value, the smaller the ↩ file, but
the longer it takes to (de)crunch.

- Crunched banks can only be reloaded with the
Elxpk Load
command.

- Unlike
Elpp Crunch
, Elxpk save does not destroy the original copy of the data that your are crunching & saving.

Errors

See the
Elxpk Error

function for details.

1.58 Compression / Saving XPK Crunched Raw Data

Command Syntax

```
Elxpk Bsave START , LENGTH To FILENAME$ , METHOD$
Elxpk Bsave START , LENGTH To FILENAME$ , METHOD$ , PASSWORD$
```

Description

These commands save the block of memory from START, for LENGTH bytes to FILENAME$. The byte at address START is saved. The byte at address END is not, as with the normal AMOS Bsave command. METHOD$ is the XPK packing method to use, using the same format for the string as the

Elxpk Save
function. PASSWORD$ contains the encryption string for those packing methods which support encryption.

Notes

- Unlike
Elpp Crunch
, Elxpk Bsave does not destroy the original copy of the data that your are crunching & saving.

Errors

See the
Elxpk Error
function for details.

1.59 Compression / Finding the length of an XPK crunched file

Command Syntax

```
=Elxpk Lof( FILENAME$ )
```

Description

This function will return the length of the file FILENAME$, just like the normal AMOS Lof function (Except it take the filename, not a channel number). However, if the file has been compresed with

XPK
, or

Powerpacker
the length of the file once it has been uncompressed is returned.

Notes

- Elxpk Lof does not actually need to decrunch the file to find its length, so this command is reasonably fast & uses little memory.

Errors

See the
Elxpk Error
function for details.

1.60 Compression / Listing XPK Packing methods

In the @{ "Future" link todo} I intend to provide EasyLife functions for listing the types of XPK compression method that are available, and some of their properties to aid the building of packing method requesters. Until then, the only way to list the methods is to use Dir First$ / Dir Next$ on the directory 'LIBS:Compressors/'.

1.61 Compression / Handling XPK Errors

Command Syntax

=Elxpk Error

Description

When an error occurs with any of the XPK functions, one of 2 things will happen:

- It it is an obvious error in one of the parameters (e.g. BNKNO=0 in Elxpk Save) an 'Illegal Function Call' error will occur.

- It it is an error generated by the XPK library, the error message 'An XPK Error Has Occured' is displayed.

When this happens, you should call Elxpk Error to return the error number. The meanings of the various error numbers are listed below. Those marked

```
 with a '*' are most likely to occur from Easylife. If any marked with a
 '!' occur, please E-Mail me with the code that caused the error, as this
 probably means a bug in Easylife. This information is from the xpk.i file,
 by Christian Schneider * U. Dominik Mueller.

  0  No error has occured.
 -1! This function not is implemented
 -2! No files allowed for this function
 -3  Input error happened.
 -4  Output error happened.
 -5* Check sum test failed - corrupt file
 -6* Packed file's version newer than your libraries
 -7* Out of memory
 -8! For not-reentrant libraries
 -9  Was not packed with this library
 -10!  Output buffer too small
 -11!  Input buffer too large
 -12 This packing mode not supported
 -13*  Password needed for decoding this file
 -14*  Packed file is corrupt
 -15*  Required library is missing
 -16!  Caller's TagList was screwed up
 -17*  Would have caused data expansion
 -18 Can't find requested method
 -19!  Operation aborted by user
 -20 Input file is truncated
 -21 Better CPU required for this library
 -22 Data are already XPacked
 -23 Data not packed
 -24*  File already exists
 -25 Master library too old
 -26 Sub library too old
 -27*  Cannot encrypt
 -28!  Can't get info on that packer
 -29 This compression method is lossy
 -30 Compression hardware required
 -31 Compression hardware failed
 -32*  Password was wrong
```

1.62 Patterm Matching Commands - Contents

```
                                    Pattern Matching Commands


             What Is A Pattern ?

             Pattern Control Characters

             Simple Pattern Matching

             Repeated Pattern Matching

             Testing for patterns
```

```
                Escaping A String
                NOTE: The Easylife pattern matching commands require pattern. ↩
                   library to be
      in your LIBS: directory.
```

1.63 Patterns / What is a Pattern ?

```
                                                What is a pattern ?

   A pattern is a string with some special characters which match to one
   or more characters in another target string. The most common use of
   patterns is in the Amiga Shell / CLI windows, when you use:

        Copy DF0:C/#? RAM:


   To copy every file in the DF0:c directory to RAM:. '#?' is a pattern
   which matches any filename string. 'a?b' is also a pattern which only
   matches 3 character strings in which the first characters is 'a' and the
   last is 'b'.

   Easylife uses the pattern.library to provide the pattern matching. This
   is loaded from your LIBS: directory when you first use a pattern command
   (Except ElPat free, which doesn't load the library if no pattern has been
   defined). This libarary is also used by the CSH command shell program,
   so users of this program should already be familar with the syntax of

                Pattern Expressions
                .
```

1.64 Patterns / Pattern Control Characters

```
Expressions

An expression in a pattern is a single character, or any string of
characters in parenthesis (). Every expression will match itself, and
only itself in the target string it is matched with, unless it contains
control characters. Examples:

  "a" only matches the string "a"
  "fred" only matches the string "fred"
  "(hello) world" only matches the string "hello world"


Control Characters


%   This character matches the empty string "". Although it seems of
    little use, it can be important when used  with the | character
```

```
?   This character matches any single character, as with AmigaDOS.
    Example:

  "b?b" matches the string "bab", but not "baab".


#   Preceeding any expression with a # means it can be repeated 0 or more
    times. Example:

  The pattern "#(ab)" matches the strings "","ab",
  "abab","ababab" etc. This can be combined with the
  ? to match anything as with AmigaDos - "#?".

    NOTE: If an unescaped # is the last character of a pattern, any commands
          attempting to use the pattern will produce an Illegal Function
          Call error.


*   This is a shortcut for the pattern "#?", and means match anything
    E.g.

   "a*" matches "a","and","aardvark" etc.


|  This is the 'or' character, and allows alternative patterns e.g.

  "ab|cd" will match the strings "ab" and "cd", but nothing else.

    Usually you will have to suround the optional patterns with parenthesis
    to make them appear to be a single expression. The % character can be
    used as an alternative to allow the empty pattern - Example:

  "(ab|cd|%)" matches the strings "","ab","cd" only.

    NOTE: If any of the alternative patterns is empty, any command using
          the pattern will produce an illegal function call.


~   Negates the following expression.  Example:

  "~(ab*)" matches "","and","bass","can", etc. and anything else
       that doesn't begin with "ab".

    NOTE: "~ab*" is different to "~(ab*)", as "~ab*"
  matches anything that doesn't begin with "a", but
  does have a "b" as the second character.

    NOTE: If an unescaped ~ is the last character of the pattern, any
          commands using the pattern will produce an illegal function call.


[]  A string of expressions enclosed in square brackets will match any one
```

```
    of the expressions in the target string. Example:

  "[abc]" matches "a", "b" and "c", but nothing else.
  It is a shorthand for "(a|b|c)".

    Remember that expressions can be single characters (As Above) or
    parethansised strings - Exmaple:

  "[(ab)c(de)]" matches "ab","c" and "de" only.

    You can also specify character ranges in square brackets, e.g "[0-9]*"
    will match any positive integer. This means if you want to match the "-"
    character, it must be either the first or last character in the brackets
    E.g. "[-0-9a-z ]*" means any lower case word, spaces, digits, and the
    "-" character.

    Also, if the first character in the square brackets is a ~, all
    characters except those in the in the brackets are matched, E.g.
    "[~a-z]" will match all single characters that are not lower case
    letters.

    NOTE: The string in square brackets is always treated as a single
          expression - you never need to put parenthesis around the square
          brackets.


'   This character "Escapes" the next character. This means that if the
    next character in another control character, that special meaning is
    ignored, so "a'*" will only match the string "a*". "'" can be used
    to escape itself E.g. "''" matches the string "'".

    NOTE: You can also use ' to escape parenthesis.


Important Note:

    All parenthesis & square brackets must be paired (Unless they are
    escaped), otherwise any command using the pattern will produce an
    illegal function call.
```

1.65 Patterns / Simple Pattern Matching

```
                Command Syntax

  = ElPat Case( P$ , S$ )
  = ElPat NoCase( P$ , S$ )


Description

    These commands return True (-1) if the string S$ matches the pattern
    P$, or false (0) if it doesn't. The first form is case sensitive to
    alphabetic characters, the second isn't.
```

Errors

Illegal Funciton Call

There is an error in the
Pattern Definition
.

1.66 Patterns / Repeated Pattern Matches

Command Syntax

```
Elpat Set Case P$
Elpat Set Nocase P$

=El Pat Def( S$ )

ElPat Free
```

Description

The
Pat Case & El Pat Nocase
functions are not very efficient when you have to repeatedly match many strings with the same pattern, as the pattern has to be checked & converted to an internal format every time you test a string against the pattern. These commands split the pattern matching into 3 phases:

ElPat Set Case & ElPat Set Nocase are used to select a pattern to match strings against. These commands check the validity of the pattern, and compile it to the internal format.

You can then test any string S$ against the most recently defined pattern with the =El Pat Def instruction, which will return true if the pattern matches, and false if it doesn't.

To free the memory containing the compiled pattern when it is no longer needed, call ElPat Free.

Notes

- You can call El Pat Set Case / El Pat Set Nocase again to change the pattern to match against without calling ElPat Free first.

- You can use the
Pat Case & ElPat Nocase
commands for simple pattern

```
    matching while a default pattern is defined, without overwriting the
    default pattern.

  - ElPat Free is also
                Called Implicilty
                 by other
    AMOS commands.


Errors

    Illegal Function Call

    ElPat Set Case & ElPat Set Nocase will produce this error if the
    pattern is not
                Correctly Defined
                .


    No Default Pattern Defined

    You have called = ElPat Def, without first defining a default pattern
    with ElPat Case or ElPat Nocase. This also occurs if you try to use a
    default pattern after freeing it.
```

1.67 Patterns / Testing for patterns

```
Command Syntax

  = ElPat Test( S$ )


Description

   This funciton returns True if the string S$ contains any special pattern
   matching control characters, or False if it doesn't. It can be used to
   decide whether to compare that pattern with pattern matching, or with
   the much faster AMOS string comparison.
```

1.68 Patterns / Optimising Patterns

```
Command Syntax

  = ElPat Remove( P$ )


Description

    This funciton removes all unecessary pattern matching characters from
```

```
    the string P$. A common test is:

  ’
  ’P$ is an arbitary pattern
  ’
  P$ = ElPat Remove(P$)
  If [6\5\ElPat Test](P$)
     ’
     ’ Code to operate on P$ as a pattern
     ’
  Else
     ’
     ’ Code to operate on P$ as a string
     ’
  End If
```

1.69 Patterns / Escaping A String

```
Command Syntax

  = ElPat Escape( S$ )
```

```
Description

    This function escapes all the special pattern characters in S$, by
    preceeding them with an ’. It is usefull where a string input by the
    user is to become part of a pattern - by escaping that string first,
    you prevent it from changing the pattern behaviour - E.g.
```

```
  Input "Enter Sub-String to find :";S$
  ElPat Set Nocase "*"+ElPat Escape(S$)+"*"
  ’
  For A = 1 To ARRAYSIZE
     If ElPat Def(ARRAY$(A)) Then Print ARRAY$(A)
  Next A
```

```
    This code fragment finds all occurances of a sub-string in an array,
    without being case-sensitive. ElPatEscape is used to ensure that all
    user inputs are valid - without it, the substring "abc(de" would cause
    an illegal function call as "*abc(de*" is not a valid pattern because
    it has an unescaped unpaired parenthesis, whereas "*abc’(de*" is
    valid.
```

1.70 AmigaDos / Intuiton Commands - Contents

```
                                        AmigaDos / Intuiton Commands

Section 1: AmigaDos Commands
```

1.71 AmigaDos / File Protection Bits

Command Syntax

```
  = ElProtect( FILENAME$ )
  Els Protect FILENAME$ , BITS
```

Description

Each AmigaDos file or directory has a set of 8 protection bits, which are usually set to '----rwed'. The full set is 'hsparwed'.

h = Hidden file/dir.

When a file/dir has it's h bit set, it is not listed in many directory listing & file selectors - well thats the theory anyway - I've never come across anything that takes any notice of it!

s = script file.

When a file has it's s bit set, it means it is a script file. Under KickStart 2/3 this means you can execute the script as if it was a program by typing it's filename into the CLI/Shell.

p = pure file.

A file's pure bit is used by the AmigaDos resident comand to indicate that it can be made resident safely. Pure means that it is re-execuatable (If you loaded the code into memory, executed it, then executed the same copy again, it would still work the second time), and re-enterant (If you loaded the code into memory, then executed it twice as two processes simultaneously, both would work).

a = archived file / directory.

In general, whenever a program writes to a file, it’s archive bit is cleared. Archivers such as lha, and harddisk backup programs can be made to set the archive bit of the files they make a backup of. Then at a later date, when you come to update your backup , you can make the archiver ignore all files with the archive bit set, as you know that they have not been changed since the last time you made a backup.

r = readable file.

If a file does not have it’s readable bit set, many programs will refuse to read from / load it.

w = writeable file.

If a file does not have it’s writeable bit set, many programs will refuse to overwrite it, or append to it.

e = executable file.

If a file does not have it’s executable bit set, it cannot be run as a program.

d = deletable file.

If a file does not have it’s deletable bit set, you cannot delete it.

The =ElProtect function call returns the protection bits of the given file or directory name as an integer. Els Protect allows you to overwrite the bits of the given filename with a new set. The integer returned from ElProtect, and passed to Els Protect, has it’s lower 8 bits corresponding to the 8 protection bits.

For the lower 4 bits, a value of 0 means on, and 1 off, but for the upper 4 bits, 0 is off, and 1 is not. This means that the default flags "----rwed" have a value of 0.

Bit	Meaning Clear (0)	Meaning Set (1)
0	Can be deleted	Can’t be deleted
1	Can be executed	Can’t be executed
2	Can be read from	Can’t be read from
3	Can be writen to	Can’t be writen to
4	File not archived	File is archived
5	File not pure	File Pure
6	File not a DOS script	File is a DOS script
7	File visible in lists	File hidden is lists

Notes

- You should not set any of the upper 24 bits of the integer passed to Elsprotect.

- Bits s & p should not be set on directories.

- Not all programs abide by the protection bits - they are often ignored.

Errors

Any I/O error such as 'File Not Found', 'Device Not Available' etc. may occur if the filename given is not valid, or there the disk is corrupted.

1.72 AmigaDos / Console Output

Command Syntax

```
=Elout Exists
Elout STRING$
```

Description

The Elout Exists function will return True (-1) when there is a standard output console window availble to write to, and False when there isn't.

The Elout command sends the string to the standard output. It may contain AmigaDOS formating codes:

AscII characters 32-127 and 160-255 are sent directly. The following ANSI control sequences are also accepted. Hex values refer to AscII codes - e.g. in AMOS $9B $40 is Chr$($9B)+Chr$($40).

NOTE: Wherever $9B occurs, you can either send $9B, or $1B followed by a '[' character.

NOTE: N refers to a ascII number - i.e. A string of digits. [] around a term means it is an optional parameter and may be omited.

Value	Action
$8	Backspace
$A	Line Feed (See Below)
$B	Move Cursor Up One Line
$C	Clear Console Window
$D	Carriage Return (Without Line Feed)

```
$E    Shift Mode Off

$F    Shift Mode On. If Shift Mode is On, the Most
    significant bit of each character is set. I.E. Each
    character is bitwise or'd with $80.

$1B $63   Reset Console to its initial state.

$9B [N] $40 Insert N spaces at cursor position (Defaults to 1)
$9B [N] $41 Move cursor up N lines (Default 1)
$9B [N] $42 Move cursor down N lines (Default 1)
$9B [N] $43 Move cursor right N positions (Default 1)
$9B [N] $44 Move cursor left N positions (Default 1)
$9B [N] $45 Move cursor down N lines, and to column 1 (Default 1)
$9B [N] $46 Move cursor up N lines, and to column 1 (Defualt 1)

$9B [Y] [$3B X] $48
    Move cursor to co-ordinates X,Y - Although both
    are optional, at least one of X and Y must be
    specified. Note that the $3B (;) always preceeds
    X even if there is no Y.

$9B $4A   Clear from current cursor position to the end of
    the window

$9B $4B   Clear from current cursor posiiton to the end of
    the line

$9B $4C   Insert a new line above the line containing the
    cursor. This shifts all lines below the current
    line down one line.

$9B $4D   Remove the line containing the cursor. This shifts
    all lines below up one line, and clears the bottom
    line.

$9B [N] $50 Delete N characters to the right of the cursor,
    including the one under the cursor (Default 1).

$9B [N] $53 Scroll display up N lines (Default 1).

$9B [N] $54 Scroll display down N lines (Default 1).

$9B $32 $30 $68 Set PC Line feed mode (See below)

$9B $32 $30 $6C Set Amiga Line feed mode (See below)
```

The line feed code ($A) usually only sends a line feed. However, if the PC line feed mode sequence has been sent, a carriage return is also sent along with each line feed. Amiga line feed mode disables this again.

You can also send ANSI text colour / style change commands. These are of arbitary length, but always start with character $9B, and end with character $6D. In between are strings of AscII digits seperated by semi-colons. The digits strings have the following meanings:

```
        0       Plain text
        1       Bold Face
        3       Italics
        4       Underscore
        7       Inverse video

        30 - 37 Change foreground colour to N-30
        40 - 47 Chagge background colour to N-40

   E.g. $9B "31;43;1m" sets pen 1, paper 3, in boldface. (m=$6D)


   You can also include the following character sequences in the output
   string. These are not ANSI sequences - they only apply to the amiga
   console device:

  $9B N $74 Set Number of lines to N. NOTE: This doesn't
      change the window size, just allows you to use
      only part of it.

  $9B N $75 Set Number of columns to N. This also doesn't
      affect the window size.

  $9B N $78 Set the offset from the left of the window
      to the first column to N pixels.

  $9B N $79 Set the offset from the top of the window
      to the first row to N pixels.

  $9B $30 $20 $70 Make the cursor invisible
  $9B $20 $70 Make the cursor visible
```

1.73 AmigaDOS / Console Input

```
Command Syntax

  =Elin Exists
  =Elin$( NUMCHARS )
  =Elin Get$



Description

    The Elin Exists function returns True (-1) when there is an active
    standard input available for AMOS to read from. Elin$ will read up
    to NUMCHARS characters from the standard input, but will stop reading
    at the first linefeed character. (I.E. Where the user pressed return).

    Elin Get$ will return a single character from the standard input, or
    "" if nothing has been typed

Notes
```

- NUMCHARS should be 256 for normal Console user input.

- Elin$ will wait until enough characters, or a linefeed is read, suspending AMOS for the intervening period. Elin Get$ always returns immediately.

- Make sure that the Amos To Back command has been used when waiting for interactive input, or AMOS will freeze.

1.74 AmigaDos / Executing Programs

Command Syntax

=Elexec(FILENAME$)

Description

This function is similar to the AMOSPro Exec function except it executes the specified file and passes it AMOS's standard input, and standard output, and it also gives the AmigaDos return code of the program as a result

Notes

- Standard Result Codes are:

  0 = No Error

  5 = Warning

  10 = Error

  20 = Fail

  But not all programs use them.

- When STDIN or STDOUT don't exist, the program is passed a NIL: input or output.

Errors

Standard AmigaDos error return codes for File Not Found, Out of Memory etc. may be returned in the Return Code.

1.75 Intuition / The Workbench

Command Syntax

```
= Elwb Close
= Elwb Open
= Elwb Test
```

Description

AMOS provides a close workbench command, but it does not tell you whether the workbench did actually close or not. =Elwb Close closes the workbench, and returns True if it was successful.

=Elwb Open can be used to reopen a closed workbench screen. If the workbench program was running on the screen (I.E. You had called LoadWb), it will be restarted. It too returns a boolean to indicate if it was sucessfull.

=Elwb Test simply returns True if the workbench is currently open.

Notes

- Elwb close returns true if the workbench is closed when the function has finished executing, even if it didn't close it because it was already closed.

- Similarly Elwb open returns true if the workbench is open when it finishes executing, even if it was already open.

- Elwb Close and Elwb Test have the side effect of bringing the workbench screen to the front of all other intuition screens (But not in front of AMOS Screens). This means:

  o If Elwb Close fails, the workbench screen will be put in front of other screens.

  o After an Elwb Test, the workbench screen is brought to the front if it is open. You may like to call Elwb Test simply for this purpose after an Elwb Open.

1.76 Intuition / Iconifying AMOS

Command Syntax

```
=ElIconify Amos( X , Y , TITLE$ )

=ElIconify Begin( X , Y , TITLE$ )
=ElIconify Test
ElIconify End
```

Description

ElIconify AMOS opens the workbench screen, and opens a small window on it at co-ordinates (X,Y) with the name TITLE$. It has a close window gadget, depth gadget(s), and is moveable.

If you activate the window, then press the right mouse button, or you press the close window gadget, the window is closed, and one of the following values is returned:

-1 = The close window gadget was pressed.

0 = Then right mouse button was pressed with the window active.

1 = Couldn’t open workbench screen (There was not enough free chipmem).

2 = Couldn’t open window (Usually means that It wouldn’t fit on the screen at ↩ the
given co-ordinates).

ElIconify AMOS suspends your AMOS program until the user de-iconifies it. You may keep your program running while iconified by using the other three functions instead.

=ElIconify Begin takes the same arguments are ElIconify AMOS, but returns immediately. The values returned are:

0 = Window opened successfully,

1 = Couldn’t open workbench screen.

2 = Couldn’t open window.

=ElIconify Test also returns immediately, and if returns the following values:

-1 = The user has pressed the right mouse button in
the window title bar since the last call.

0 = The user has done nothing since the last call.

1 = The user has pressed the close window gadget
since the last call.

ElIconify End closes the window opened by ElIconify Begin.

Notes

- Since ElIconfiy AMOS suspends your program, you should always use the following procedure:

```
Procedure ICONIFY[X,Y,TITLE$]
    Amos Lock
    Amos To Back
    A=Iconify Amos[X,Y,TITLE$]
    Amos To Front
    Amos Unlock
End Proc[A]
```

The Lock/Unlock disable Left-Amiga-A switching, as the program is frozen while the window is displayed. An alternative method is to lock amos at the start of your program, and leave it locked, and remove the lock / unlock commands from this procedure.

If possible you should close your AMOS screens when iconifing, and free other resources as this saves some memory, which is the whole point of iconification.

Errors

Illegal Function Call

You called =ElIconfiy Test, or ElIconify End without first creating an iconified window with =ElIconfiy Begin

Other error conditions are returned as result values.

1.77 Miscellanous Commands - Contents

Miscellaneous Commands & Functions

1.78 Misc / Reading AMOS Internal Data

Command Syntax

```
= Elbase NUM
```

Description

AMOS has a rather large internal datazone, which is documented in the |lequ.s file in the extensions drawer on AMOSPro_Tutorial:. (The AMOS Creator version is in Extras:Extensions/"). If you call Elbase with a parameter of 0, the address of this data structure is returned.

NOTE: For extension programmers, this is the value of register A5.

Also, each AMOS extension can have its own data zone. To obtain the address of any extensions data zone, pass the extension number as the parameter.

Notes

- It goes without saying that the contents of the data zones will change with different versions of AMOS / The extensions, but this can be useful for debugging extensions (I should know :-)
- Negative valus of Elbase may return various values I use to debug easylife, or for Easylife support programs to communicate with the extension. Do not use these.
- A 0 will be returned for non-existant extensions, or extensions without any data.

1.79 Misc / Waiting On The Raster Beam Position

Command Syntax

```
ElRaster Wait SCANLINE
```

Description

This is an enhancement to Wait Vbl - It waits for the raster beam to reach the given Scanline. Easylife checks all 9 bits of the scan position, so you can wait on values >256.

Notes

- The scanline check is done with a busy wait loop, so it eats processor time - don't use it when multi-tasking!

1.80 Misc / Detecting Your Runtime Envoironment

Command Syntax

= ElPro
= ElCompiled

Description

=ElPro returns true when your program is being run from AMOS Pro, or if it was compiled from the AMOS Pro compiler. It returns False if it was run from AMOS Creator, or was compiled with the AMOS Creator compiler.

=ElCompiled returns true if your program is running as a stand-alone program, and false when it is being run under AMOS.

Notes

- If your program is compiled in AMOS mode - I.E. to be loaded into the AMOS interpreter when compiled and run from there, ElCompiled will return false. ElCompiled will only return True if it is being run as a stand-alone program outside the AMOS envoironment.

- The compiler extenion has an equivilent instruction, but it is not a lot of use to people who haven't bought the compiler!

1.81 Misc / Reseting AMOS Extensions

Command Syntax

El Reset NUM

Description

This command will make extension number NUM think that the AMOS 'Default' command has been called, and the extension will reset itself. However the default command is not called, so the screen etc. is not reset.

Notes

- To reset easylife use 'Elreset 16'. The effects on easylife will be

```
    the same as the effects of the
                Default
                 command.
```

1.82 Misc / Overlaping Rectangles

```
Command Syntax

  =El Overlap( A1 , B1 , A2 , B2 To X1 , Y1 , X2, Y2 )
  =El Lapsx
  =El Lapsy
  =El Lapex
  =El Lapey



Description

    These functions returns the co-ordinates of the portion of two rectangles
    which overlap. The two rectangles are described as follows:

    A1 , B1  : The co-ordinates of the top-left hand corner of rectangle 1
    A2 , B2  : The co-ordinates of the bottom-right had corner of rectangle 1

    X1 , Y1  : The co-ordinates of the top-left hand corner of rectangle 2
    X2 , Y2  : The co-ordinates of the bottom-right had corner of rectangle 2

    Eloverlap returns True if the two rectangle overlap, and false if they
    do not. If they do overlap, the other 4 functions will return the
    co-ordinates of the rectangle which describes the overlaping area.

Notes

  - If Eloverlap returns False, the values of the other functions are
    undefined.

  - This may seem fairly useless, but try writting a program that creates
    windows with depth gadgets on an AMOS screen :-)
```

1.83 Misc / Bank Existance Checking

```
Command Syntax

  =Elbnk Here ( BNKNO )



Description

    This function will return True (-1) if the specified bank has been
    reserved for the current program, or False (0) if it has not.
```

1.84 Misc / File Existance Checking

```
Command Syntax

  =Elexists ( FILENAME$ )



Description

    This is similar to the AMOS exists function, except:

    - If the file does not exist, a 'Please Insert Volume...'
      requester will appear.

    - If it returns 0, the file did not exist.

    - If it returns a negative number, the file did exist.

    - If it returns a positive number, then this is the name of an
      existing directory, not a file.



Errors

    Various Disk I/O Errors may occur.
```

1.85 Index - By Subject

```
                                            Index - By Subject

This is a complete index of all subjects covered in this Easylife
manual. There is also an index to all
                Commands & Functions
                .

AMOS Commands:

                Default

                ReserveZone

                AMOS Internal Datazone
                AMOS Mailing List
AMOS World-Wide-Web Site

                ANSI Control Strings
                Banks:

                Loading XPK Compressed

                Reading Name
```

1.86 Index - By Commands & Functions

Command & Funcion Index

This is a complete index of all Easylife Commands & Functions, along with AMOS command whose behaviour has been modified. These is also an index to all
Subjects
covered in this manual.

Easylife Commands:

```
=Elf Last Not Asc

=Elf Last Not Char

=Elf Not Asc

=Elf Not Char

=Elf Nth Asc

=Elf Nth Char

=Elf Num Asc

=Elf Num Char

=Eliconify Amos

=Eliconify Begin

 Eliconify End

=Eliconify Test

=Elin$

=Elin Exists

=Elin Get$

=El lapex

=El lapey

=El lapsx

=El lapsy

 Ellchg

 Ellclr

=Ellock Font

=Ellong

=Ellong$

 Ellset

=Elltst

 Elmem

=Elmem Inc

=Elmem$
```

```
=Elmessage Exists
=Elmessage$
 Elmz Erase
 Elmz Reserve
 Elmz Set
=Elmzone
=Elmzoneg
=Elmzonen
=Elmznex
=Elmzney
=Elmznsx
=Elmznsy
=Elopen Font
 Elout
=Elout Exists
=Eloverlap
=Elpad Asc
=Elpad Char
=Elpat Case
=Elpat Def
=Elpat Escape
 Elpat Free
=Elpat Nocase
=Elpat Remove
 Elpat Set Case
 Elpat Set Nocase
=Elpat Test
=Elpp Allocate
```

```
=Elpp Buf
=Elpp Crunch
 Elpp Free
 Elpp Free All
 Elpp Keep Off
 Elpp Keep On
 Elpp Load
=Elpp Len
=Elpro
=Elprotect
 Elraster Wait
 Elreset
 Els protect
 Els Bank Name
 Elset Font
 Elunlock Fonts
=Elwb Close
=Elwb Open
=Elwb Test
 Elwchg
 Elwclr
=Elword
=Elword$
 Elwset
=Elwtst
 Elxpk Bload
 Elxpk Bsave
=Elxpk Error
 Elxpk Load
```

```
                =Elxpk Lof

                 Elxpk Save

                 ElZb Add

                 ElZb Multi Add
                     Installing Single Groups

                 ElZb Muiti Add
                     Installing Whole Banks

                 ElZn Shift

                =ElZnex

                =ElZney

                =ElZnsx

                =ElZnsy
                MUI Commands / Functions

 Mui Add
=Mui App
=Mui Application
 Mui Begin
 Mui Dispose
 Mui Do
 Mui Flush
=Mui Fn
=Mui Get
=Mui Get$
=Mui Hook
=Mui Input
=Mui Make Button
=Mui Make PopButton
=Mui New
 Mui Notify
 Mui Remove
=Mui Request
 Mui Set
 Mui Set Str

=Tag
=Tag$
=Tag$                        Integer Form
=Tag Attach$
 Tag Block Size
 Tag Keep
=Tag list$
=Tag Str$


Structured Variables Commands & Functions
```

```
=St Cmp
 St Copy
=St Dup
 St Free
 St Free All
=St Get
=St Get$
 St Input
=St Len
=St Load
=St New
=St Output
 St Save
 St Set
 St Set Str
=St Type


AMOS Commands


                 Default

                 Reserve Zone
```

1.87 Changes to the behaviour of other AMOS commands

```
                                    Effects Of Easylife On AMOS Commands

  - The
                Reserve Zone
                 command will erase multi zones.

  - The Default command (And
                Elreset 16
                ) will have the following effects
    on easylife:

    o
                All powerpacker buffers are deallocated
                .

    o The
                Default Pattern
                 is removed.

    o The MUI Application object & all MUI Root objects are Disposed of

    o All Strings Stored with MUI objcets are disposed of, and the
      Memory Blocks are also deallocated.

    o All fonts are
                unlocked
                    o Elf Fail Start is called.
```

o
Elpp Keep Off
is called.

o Any libraries Easylife has open are closed.

o St Free All link EasylifeSTRUCT.guide/main} is called.

These are actions are also performed when you run a program from the AMOS / AMOSPro editor, and when you quit AMOS / AMOSPro, either via the Editor's Quit opition, the system command, or a compiled program quitting / crashing.

They are not performed when you Prun a program, or run an accessory.

1.88 To-Do List

Todo

- Make Elxpk Load & Save work with sprite / icon banks.

- Write Elxpk First$() & Elxpk Next$ to get packer information.

- Write XPK versions of Open, Close, Input$, Print# & Sload.