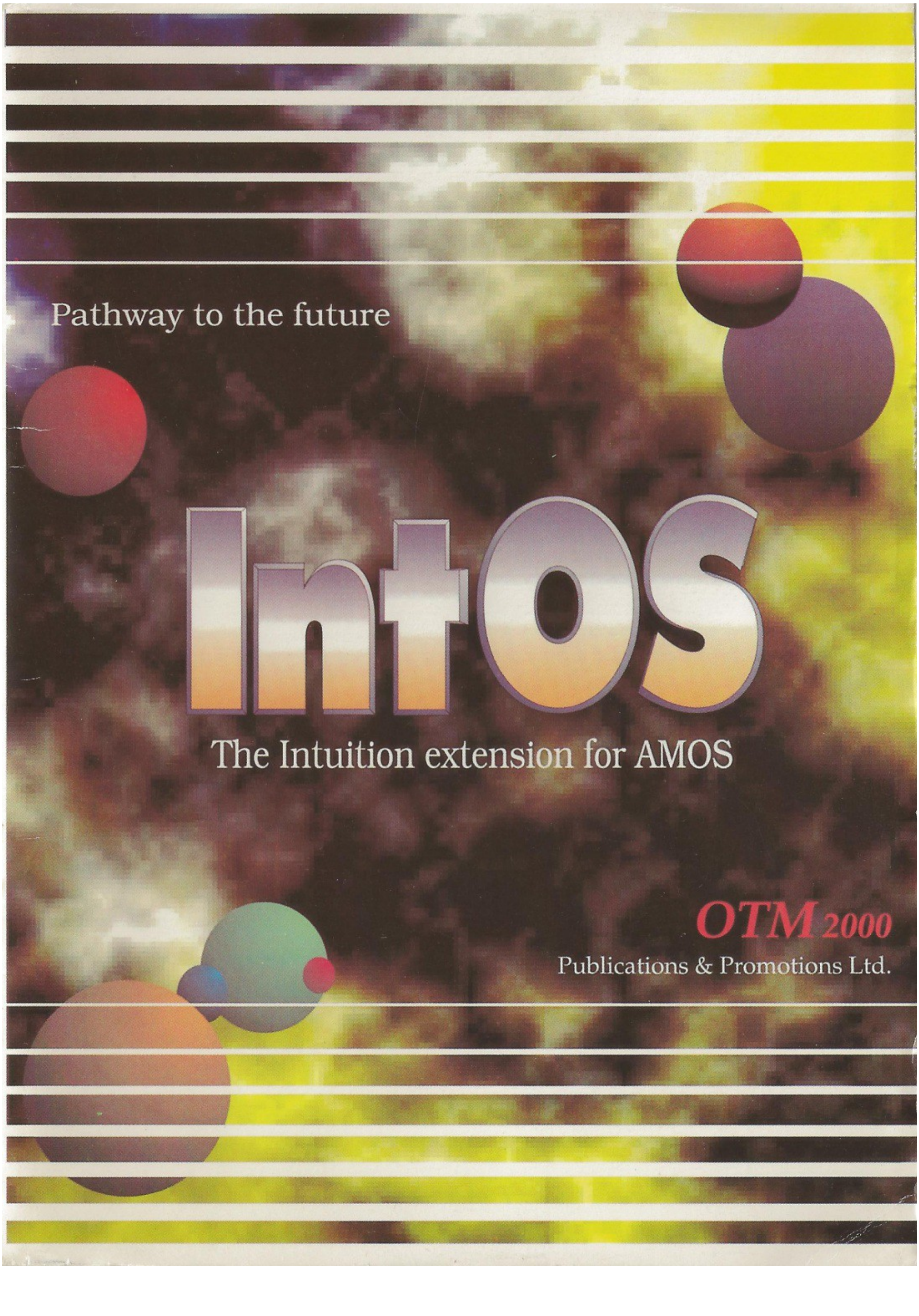
Pathway to the future
IntOS
The Intuition extension for AMOS
OTM 2000
Publications & Promotions Ltd.

IntOS

The AMOS Intuition Extension

by

Matthew R.Warren

Published by OTM Publication & promotions Ltd

In the event of any fault in any software package, goods should be returned to OTM Publication & promotions Ltd where they will be checked and a replacement given where found necessary.

Customer Services,
OTM Publication & promotions Ltd.
5 Albert Road
Tamworth
Staffordshire
B79 7JN

WELCOME - to OTM Publication & promotions Ltd.®
THANK YOU FOR YOUR CUSTOM

Credits and General

The Intuition Operating System (IntOS) v1.01

Development Version 4.22(Turbo)

Special thanks to: Nigel Cockayne for his kind support and help with the examples.

We are proud to present the Intuition Operating System (IntOS) v1.01.

IntOS consists of a customised Library file called 'IntOS.CustLib' and over 120 command procedures, which will allow you, the programmer, to fully manipulate the Amiga's Intuition system using AMOS.

This means that you can; Open true intuition screens and windows with gadgets and menus and fully control how they are to be used. Literally, the imagination is your **ONLY** limitation. IntOS opens up a whole new world of applications and utility software development.

1. Make a BACK UP of your AMOS work disk

2. Boot up the IntOS program disk.

3. If you only have a single drive, go to 4

If you have two or more drives, go to 5

If you have a hard drive, go to 6

4. Type the following...

Copy SYS:Libs/IntOS.CustLib To Ram: All Quiet

Now insert your AMOS Disk Un-write protected.

Type the following…….

Delete DFO:Devs/Narrator.Device

The above is to create a little disk space... If you want to delete something else instead, then do so.

Copy Ram:#? to DFO:Libs/ All Quiet

The above replaces relevant libraries with the versions IntOS needs to function.

Now go to 7

5. Insert your AMOS work disk into the external drive

Type the following

Delete DF1:Devs/Narrator.Device

The above is to create a little disk space... If you want to delete something else instead, then do so.

Copy SYS:Libs/IntOS.CustLib to DF1:Libs/ All Quiet

The above replaces relevant libraries with the versions IntOS needs to function.

Now go to 7

6. You can now have all the routines on your hard disk!

Type the following...(??? = HardDisk device name)

Copy SYS:Libs/IntOS.CustLib to ???:Libs/ All Quiet

MakeDir ???:IntOS_Routines

Copy SYS:AMOS-Routines/#? to ???:IntOS_Routines/ All Quiet

or

Copy SYS:AMOSPro-Routines/#? to ???:IntOS_Routines/ All Quiet

7. That's it! You should now be ready to use IntOS!

Floppy disk users : the IntOS routines are still on the IntOS disk so you will need to load them off the IntOS disk.

Hard disk users : All routines are on your harddisk ready for loading into AMOS / AMOS Pro.

The IntOS Software consists of the following:

IntOS.CustLib	-	In LIBS: directory (needs to be on your AMOS disk)
AMOS routines	-	These allow you to use IntOS, versions for AMOS and AMOS professional and broken down into groups of procedures.
Examples	-	Example IntOS + AMOS programs

To be able to use IntOS, you must merge in some pre-prepared code. **But to make it easier for you,** we will go through it, step-by-step...

Step 1: Make sure that you have installed IntOS.CustLib to your AMOS disk in the LIBS directory or onto your harddisks LIBS directory.

Step 2: Boot up AMOS / AMOS Pro

Step 3: Now reset your Text Buffer to around 64,000 kbytes to begin with. (This can be altered at a later date)

Step 4: To start with, load in 'IntOS_All_Procs.AMOS'.
This file contains ALL the AMOS side of IntOS. These procedures constitute the entire IntOS command set. Their only job in life is to feed information to the IntOS.CustLib

Step 5: Okay, now go to the line saying...

```
'Rem >> **** - YOUR CODE HERE!! - **** <<
```

Delete the entire line.
Type in the following... (comments are optional)

```
' Get the Workbench as screen '0'
IN_WB_TO_SCREEN_[O]
'
' Open a simple Window
IN_WINDOW_[0,160,12,320,188,$100E,"Hello",-1]
'
' Print a little message
```

```
IN_RPRINT_["Press Left Mouse Button (LMB)"]
'
' Wait for mouse press
IN_WAIT_RAT
'
```

Now SAVE it!! Just in case, you never can tell!

Done that? Okay then, **Run it... and wow!**

NOTE:- AmosPro Users with under 2meg memory... It is suggested that you use the 'Kill Editor' command before you run IntOS... Just to add to the memory pool.

Let us explain what its doing...

First of all it is VERY important that you define a screen for output, it is impossible to open a window, without it knowing where it should be opened. What 'IN_WB_TO_SCREEN' has done, is get the Workbench screen for your use, as screen 0 to be exact.

Then we told IntOS to open a window. We opened Window 0 at the co-ordinates 160 by 12. The window itself was 320 wide by 188 high and was titled 'Hello'. The '-1' value basically tells IntOS that there are no Gadgets (buttons) available for the window to use. The '$100E' value is more complicated, and is discussed in full under the reference section.

Then we told IntOS to print a message to the window along with a 'Return Character'. It should be explained that BASIC programmers are shielded mostly from the way in which the computer works, so all that needs to be said is, If you want to create a new line after printing a message, then please use 'IN_RPRINT', else please use 'IN_PRINT'.

We then instructed IntOS to wait until a mouse button was pressed. This command halts ALL program execution until a button has been pressed

Simple ? But, what about all of those redundant procedures?
Well, as they're not being used, you may as well rip them out!

DO NOT delete the _INTOS_INIT: routine or the 'IntOS_Procs' though!

The IntOS procedures are split into the general tasks they do...

'IntOS_General.AMOS'	-	General and miscellaneous commands.
'IntOS_Screen.AMOS'	-	**ALL** the Screen definition and manipulation commands.
'IntOS_Screen_2D.AMOS'	-	**ALL** the 2D screen drawing commands.
'IntOS_Window.AMOS'	-	**ALL** the window definition and manipulation commands.

The following are **ALL** dependent on the 'IntOS_Window.AMOS' file, as you can only really use them in conjunction with a window.

'IntOS_Window_Event.AMOS'	-	ALL the AmigaDOS event handling commands.
'IntOS_Window_2D.AMOS'	-	ALL the Window 2D drawing commands.
'IntOS_Gadget.AMOS'	-	ALL the gadget (button) definition and manipulation commands.
'IntOS_Menu.AMOS'	-	ALL the Menu definition and manipulation commands.

The following files are compulsory when using IntOS...

'IntOS_Init.AMOS'	-	This Sub routine initiates IntOS.
'IntOS_Procs.AMOS'	-	All the nitty-gritty control commands.

NOTE:- The **ONLY** procedure deletable in the IntOS Procs. is the 'IN_TURBO' command.

All of these procedures are split into 2 categories. Statements and Functions. With Statements, you just issue the command with any parameters it might need... and... that's it. Functions are slightly different.

Functions are commands, but they return a result. So, for instance, If you want to ascertain the width of a screen, you would issue the '=Screen Width' function. Its as simple as that; or is it? Well, for those experienced Amos programmers out there, you will know that Amos's way of handling custom functions leaves something to be desired! Its quite messy...

To get the width of a window in IntOS you will have to do this...

```
' Issue the Function
IN_W_WIDTH
'
' Get its result
WIDTH=Param
```

This means that you will have to issue a function as if it were a statement, and use the '=Param' function to get its result. Sorry for that small inconvenience, but you will have to blame Amos's author for that!

This command is included because its faster than normal if you are issuing IntOS statements one after the other, giving between 5% - 70% speed increases, depending on the command you are telling it to issue.

For instance, Opening a window is the same speed under IN_WINDOW as it is under 'Window{;}'(The TURBO version) because IntOS is left waiting for AmigaDOS to finish opening the window. Most commands however, do show a noticeable speed increase.

The Turbo command works by first creating a string full of commands, and then issuing the IN_TURBO. Its very similar to the way AMAL or Interface works. The differences are as follows...

1. Only one script can be run at once.

2. Unlike AMAL it doesn't multitask with your program.

3. Its command syntax's are different.

4. **NO** functions can be issued (as they aren't supported by the TURBO command)

5. **NO** command which involves disk activity can be used.

6. ALL numeric parameters must be absolute values! NO fractions, no mathematical sums or functions... they MUST be the value you want to use! i.e. '2+2' will **NOT** work, it MUST be issued as '4'!

7. ALL numeric values must be in Decimal form ONLY!! HEX and BINARY numbers will be considered as the number '0'!!

8. String parameters do **NOT** have to be issued between speech marks... They aren't needed.

9. An 'End{;}' statement **MUST** be given to 'end' the script.

The TURBO command may be a bit hard to grasp at first, but you will find it very useful if its used properly. Now, because the TURBO command is optimised for speed, you will have to follow some fairly strict layout rules, because there is very little error checking!

NOTE:- The parameter layout is the same as the normal equivalent!

Before every command an '@' must be used. There must be **NO** spaces between the '@' and the command, i.e. "@**PRINT{hello;}"**

First you issue the command. Then you have to issue the parameters. These must be placed within a pair of curly brackets '{}' immediately after the command. There **MUST** be **NO** spaces between the command and the parameter brackets!!! Now, after **EVERY** parameter, there must be a Semi-Colon';'

For EXAMPLE...

```
T$=T$+"@WB_To_Screen{0;}"
T$=T$+"@Window{0;160;12;320;188;4110;Hello;-1;}"
T$=T$+"@R_Print{Hello folks!;}"
T$=T$+"@End{;}"
```

Its best to keep one command per line. Its easier to read.

IN_WAIT_RAT

command: wait for mouse input

syntax: IN_WAIT_RAT

Halts all program flow until a mouse input in made.

Please note that this command is only a temporary command while creating your program and **SHOULD NOT** be used in the final version. This is because it is supplied only as a debugging aid. This command really mess's with Multitasking.

IN_PRINT

command: renders text to a screen/window

syntax: IN_PRINT_[$]

turbo: @Print{$;}

This command is used to render text to screen. It should be noted that unlike the normal AMOS Print command, this command will ONLY except text as its input.

You can however use normal +,- operators...

i.e. IN_PRINT_["Hello"+" "+"Folks!"]

Performing maths with this command is a bit awkward. You have to perform every numerical operation within the =Str$() function.

i.e. IN_PRINT_["Number = "+Str$(AGE+DOB)]

There is a small problem with AMOS's =Str$() function. It insists on adding a space character (ASCII=32) onto the front of its result. This as usual can be tackled with:

(Str$(AGE+DOB)-" ")

Also See: =Str$()

IN_RPRINT

IN_RPRINT

command: renders text to a screen/window and creates a new line

syntax: IN_RPRINT_[$]

turbo: @R_Print{$;}

Uses exactly the same syntax as IN_PRINT_[] except that a new line is created. Please note that this new line is **NOT** created using the return or linefeed characters. Don't ask why, it just doesn't.

Also See: IN_PRINT

IN_FSEL

function: opens up a file requester on the current screen

syntax: IN_FSEL_[X , Y , W , H , T$, P$, F$, PT$]

result: $=Param$

This command opens up either the WB2.0+ standard ASL file requester, or a custom temporary file requester.

The computer automatically makes the choice for you. If your machine has WB2.0+ and has the ASL.Library on the LIBS directory of your SYS disk.

Then the standard WB2.0+ requester will be used. However. If don't have the ASL.library on your SYS disk, then IntOS will open another 'lesser' requester for your use. It should be noted that with this 'lesser' requester, only parameters 5, 6 and 7 are used.

1. X - X positioning of the requester
2. Y - Y positioning of the requester
3. W - Width of requester
4. H - Height of requester
5. T$ - Title
6. P$ - Path
7. F$ - File
8. PT$ - Pattern for requester (normally use's the '#?' wildcards)

i.e. IN_FSEL_[160,12,320,176,"Load","SYS:","","#?"]
IN_RPRINT_[Param$]

Also See: =Param$

IN_RPRINT

IN_NTSC

function: used to check whether or not the programs running under NTSC

syntax: IN_NTSC

result: bool=Param

This function is used to check which display system you are using. The result given is a boolean value (True/False).

NTSC runs at 60 hertz while PAL and SECAM run at 50 hertz.

Also See: IN_DISP_HEIGHT

IN_DISP_HEIGHT

function: to gain your display's verticle height in pixels

syntax: IN_DISP_HEIGHT

result: ubyte=Param

This command is usually used in conjunction with _NTSC. It is used to gain the height of your display.

NTSC should return a value of 200

PAL and SECAM should return a value of 256

Also See: IN_NTSC

_REPLACE

function: to replace characters within a string

syntax: _REPLACE_[S$, F$, R$]

result: $=Param$

This command is to supplement an omission from AMOS's command set. This you must agree has been the bane of many a programmers life.

This command replaces characters within a string with new ones.

S$ - Source String

F$ - Find String

R$ - Replace String

```
i.e.    T$="Matthew Was Ere."
'
IN_REPLACE_[T$,"Matthew","Nigel"]
T$=Param$
'
IN_RPRINT_[T$]
```

The result from the above example should convert 'Matthew Was Ere' to 'Nigel Was Ere'. This function has more practical uses though.

Also See: =Param$

IN_RPRINT

IN_WB_TO_FRONT

command: sends the workbench screen to the front of the display

syntax: IN_WB_TO_FRONT

turbo: @WB_To_Front{;}

This command should only really be used when after an 'Amos To Back' command has been issued.

This command will move the WorkBench screen to the front of the display.

Also See: IN_WB_TO_BACK

IN_WB_TO_BACK

command: sends the workbench screen to the back of the display

syntax: IN_WB_TO_BACK

turbo: @Wb_To_Back{;}

This command should only really be used after an 'Amos To Back' command has been issued.

This command will move the WorkBench screen to the back of the display.

Also See: IN_WB_TO_FRONT

IN_WB_TO_SCREEN

command: makes the workbench a program screen

syntax: IN_WB_TO_SCREEN_[Num]

turbo: @WB_To_Screen{Num;}

This command is simple. It tells your program to refer to the Workbench screen as the specified screen number you assign it. This basically allows you to create Workbench programs.

i.e. IN_WB_TO_SCREEN_[0]

Will tell you program to refer to the Workbench screen as screen '0'. IntOS can allow screen numbers from 0-63! YES... 64 screens are available!

IN_Q_SCREEN

command: allows quick definitions of intuition screens

syntax: IN_Q_SCREEN_[Num , flags , title$]

turbo: @Q_Screen{Num;Flags;Title$;}

This command is the simplest way to create an Intuition screen. But this command is not very flexible. The screen number can be between 0 and 63.

The screen flags are as follows...

Screen Depth:	1	2 Colours	
	2	4 Colours	
	3	8 Colours	
	4	16 Colours	
	5	32 Colours	(Lowres ONLY!)
	6	64 Colours	(Lowres ONLY!)
	Hires:	8	
	Interlace:	16	

So, a lowres 32 colour interlaced screen would look like this...

IN_Q_SCREEN_[1,5+16,"Hello"]

or

"Q_Screen{1;21;Hello;}"

(Sorry, no HAM modes available at the moment.)

Also See: IN_SCREEN

IN_SCREEN

command: more detailed and flexible screen definition

syntax: IN_SCREEN_[num , Y , W , H , D , V , Title$]

turbo: @Screen{num;Y;W;H;D;V;Title$;}

The more detailed version. This command is much more flexible, but more complex. Here's how it goes...

First you give the screen number (0-63).

Then the Y-Pos (usually '0')

Then the Depth of the screen:

1	=	2	Colours	
2	=	4	Colours	
3	=	8	Colours	
4	=	16	Colours	
5	=	32	Colours	(LOWRES ONLY!)
6	=	64	Colours	(LOWRES ONLY!)
7	=	128	Colours	(AGA Only!)
8	=	256	Colours	(AGA Only!)

Then we have the Viewmodes:

0	$0	=	Lowres
4	$4	=	Interlace
32	$20	=	SuperHires (AGA ONLY! - use with $8000)
32768	$8000	=	Hires

Then the Title, which should always be in String format.

So here's an example...

IN_SCREEN_[1,44,640,512,4,$8000+$4,"Hello"]

or

Screen{1;44;640;512;4;32772;Hello;)

This opens a screen 640 by 512 in size as a Hires Interlaced screen.

Also See: IN_Q_SCREEN

IN_S_SHOW

command: brings an already opened screen to front of display

syntax: IN_S_SHOW_[num]

This command will 'show' or bring to the front of the display a specified screen.

IN_S_FIND

command: find the front most screen for use.

syntax: IN_S_FIND_[Num]

This allows you to grab the front most screen and give it a number. This basically can allow you to open a window on DPaints screen for instance.

IN_LOAD_IFF

command: to load and display an IFF picture

syntax: IN_LOAD/_IFF_[Num , path$, P]

This command will load and display an IFF picture into a screen. It is not necessary to define the screen beforehand. But please note that the loaded IFF will use the default palette.

Num = screen number (0-63)

path$ = the disk path of the picture

P = Palette number

i.e. IN_LOAD_IFF_[1,"SYS:Pics/Piccy",0]

IN_S_RAT_X

function: give the horizontal position of the mouse

syntax: IN_S_RAT_X

result: num=Param

This command is similar to the '=X Screen(X Mouse)' function in AMOS.

Also See: =Param$

=X Screen(X Mouse)

IN_S_RAT_Y

function: give the verticle position of the mouse

syntax: IN_S_RAT_Y

result: num=Param

This command is similar to the '=Y Screen(Y Mouse)' function in AMOS.

Also See: =Param$

=Y Screen(Y Mouse)

IN_VIEWPORT

function: Give the address of a specified screen

syntax: IN_VIEWPORT_[num]

result: address=Param

This command is of most use to people who insist on messing with the Amiga's system. It gives the address of the start of a specified screen.

IN_SCREEN_PENS

command: Change the 'pens' of a screen

syntax: IN_SCREEN_PENS_[AT , IT , H , S , AF , GF]

Screen_pens is used to configure 6 of the 10 default screen pens.

AT	=	Active Text
IT	=	Inactive Text
H	=	Highlight
S	=	Shadow
AF	=	Active Fill
GF	=	Gadget Fill

IN_S_BITMAP

command: to allow drawing to a screen

syntax: IN_S_BITMAP_[Scrn Num , Bitmap]

turbo: @S_Bitmap{Num;Bitmap;}

This command is to be issued before ANY 2D screen drawing is done. A failure to do so will invoke an error report.

A Bitmap is literally the raw Memory area of the screen, and unlike AMOS which draws straight onto screen, IntOS will draw straight to the memory area. This has a huge speed advantage.

The 'Scrn Num' MUST be an already opened screen. The 'Bitmap' is the bitmap number (0-63)

Also See: IN_BITMAP_OUTPUT

IN_CLS

IN_PLOT

IN_DRAW

IN_BOX

IN_BAR

IN_CIRCLE

IN_DISC

IN_SCROLL

IN_FILL

IN_P_LOAD

command: to load (but **NOT** display) a palette from an IFF picture

syntax: IN_P_LOAD_[P , path$]

This command will load a pictures palette into the specified PaletteNumber.

i.e.
```
IN_P_LOAD_[1,"SYS:Pics/Piccy"]
IN_P_USE_[1]
```

IN_P_USE

command: use a previously loaded or defined palette

syntax: IN_P_USE_[P]

turbo: @P_Use{P;}

This command will assign a specified palette to the current screen.

Also See: IN_P_LOAD

IN_P_RGB

IN_P_CLOSE

command: to erase a palette from memory

syntax: IN_P_CLOSE_[P]

turbo: @P_Close{P;}

This command will erase a previously loaded or created palette.

IN_P_RGB

command: used in creating a palette

syntax: IN_P_RGB_[P , C , R , G , B]

turbo: @ P_RGB_{P;C;R;G;B;}

This command is primarily used in Palette Creation, and will only be displayed after a '_USE_PALETTE' has been issued.

P	=	Palette Number (0-63)
C	=	Colour Index (0-31)
R	=	Red value
G	=	Green value
B	=	Blue value

Also See: IN_P_USE

IN_RGB

command: to directly alter the current screens palette

syntax: IN_RGB_[C, R , G , B]

This command is used to directly alter a screens palette. This is not a suggested way around palette definition, but was included anyway.

C	=	Colour register (0-31)
R	=	Red value
G	=	Green value
B	=	Blue value

IN_BITMAP_OUTPUT

command: to select a 2D direct drawing output

syntax: IN_BITMAP_OUTPUT_[Bitmap]

Also See: IN_S_BITMAP

IN_CLS

IN_PLOT

IN_DRAW

IN_BOX

IN_BAR

IN_CIRCLE

IN_DISC

IN_SCROLL

IN_FILL

IN_COLOUR

command: to change the colour text is rendered in.

syntax: IN-COLOUR_[Pen , Paper]

turbo: @Colour{Pen;Paper;}

This command does the same job basically as the Pen and Paper commands

Amos supplies. Just supply the colour index of the colours to use.

i.e IN_COLOUR_[1,0] is the same as... Pen 1
Paper 0

IN_LOCATE

command: used to position the text cursor

syntax: IN_LOCATE_[X pos , Y pos]

turbo: @Locate{X Pos;Y Pos;}

This command simply moves the text cursor in the same way as the standard Amos 'Locate' command. This command is usually used with _PRINT or _NPRINT.

Also See: IN_PRINT
IN_RPRINT

IN_CURS_X

function: give the x position of the cursor

syntax: IN_CURS_X

result: pos=Param

Gives the x position of the text cursor of a screen.

IN_CURS_Y

function: give the y position of the cursor

syntax: IN_CURS_Y

result: pos=Param

Gives the y position of the text cursor of a screen.

IN_S_CLOSE

command: close a screen

syntax: IN_S_CLOSE_[num]

turbo: @S_Close{Num;}

This will close a screen and free the memory it used.

IN_S_TO_BACK

command: to put the current screen to back of display

syntax: IN_S_TO_BACK_[0]

turbo: @S_To_Back{0;}

<u>'0' must be issued.</u> This is a minor 'bug' - we will fix. This will send the current screen to the back of the display.

Also See: IN_S_TO_FRONT

IN_S_TO_FRONT

command: to put current screen to front of the display

syntax: IN_S_TO_FRONT_[0]

turbo: @S_To_Front{0;}

<u>'0' must be issued.</u> This is a minor 'bug' - we will fix. This will send the current screen to the front of the display.

Also See: IN_S_TO_BACK

IN_CLS

command: To clear a bitmap

syntax: IN_CLS_[colour]

This command is used to clear a bitmap screen.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_PLOT

command: Plot a pixel point

syntax: IN_PLOT_[x , y , c]

turbo: @Plot{x;y;c;}

This command is very similar to the Amos Plot command. The difference is that the colour **MUST** be given and it is **NOT** an option.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_POINT

function: used to gain the colour index of a pixel

syntax: IN_POINT_[x , y]

result: c=Param

This command is similar to the Point function in AMOS.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_DRAW

command: draw a line to your bitmap

syntax: IN_DRAW_[x1 , y1 , x2 , y2 , c]

turbo: @Draw{x1;y1;x2;y2;c;}

This command will draw a single line to a bitmap. This command is similar to the AMOS 'Draw' command. But notice that you must give the colour it is to draw with.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_BOX

command: draw a rectangular box to a bitmap

syntax: IN_BOX_[x1 , y1 , x2 , y2 , c]

turbo: @Box{x1;y1;x2;y2;c;}

This command will draw a rectangular box to the current bitmap. This command is similar to the AMOS version except that the colour it uses **MUST** be issued.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_BAR

command: draw a block of colour to a bitmap

syntax: IN_BAR_[x1 , y1 , x2 , y2 , c]

turbo: @Bar{x1;y1;x2;y2;c;}

This command is similar to the AMOS version. Except the colour it is to use must be issued.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_CIRCLE

command: draw a circle to a bitmap

syntax: IN_CIRCLE_[x , y , r1 , r2 , c]

turbo: @Circle{x;y;r1;r2;c;}

This command is similar to the Amos 'Ellipse' command. Except that the colour it uses must be issued. To draw a circle, r2 should equal r1.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_DISC

command: drawing a filled ellipse to a bitmap

syntax: IN_DISC_[x , y , r1 , r2 , c]

turbo: @CircleF{x;y;r1;r2;c;}

This command is the same as the IntOS command 'Circle', except that result is a filled area, and not outlined.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_SCROLL

command: To scroll a bitmap area

syntax: IN_SCROLL_[x1 , y1 , w , h , x2 , y2]

This command can be used to create a rectangular scrolling area.

x1 and y2 are the positions the area is to be moved to.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_FILL

command: Fill an area of a bitmap with a colour

syntax: IN_FILL_[x , y , c]

This command will fill an area of a bitmap with a colour at the specified co-ordinates.

NOTE: This command operates straight to the screens memory location. This means that you must tell IntOS to consider the screen you are using as a Bitmap and **NOT** a normal screen. I know this seems a bit strange, but this system is much faster. Before you use this command you should see the 'IN_S_BITMAP' command.

Also See: IN_S_BITMAP

IN_WINDOW

command:	to open an intuition window
syntax:	IN_WINDOW_[num , x , y , w , h , f , t$, gl]
turbo:	@Window{num;x;y;w;h;f;t$;gl;}

This command will open an Intuition Window onto a previously opened screen. Please note that this command can ONLY be issued AFTER a screen command, and ALL other window dependant commands can only be issued AFTER the _WINDOW command. Otherwise errors may result!

Upto 64 windows can be opened (0-63) f = Flags

x	=	x position
y	=	*y* position
w	=	width of window
h	=	height of window

1	$1	=	Attaches the sizing gadgets
2	$2	=	Allows the window to be 'dragged' by the mouse
4	$4	=	Allows 'depth' gadgets
8	$8	=	Allows for the 'Close' gadget
16	$10	=	With '$400' and '$1' set, this will leave the right hand margin, the width of the sizing gadget, clear. And any drawing to the window will not extend over this margin.
32	$20	=	Same as '$10' except its for the bottom margin
256	$100	=	Forces your window to the back of the display. You CAN **NOT** have '$4' set at the same time.
1024	$400	=	This keeps the windows border separate from the rest of the window. Convenient, but memory hungry.
2048	$800	=	Opens a window without Borders
4096	$1000	=	Activates the window after opening.

So... To open a window with Depth, Sizing & Close gadgets and to have it activated upon opening, the number would be...

```
        $100E
    or  4110
```

So... IN_WINDOW_[0,0,0,640,256,$100E,-1]

or Window{0;0;0;640;256;4110;-1;}

The most common of the flags is '$100E' (4110) (Depth, Close, Activation, dragging etc. etc.)

GL is the gadget list to assign to the window. If you haven't created one, or you don't want to add one to the window. Then give the value of '-1'.

To open a Window on WorkBench....

```
'
IN_WB_TO_SCREEN_[0
'
IN_WINDOW_[0,160,64,320,128,$100E,"Hello",-1]
'
IN_WAIT_RAT
'
```

or

```
T$=T$+"@WB_To_Screen{0;}"
T$=T$+"@Window{0;160;64;320;128;4110;Hello;-1;}"
'
IN_TURBO_[T$]
'
IN_WAIT_RAT
'
```

Also See: IN_WB_TO_SCREEN

IN_WAIT_RAT

IN_W_USE

command: tell program to use a different window

syntax: IN_W_USE_[num]

turbo: @W_Use{num;}

This command allows you to switch program flow between the programs already opened windows.

IN_W_CLOSE

command: closing a previously opened window

syntax: IN_W_CLOSE_[num]

turbo: @W_Close{Num;}

This window will close an already opened window.

IN_W_INPUT

command: tell program to get its input from a window

syntax: IN_W_INPUT_[num]

turbo: @W_Input{Num;}

This command forces your program to get any future inputs from the specified window.

Also See: IN_W_OUTPUT

IN_W_OUTPUT

command: force program to send all outputs to a window

syntax: IN_W_OUTPUT_[num]

turbo: @W_Output{Num;}

This command will force all outputs to a specified window.

Also See: IN_W_INPUT

IN_W_ACTIVATE

command: activating a newly opened window

syntax: IN_W_ACTIVATE_[num]

This command should be issued before you use ANY newly opened window which has not had the '$1000' flag set when it was opened.

IN_MENUS

command: turn on or off previously attached menu's

syntax: IN_MENUS_[True/False]

This command will turn on or off any menu's which have been attached to the current window.

To turn on a menu = True

To turn off a menu = False

IN_W_MOVE

command: moving a window

syntax: IN_W_MOVE_[x, y]

turbo: @W_Move{x;y;}

This command will move a previously opened window to the specified position on a screen.

NOTE:- The X position and the width of the window MUST **NOT** be greater than the width of the screen!

i.e. Xpos+WindowWidth MUST **NOT** be greater than ScreenWidth!! The y position and the height of the window **MUST NOT** be greater than the height of the screen!

i.e. ypos+Windowheight **MUST NOT** be greater than Screenheight!!

IN_W_SIZE

command: resize a window

syntax: IN_W_SIZE_[w, h]

turbo: @W_Size{w;h;}

This command will resize the current window.

PLEASE NOTE! Please make sure that the windows width, height, as well as Xpos and Ypos equal no more than the width and height of the screen its opened on. Otherwise an error will occur.

IN_W_RAT_X

function: gives mouses x pos

syntax: IN_W_RAT_X

result: xpos=Param$

This function returns the current x position of the mouse in the current activated window.

Also see: IN_W_RAT_Y

IN_W_RAT_Y

function: gives mouses y pos

syntax: IN_W_RAT_Y

result: ypos=Param$

This function returns the current y position of the mouse in the current activated window.

Also see: IN_W_RAT_X

IN_W_CURS_X

function: return x pos of cursor

syntax: IN_W_CURS_X

result: xpos=Param$

This command will return the current X position of the cursor in the current activated window.

Also see: IN_W_CURS_Y

IN_W_CURS_Y

function: return y pos of cursor

syntax: IN_W_CURS_Y

result: ypos=Param$

This command will return the current Y position of the cursor in the current activated window.

Also see: IN_W_CURS_X

IN_W_LOCATE

command: to re-locate the window cursor

syntax: IN_W_LOCATE_[x, y]

turbo: @W_Locate{x;y;}

This command is similar to the AMOS 'Gr Locate' command. The difference is that this works with Windows.

IN_W_POS_X

function: return x pos of current window

syntax: IN_W_POS_X

result: xpos=Param$

This function will return the windows current x offset from the edge of the current screen.

Also see: IN_W_POS_Y

IN_W_POS_Y

function: return y pos of current window

syntax: IN_W_POS_Y

result: ypos=Param$

This function will return the windows current y offset from the edge of the current screen.

Also see: IN_W_POS_X

IN_W_WIDTH

function: return width of window

syntax: IN_W_WIDTH

result: wid=Param$

This function will return the width of the current activated window.

Also see: IN_W_HEIGHT

IN_W_HEIGHT

function: return the height of window

syntax: IN_W_HEIGHT

result: hgt=Param$

This function will return the height of the current activated window

Also see: IN_W_WIDTH

IN_W_INNER_WIDTH

function: return the width of a window

syntax: IN_W_INNER_WIDTH

result: wid=Param$

This function will return the width minus the width of the borders of the current activated window.

Also see: IN_W_INNER_H

IN_W_INNER_H

function: return the width of window

syntax: IN_W_INNER_H

result: hgt=Param$

This function will return the height minus the height of the borders of the current activated window.

Also see: IN_W_INNER_WIDTH

IN_W_TOP_OFF

function: return width of window top bar

syntax: IN_W_TOP_OFF

result: ypos=Param$

This returns the height of the current windows 'drag' bar.

Also see: IN_W_LEFT_OFF

IN_W_LEFT_OFF

function: give width of left hand side border of window

syntax: IN_W_LEFT_OFF

result: xpos=Param$

This function returns the width of the current activated windows left border.

Also see: IN_W_TOP_OFF

IN_W_SIZE_LIMITS

command: To limit the size of window.

syntax: IN_W_SIZE_LIMITS_[w1,h1,w2,h2]

turbo: @W_Size_Limits{w1;h1;w2;h2;}

This command is used to define the minimum & maximum size of a window.

w1 = Minimum Width
h1 = Minimum Height
w2 = Maximum Width
h2 = Maximum Height

IN_RASTPORT

function: return address of current window

syntax: IN_RASTPORT_[W]

result: addr=Param$

This command is of most use to people who insist on messing with the Amiga's system. It gives the address of the start of a specified window.

W = Window number

IN_W_FRAME

command: for presentation purposes.

syntax: IN_W_FRAME_[x1,y1,x2,y2,c1,c2]

turbo: @W_Frame{x1;v1:x2;y2;c1;c2;}

This box can help greatly in program presentation.

x1	=	First X co-ord
y1	=	First Y co-ord
x2	=	Second X co-ord
y2	=	Second Y co-ord
c1	=	highlight colours
c2	=	highlight colours

IN_WAIT_EVENT

function: halt program until AmigaDOS event occurs

syntax: IN_WAIT_EVENT

result: EV=Param

This command halts ALL program flow, IntOS's as well as Amos's until a Window event occurs. These are as follows...

Dec	Hex		
2	$2	=	Report when a window's size has changed
4	$4	=	Report when a windows display has been corrupted
8	$8	=	Report when a mouse button has been pressed
16	$10	=	Report when mouse has moved
32	$20	=	Report when a gadget has been pressed
64	$40	=	Report when a gadget has been released
256	$100	=	Report when a menu option has been chosen
512	$200	=	Report when a windows close gadget has been pressed
1024	$400	=	Report when a key press has been issued
32768	$8000	=	Report when a disk has been inserted
65536	$10000	=	Report when a disk has been removed
262144	$40000	=	Report when a window has been activated
524288	$80000	=	Report when a window has been de-activated

Its best to use the hexadecimal numbers as they are easier to remember and use.

Also See: IN_EVENT
IN_W_EVENT
IN_GADGET_HIT
IN_MENU_HIT
IN_ITEM_HIT
IN_SUBITEM_HIT
IN_RAT_BUTTONS
IN_EVENT_RAT_X
IN_EVENT_RAT_Y

IN_EVENT

function: get window event

syntax: IN_EVENT

result: EV=Param

The same as IN_WAIT_EVENT except that it does NOT stop program flow.
Refer to IN_WAIT_EVENT for list of events

Also See: IN_EVENT
IN_W_EVENT
IN_GADGET_HIT
IN_MENU_HIT
IN_ITEM_HIT
IN_SUBITEM_HIT
IN_RAT_BUTTONS
IN_EVENT_RAT_X
IN_EVENT_RAT_Y

IN_W_EVENT

function: report window last event occurred in

syntax: IN_W_EVENT

result: win=Param

Report which window the current Event occurred

Also See: IN_EVENT
IN_W_EVENT
IN_GADGET_HIT
IN_MENU_HIT
IN_ITEM_HIT
IN_SUBITEM_HIT
IN_RAT_BUTTONS
IN_EVENT_RAT_X
IN_EVENT_RAT_Y

IN_GADGET_HIT

function: report which gadget has been pressed

syntax: IN_GADGET_HIT

result: EV=Param

Report the gadget number pressed. This should be used in conjunction with the event flags $20 (32) or $40 (64).

Also See: IN_EVENT

IN_W_EVENT

IN_GADGET_HIT

IN_MENU_HIT

IN_ITEM_HIT

IN_SUBITEM_HIT

IN_RAT_BUTTONS

IN_EVENT_RAT_X

IN_EVENT_RAT_Y

IN_MENU_HIT

function: report which menu has been accessed

syntax: IN_MENU_HIT

result: menu=Param

Report the menu number just accessed. This should be used in conjunction with the event flag $100 (256)

Also See: IN_EVENT

IN_W_EVENT

IN_GADGET_HIT

IN_MENU_HIT

IN_ITEM_HIT

IN_SUBITEM_HIT

IN_RAT_BUTTONS

IN_EVENT_RAT_X

IN_EVENT_RAT_Y

IN_ITEM_HIT

function: report which menu item has been accessed

syntax: IN_ITEM_HIT

result: EV=Param

Report the item number just accessed. This should be used in conjunction with the event flag $100 (256) and the command IN_MENU_HIT

Also See: IN_EVENT

IN_W_EVENT

IN_GADGET_HIT

IN_MENU_HIT

IN_ITEM_HIT

IN_SUBITEM_HIT

IN_RAT_BUTTONS

IN_EVENT_RAT_X

IN_EVENT_RAT_Y

IN_SUBITEM_HIT

function: report which subitem has been accessed

syntax: IN_SUBITEM_HIT

result: EV=Param

Report the subitem number just accessed. This should be used in conjunction with the event flag $100 (256) and the commands IN_MENU_HIT and IN_ITEM_HIT

Also See: IN_EVENT

IN_W_EVENT

IN_GADGET_HIT

IN_MENU_HIT

IN_ITEM_HIT

IN_SUBITEM_HIT

IN_RAT_BUTTONS

IN_EVENT_RAT_X

IN_EVENT_RAT_Y

IN_RAT_BUTTONS

function: report which mouse buttons have been pressed after an event

syntax: IN_RAT_BUTTONS

result: button=Param

This function will allow the program to find out which mouse button was used (if at all) in the previous event.

	down	**up**
Left	1	5
Right	2	6

Also See: IN_EVENT

IN_W_EVENT

IN_GADGET_HIT

IN_MENU_HIT

IN_ITEM_HIT

IN_SUBITEM_HIT

IN_RAT_BUTTONS

IN_EVENT_RAT_X

IN_EVENT_RAT_Y

IN_EVENT_RAT_X

function: report mouse position after an event

syntax: IN_EVENT_RAT_X

result: EV=Param

This function will return the mouse's X position when the last event occurred.

Also See: IN_W_EVENT

IN_GADGET_HIT

IN_MENU_HIT

IN_ITEM_HIT

IN_SUBITEM_HIT

IN_RAT_BUTTONS

IN_EVENT_RAT_X

IN_EVENT_RAT_Y

IN_EVENT_RAT_Y

function: report mouse position after an event

syntax: IN_EVENT_RAT_Y

result: EV=Param

This function will return the mouse's Y position when the last event occurred.

Also See: IN_EVENT

IN_W_EVENT

IN_GADGET_HIT

IN_MENU_HIT

IN_ITEM_HIT

IN_SUBITEM_HIT

IN_RAT_BUTTONS

IN_EVENT_RAT_X

IN_EVENT_RAT_Y

IN_W_PLOT

command: plot a pixel in current window

syntax: IN_W_PLOT_[x,y,c]

turbo: @W_Plot{x;y;c;}

This command is very similar to the Amos Plot command. The difference is that the colour **MUST** be given and it is **NOT** an option.

x	=	x pos
y	=	y pos
c	=	colour

IN_W_BOX

command: to draw a box inside a window

syntax: IN_W_BOX_[x1,y1,x2,y2,c]

turbo: @W_Box{x1;y1;x2;y2;c;}

This command will draw a rectangular box to the current window. This command is similar to the Amos version except that the colour it uses **MUST** be issued.

x1	=	first x pos
y1	=	first y pos
x2	=	second x pos
y2	=	second y pos
c	=	colour

IN_W_CIRCLE

command: to draw a circle

syntax: IN_W_CIRCLE_[x1,y1,r,c]

turbo: @W_Circle{x1;y1;r;c;}

This command will draw a circle in the current active window.
This is similar to the AMOS version, except the colour **MUST** be issued.

x1	=	x pos
y1	=	y pos
x2	=	Radius
c	=	colour

IN_W_ELLIPSE

command: to draw an ellipse

syntax: IN_W_ELLIPSE_[x1,y1,r1,r2,c]

turbo: @W_Ellipse{x1;y1;r1;r2;c;}

This command will draw an ellipse in the current active window.
This is similar to the AMOS version, except the colour **MUST** be issued.

x1	=	x pos
y1	=	y pos
r1	=	width radius
r2	=	height radius
c	=	colour

IN_W_DRAW

command: to draw a line

syntax: IN_W_DRAW_[x1,y1,x2,y2,c]

turbo: @W_Draw{x1;y1;x2;y2;c;}

This command will draw line in the current activated window. This command is similar to the AMOS version, except a colour **MUST** be issued!

x1	=	first x pos
y1	=	first y pos
x2	=	second x pos
y2	=	second y pos
c	=	colour

IN_W_CLS

command: to clear a window

syntax: IN_W_CLS_[c]

turbo: @W_Cls{c;}

This command completely clears the currently active window with the given colour.

IN_W_INNER_CLS

command: to clear a window

syntax: IN_W_INNER_CLS_[c]

turbo: @W_Inner_Cls{c;}

This command will clear anything within the currently active window, but the windows borders will remain intact.

IN_W_SCROLL

command: to scroll a window area

syntax: IN_W_SCROLL_[x1,y1,x2,y2,dx,dy]

This command will scroll a rectangular window area by the delta's X & Y in the current active screen.

x1	=	first x pos
y1	=	first y pos
x2	=	second x pos
y2	=	second y pos
dx	=	X Delta (minus numbers scroll left)
dy	=	Y Delta (Minus numbers scroll up)

IN_W_COLOUR

command: set foreground and background pens

syntax: IN_W_COLOUR_[fc,bc]

turbo: @W_Colour{fc;bc;}

This command does basically the same as both the AMOS commands PEN and PAPER in one command. This command will set the currently active windows pens.

fc = foreground colour
bc = background colour

IN_W_JAM

command: change a window GFX mode

syntax: IN_W_JAM_[jm]

turbo: @W_Jam{jm;}

This command sets the currently active window's GFX mode...

jam...
0 = Only foreground colour will be drawn
1 = Draws both Foreground and background colour
2 = Inverse
4 = Inverse characters

Gadget Flag settings:

BIT			text	string	prop
0	=	Toggle on/off	yes	no	no
1	=	Relative to right of window	yes	yes	yes
2	=	Relative to bottom of window	yes	yes	yes
3	=	Size relative to width of window	no	no	yes
4	=	Size relative to height of window	no	no	yes
5	=	Box select	yes	yes	yes
6	=	Prop gadget has horizontal movement	no	no	yes
7	=	Prop has vertical movement	no	no	yes
8	=	Prop has no border	no	no	yes

IN_BORDER_PENS

command: to change highlight colours

syntax: IN_BORDER_PENS_[hc,sc]

turbo: @Border_Pens{hc;sc;}

Allows you to change gadget borders.

hc = highlight colour

sc = shadow colour

IN_TEXT_GADGET

command: a simple text gadget (button)

syntax: IN_TEXT_GADGET_[gl,x,y,f,num,txt,cw]

turbo: @Text_Gadget{gl;x;y;f;num;txt;cw;}

Defines a simple text gadget within a list. This list can be assigned to a window when one is opened.

```
'
' Define a gadget into list '0'
IN_TEXT_GADGET_[0,20,20,0,1,"Hello",32]
'
' open a window and attach gadget list '0'
IN_WINDOW_[0,0,12,320,200,$100E,"Hello",0]
'
```

IN_GADGET_PENS

command: change text colours within gadgets

syntax: IN_GADGET_PENS_[fc,bc]

turbo: @Gadget_Pens{fc;bc;}

This command will change the colours of the text used by the gadgets

fc	=	foreground colour
bc	=	background colour

IN_GADGET_JAM

command: change text GFX mode

syntax: IN_GADGET_JAM_[mode]

turbo: @Gadget_Jam{mode;}

This command will create different FX with any drawing command.

0	=	Draws only foreground colour
1	=	Draws both foreground and background colours
2	=	Inverse's
4	=	Inverse characters

IN_TOGGLE

command: toggle text gadget

syntax: IN_TOGGLE_[gl,num,on/off]

turbo: @Toggle{gl;num;on/off;}

Will switch On or Off a text gadget.

gl = List of gadgets

num = Number of gadget

on/off = On ... -1 (true)

Off... 0 (false)

IN_STRING_GADGET

command: create a text entry gadget.

syntax: IN_STRING_GADGET_[gl,x,y,f,num,ml,w]

turbo: @String_Gadget{gl;x;y;f;num;ml;w;}

String gadget allows you to create a text entry box in your window. This command will only define the gadget, but will not automatically place text into it, the IN_SET_STRING function must be used to do this.

gl = list of gadgets

x = X co-ord

y = Y co-ord

f = flags

num = number of gadget

ml = maximum length of text entry in characters-1

w = width of gadget in pixels

IN_STRING_TEXT

function: get text from string gadget

syntax: IN_STRING_TEXT_[gl,num]

result: $=Param$

Use this function to gain the contents of the specified string text gadget.

gl = list of gadgets

num = number of gadget

IN_ACTIVATE_STRING

command: to activate automatically a stringtext gadget

syntax: IN_ACTIVATE_STRING_[gl,num]

This command will activate a specified StringText gadget automatically without the programs user having to click on the StringGadget in question.

gl = list of gadget

num = number of gadget

IN_RESET_STRING

command: to reset the cursor

syntax: IN_RESET_STRING_[gl,num]

This command will place the cursor in a StringGadget at the left most position in the text.

gl = list of gadgets

num = number of gadget

IN_CLEAR_STRING

command: to clear a stringtext gadget

syntax: IN_CLEAR_STRING_[gl,num]

This command will clear a specified StringGadget of its entire contents

gl = list of gadgets

num = number of gadget

IN_SET_STRING

command: to place text

syntax: IN_SET_STRING_[gl,num,txt$]

This command allows you to place text into a specified StringGadget.

NOTE:- The text string must **NOT** be greater than the gadgets Maximum character capacity!!

gl = list of gadgets

num = number of gadget

txt$ = Text string

IN_PROP_GADGET

command: proportional gadgets

syntax: IN_PROP_GADGET_[gl,x,y,f,num,w,h]

turbo: @Prop_Gadget{gl;x;y;f;num;w;h;}

NOTE:- I have had no real experience of proportional gadgets, but I know the code to get them to work. So, for convenience I have tried to keep their syntax close to another well-known language. But this explanation (such as it is) is the best I can do

'Proportional gadget' is just a fancy name for a 'slider bar'. Proportional gadgets have 2 qualities. A potentiometer (pot for short) and a 'body' setting. A 'Pot' refers to the position of the slider bar within the gadget, the values it contains are between 0 & 1. So, for instance when the gadget is exactly half way, the 'Pot' value would be '0.5'.

Body is a bit of a mystery. But it should also be between 0 & 1.

As I am not proficient with this form of gadget, and the fact that I do not know anyone who is, no examples are offered... Sorry, you will have to do your best.

gl	=	list of gadgets
x	=	x pos
y	=	y pos
f	=	flags
num	=	number
w	=	width
h	=	height

NOTE:- Because proportional gadgets use floating point numbers, it may be a good idea to use them.

IN_SET_H_PROP

command: to after prop gadget

syntax: IN_SET_H_PROP_[gl,num,pot,body]

turbo: @Set_H_Prop{gl;num;pot;body;}

Alter or set the horizontal position of a proportional gadget. IN_REDRAW must be issued afterwards for it to take ANY effect!

gl	=	List of gadgets
num	=	Number of gadget
pot	=	Pot
body	=	Body

IN_SET_V_PROP

command: to alter prop gadget

syntax: IN_SET_V_PROP_[gl,num,pot,body]

turbo: @Set_V_Prop{gl;num;pot;body;}

Alter or set the verticle position of a proportional gadget. IN_REDRAW must be issued afterwards for it to take ANY effect!

gl	=	list of gadgets
num	=	Number of gadget
pot	=	pot
body	=	body

IN_H_PROP_POT

function: to read current pot setting

syntax: IN_H_PROP_POT_[gl,num]

result: pot#=Param#

This function returns the current pot of a horizontal proportional gadget. This will return any value between 0 and up to but not including 1.

IN_H_PROP_BODY

function: to read current body value

syntax: IN_H_PROP_BODY_[gl,num]

result: body#=Param#

This will return the gadgets current horizontal Body setting.

IN_V_PROP_POT

function: to read current pot setting

syntax: IN_V_PROP_POT_[gl,num]

result: pot#=Param#

This function returns the current pot of a verticle proportional gadget. This will return any value between 0 and up to but not including 1.

IN_V_PROP_BODY

function: to read current body value

syntax: IN_V_PROP_BODY_[gl,num]

result: body#=Param#

This will return the gadgets current verticle Body setting.

IN_REDRAW

command: to redraw any alterations

syntax: IN_REDRAW_[win,num]

This command will redraw a specified gadget within a specified window.

This command should be used if the program itself is to alter StringGadgets or proportional gadgets.

win = Window

num = number of gadget

IN_BORDERS

command: change borders

syntax: IN_BORDERS_[w,h]

turbo: @Borders{w;h;}

This command is used to specify the width and height of the spacing of the borders from the text in TextGadget's or String Gadget's.

w = width

h = Height

IN_GADGET_BORDERS

command: presentation

syntax: IN_GADGET_BORDERS_[x,y,w,h]

turbo: @Gadget_Borders{x;y;w;h;}

This command is purely for presentation purposes, it is similar to the IN_W_FRAME command. Please note that this command is effected by the IN_BORDERS command!

x	=	x pos
y	=	y pos
w	=	width
h	=	height

IN_M_TITLE

command: to create a menu

syntax: IN_M_TITLE_[ml,num,txt$]

turbo: @M_Title{ml;num;txt;}

These commands allow you to create standard AmigaDOS pull-down menu's. To attach them to a window, you must use the IN_M_SET command.

ml = Menulist

num = Number of menu (0 - ?)

txt = What the actual title is to be

IN_M_ITEM

command: define a menu option

syntax: IN_M_ITEM_[ml,f,m,num,txt$,sc$]

turbo: @M_Item{ml;f;m;num;txt$;sc$;}

This command is used to define the contents of a menu.

ml = Menu List

f = Flags. These flags are simple to use...

0 = Normal 'Select' flag

1 = Toggle option. (Identified with a tick sign)

2 = Mutually exclusive toggle

3 = Same as 1, except menu option will automatically be 'on'

4 = Same as 2, except a specified option will be 'on'

m = Menu number

num = Menu Item number

txt$ = Text to be displayed

sc$ = shortcut character.

NOTE:- A question mark'?' must be issued if there is no shortcut option!.

IN_M_SUB_ITEM

command: Sub menu option

syntax: IN_M_SUB_Item_[ml,f,m,num,i,txt$,sc$]

turbo: @M_Sub_Item{ml;f;m;num;i;txt$;sc$;}

This command is used to define sub menu options.

i = sub item number

See IN_M_ITEM for other parameters and flag settings.

IN_M_SET

command: attach a menu

syntax: IN_M_SET_[ml]

turbo: @M_Set{ml;}

This command will attach a pre-defined menu list to the currently active window.

IN_M_GAP

command: set gaps

syntax: IN_M_GAP_[x,y]

turbo: @M_Gap{x;y;}

This command allows you to control the presentation of your menu's.

x = is the amount of pixels to the left and right of any item or subitems.

y = is the amount of pixels at the top and bottom of any item or subitem

IN_M_SUB_ITEM_OFFSET

command: position subitems

syntax: IN_M_SUB_ITEM_OFFSET_[x,y]

turbo: @S_Item_Off{x;y;}

Allows to set the relative position of subitems in relation to their associated menu option.

IN_M_STATE

command: toggle menu's

syntax: IN_M_STATE_[ml,m,i,si,on/off]

This command allows you to turn on or off certain menu options. This command is hard to explain, so here goes...

If you want to turn off the whole menu (Menulist 0)...

IN_M_STATE_[0,-1,-1,-1, False]

If you want to turn off menu 2 under menulist 0

IN_M_STATE_[0,2,-1,-1,False]

If you want to turn ON item 4, under menu 3 under menulist 0

IN_M_STATE_[0,3,4,-1,True]

If you want to turn OFF subitem 2, under item 1 under menu 2, under menulist 2

IN_M_STATE_[2,2,1,2, False]

IN_M_COLOUR

command: menu text colour

syntax: IN_M_COLOUR_[c]

turbo: @M_Clr{c;}

Determine the colour of the text of the menu.

IN_M_CHECKED

function: check 'toggle' gadget

syntax: IN_M_CHECKED_[ml,m,i,si]

result: toggled?=Param

This command should be used to determine whether or not a menu option has been toggled ON (-1 or True) or OFF (0 or False). This command will only check a menu option with a flag setting of 1 or more.

To check a menu item... (menulist 1, menu 3, item 2)

```
IN_M_CHECKED_[1,3,2,-1]
RES=Param
```

To Check a menu subitem... (menulist 2, Menu 1, item 3, subitem 2)

```
IN_M_CHECKED_[2,1,3,2]
RES=Param
```

IntOS is quite a simple beast that relies on reserved memory storage area's for communication. The 4 variables which are global, do the following...

_INTOS_MODE = Holds the command number to issue

_INTOS_LOCA = Holds the location of the command parameters in memory

_INTOS_SEMA = Acts as the communications base between AMOS and IntOS...

1 = IntOS is processing commands
2 = IntOS is ready to receive commands
3 = IntOS has found a fatal error

_INTOS_COMM() = This array holds the parameter information, the location of it is held under _INTOS_LOCA. It may seem strange that I have to issue the memory location of this every time a command is issued, but AMOS keeps moving its Array locations all over the place!

The other variables IntOS uses are simply for passing information.
These are...

_INTOS0$** to **_INTOS7$

and

_INTOS0 to _INTOS7

They are pretty distinctive. You shouldn't have much trouble in avoiding them.

Although IntOS is not written by the original Amos Author you should not let this dissuade you from using IntOS, as IntOS can do **EVERYTHING!** IntOS is a powerful development add-on.

As IntOS is a custom library, please don't expect it to do feats of record breaking speed! IntOS has to compete with the rules laid down by AMOS, as well as the rules laid down by AmigaDOS. That coupled with the fact that AmigaDOS is quite slow anyway, means that when IntOS comes to the crunch, it will be just as fast as any other Intuition system in the future.

IntOS is still faster in general than any other interpreted BASIC language! So please remember all of you AMOS Compiler owners out there... AMOS is primarily an Interpreted language!

INDEX

OTM Publication & promotions Ltd will publish your software - should you have a program please do not hesitate to contact us - we are here to help you get the best possible reward for your ability.

Company Reg. No. 2972194

OTM wishes to thank Mr.M.Warren for allowing us to produce this program and everyone else involved.